Universidade Federal Fluminense
Instituto de Computação

Uso da Transformada de Hough na Vetorização de Moldes e Outras Aplicações

Maysa Malfiza Garcia de Macedo

Orientadora: Aura Conci

Niterói –RJ
Fevereiro de 2005

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

M141 Macedo, Maysa Malfiza Garcia de
Uso da transformada de hough na vetorização de moldes e outras aplicações / Maysa Malfiza Garcia de Macedo. – Niterói,. RJ : [s.n.], 2005.
109 f.

Orientador: Aura Conci.
Dissertação (Mestrado em Ciência da Computação) - Universidade Federal Fluminense, 2005.

1. Processamento de imagens – Técnicas digitais. 2. Indústria do vestuário. I. Título.

CDD 006.42

Agradecimentos

Agradeço a Deus que em toda sua magnitude olha por mim todo o tempo, faz-me seguir sem medo e ajuda-me nas horas difíceis. Agradeço a meus pais, fonte de incentivo, carinho e paciência, estando ao meu lado por todo caminho. Agradeço a professora Aura, orientadora e amiga, que através das críticas e elogios fez-me capaz de pesquisar e estar apresentando esta dissertação de Mestrado. Agradeço também aos meus amigos que me incentivaram todo o tempo e comemoram comigo o cumprimento de mais uma etapa de minha vida acadêmica.

Resumo

A popularização das imagens digitais ampliou as opções de processamento das mesmas e com isso a possibilidade de extração de diversas informações.

Esta dissertação tem por objetivo dar subsídios à indústria de vestuário, na pesquisa de um sistema automático de identificação e vetorização de moldes a partir de uma imagem digital. A idéia principal é identificar formas compostas por segmentos de retas e curvas e posteriormente reconstruí-las a partir de um conjunto mínimo de informação. Com o objetivo de efetuar a vetorização automática ou reduzir o procedimento atual que envolve trabalho manual extenso através da utilização de uma mesa digitalizadora.

Para isso foi desenvolvido um sistema baseado em uma metodologia que aplica a transformada de Hough de equações matemáticas de primeiro e segundo grau a fim de identificarmos formas genéricas que possam estar presentes na imagem.

Ao longo do desenvolvimento desta dissertação outras funcionalidades do sistema desenvolvido foram descobertas além da vetorização, entre elas, surge como mais uma alternativa de medição na aplicação de teste de dureza e também como método de controle de qualidade na área de medicamentos. Os testes e exemplos apresentados confirmam a eficácia tanto do sistema quanto da metodologia desenvolvida.

Abstract

The growing use of the digital images increases the necessity of new approaches on applications on how information from acquired captured images can be obtained.

The principal objective of this line of works is aid modeling model on apparel industry. The main point is to identify on a pattern of garment the lines and curves that forms its border and than to reconstruct it from a minimum set of information.

The aim is improve the current procedure that involves extensive manual work on digitalization of the 2D patterns. In this work a methodology that applies the Hough Transform for equations of first and second degrees is proposed, implemented and tested. This system can be used in several other applications to identify forms. In some uses as in the hardness measure and the quality control it presents an alternative way. The presented tests and examples illustrate the effectiveness of the developed system and methodology.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1- Introdução

A indústria de confecção necessita automatizar seus processos, e a busca de novas tecnologias é chave para se chegar à solução desejada. Nesta dissertação busca-se prosseguir na direção de uma proposta de substituição da mesa digitalizadora (dispositivo utilizado para o fornecimento de dados gráficos vetoriais, usual nesta indústria) por um mecanismo de vetorização automática.

A partir de um molde de roupa em uma imagem digital, é necessário extrair um conjunto mínimo de informações e armazená-lo para que posteriormente o mesmo molde possa ser reconstruído. Adicionalmente um molde é uma forma complexa que precisa ser identificada para que a vetorização possa acontecer.

Com o intuito de estabelecer uma metodologia para solucionar este problema foi iniciado em uma dissertação anterior [OLI2003] dessa linha de pesquisa, o estudo sobre um possível processo de vetorização utilizando a transformada de Hough. A atual dissertação vem acrescentar o trabalho anterior desenvolvendo um sistema para vetorizar moldes complexos a partir da aplicação comedida da transformada Hough para formas cônicas. Através da aplicação do sistema desenvolvido neste trabalho sob um molde é gerado um produto final que trata de um conjunto de características de cada forma cônica identificada na imagem digital.

1.1 Transformada de Hough

A Transformada de Hough foi desenvolvida por Paul Hough [HOU1962] no início dos anos 60. É uma técnica para reconhecimento, em imagens digitais, para que sejam facilmente parametrizadas, ou seja, que possuam uma equação com fórmula conhecida, tais como retas, círculos [CHE2001] e elipses [LEI1999], [TSU1978], entre outras. A partir da primeira

publicação, vieram muitos outros trabalhos melhorando, otimizando [XU1993] e até aplicando a transformada à formas generalizadas [CHA2004].

Para a aplicação desta transformada, normalmente é realizado um pré-processamento na imagem com o objetivo de identificar claramente os contornos dos elementos que a compõem.

A idéia da transformada de Hough é transformar a imagem do espaço digital **(x,y)** para uma representação na forma dos parâmetros descritos pela curva que se deseja encontrar na imagem. Esta transformação é aplicada de modo que todos os pontos pertencentes a uma mesma curva sejam mapeados em um único ponto no espaço dos parâmetros da curva procurada (**figura1.1**).

Para isto, o espaço dos parâmetros é discretizado e representado na forma de uma matriz de inteiros, onde cada posição da matriz corresponde a um intervalo no espaço real dos parâmetros. Cada ponto da imagem que satisfizer a equação da forma paramétrica procurada incrementa de uma unidade o contador correspondente a sua posição, na representação discretizada (matriz).

O contador que tiver, no final do processo, o valor mais alto, corresponderá aos parâmetros da curva descrita na imagem.

Figura 1.1- Etapas da aplicação da transformada de Hough para qualquer forma geométrica.

1.2 Desenvolvimento desta dissertação

O objetivo desta dissertação é vetorizar formas genéricas, semelhantes às usuais encontradas em moldes, aplicando a técnica de

transformada de Hough. Para isso partimos do pressuposto de que uma imagem complexa é constituída de formas geométricas simples, definida por combinações de equações de primeiro e segundo grau. Com isso, a união das detecções de cada forma primitiva se torna na realidade detecção de uma forma genérica.

Após a breve introdução do que é a transformada de Hough deste capítulo 1, poderá ser encontrado no próximo capítulo um estudo mais profundo sobre a transformada de Hough aplicada a retas. No capítulo 2 são mostrados todos os detalhes de funcionalidade da matriz acumuladora e como melhor discretizar o espaço para a detecção de retas, os algoritmos utilizados, e os testes de detecção de retas.

O capítulo 3 usa a transformada de Hough na detecção de cônicas. É mostrado como detectar essas formas mesmo com imagens apresentando falhas e ruídos. Exemplifica-se a aplicação da transformada de Hough para círculos, elipses e parábolas.

O capítulo 4 considera a união das detecções desenvolvidas no trabalho, como usá-las para detectar formas mais complexas através da detecção de várias formas simples e suas subtrações na imagem a ser vetorizada.

O capítulo 5 mostra exemplos de outras aplicações relacionadas à extração de características de uma imagem onde os algoritmos desenvolvidos neste trabalho podem ser empregados.

Conclusões relativas ao trabalho e idéias de como chegar ao objetivo final desta linha de pesquisa são encontradas no capítulo 6.

Capítulo 2- Identificação de retas

Dado um ponto (**x,y**) do $\mathbf{R^2}$ ou $\mathbf{Z^2}$, na forma discreta do vídeo, a equação geral das retas que passam por este ponto será:

$$y = mx + g \ ,$$

onde **m** corresponde ao coeficiente angular da reta e **g** é o valor do ponto onde há interseção com o eixo **y**. O plano **mg** é o espaço dos parâmetros onde pontos no espaço da imagem (**x,y)** são representados como retas, e o ponto de interseção dessas retas representa os valores **m** e **g** que identificam uma reta composta pelos mesmos parâmetros no espaço da imagem (**x,y)** [HOU1962]. A **figura 2.1** mostra essa forma de representação da reta no espaço da imagem (**x,y)** e seu mapeamento no espaço dos parâmetros (**m,g**).

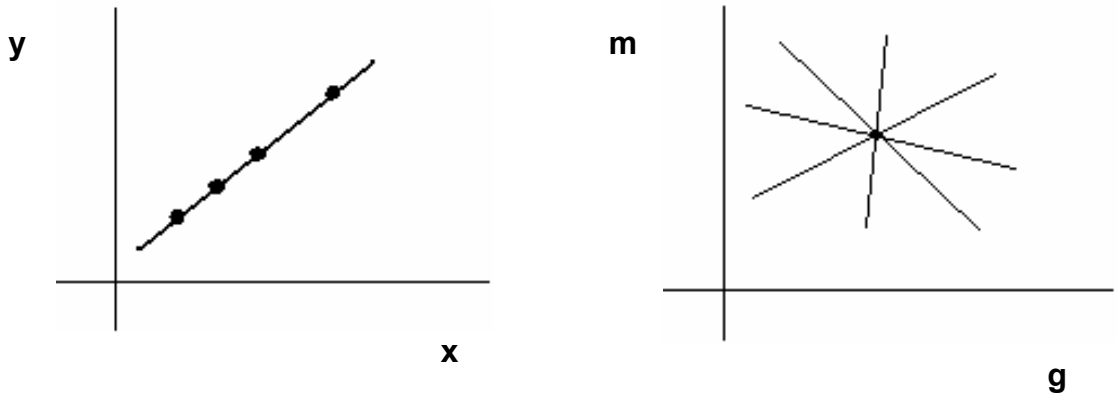

Figura 2.1- Cada ponto no espaço da imagem transforma-se em uma reta no espaço de parâmetros: $g = -mx + y$

Todos os pontos pertencentes a uma única reta interceptam-se em um único ponto que tem como coordenadas os parâmetros (**m,g**) da reta a ser identificada.

Um problema, quando essa equação de retas é usada, ocorre quando

deseja-se identificar retas verticais, pois no caso das retas verticais o coeficiente angular tende ao infinito. Para viabilizar a técnica, é proposta a utilização da equação da reta na forma polar:

$$\rho = x\cos\theta + y\sin\theta\,,$$

onde **ρ** é a distância da reta à origem e **θ** é o ângulo entre o eixo **x** e a normal desta reta, considerando positivo o sentido horário [DUD1972]. A **figura 2.2** mostra os elementos desta forma de descrição.

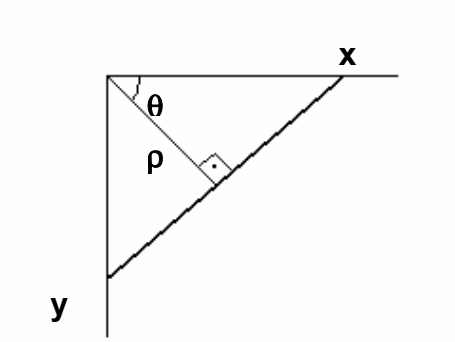

Figura 2.2- Esquema tradicional de representação de retas em coordenadas polares.

Utilizando a equação da reta na forma polar, cada ponto (**x**,**y**) passa a ser representado por (**ρ**,**θ**). Neste caso, qualquer ponto **P(x,y)** pertencente ao espaço da imagem, que está situado sobre a reta **R(ρ,θ)** tem os parâmetros **ρ,θ** constantes. O plano (**ρ,θ**) será o espaço dos parâmetros. Cada ponto **P(x,y)** de uma reta **R(ρ,θ)** é representado por uma senóide **S(ρ,θ)** no espaço dos parâmetros. A interseção das senóides no plano (**ρ,θ**) representa o valor **ρ** e **θ** de uma reta no espaço da imagem (**x,y**). Todos os pontos pertencentes à mesma reta interceptam-se em um único ponto **(ρ,θ)** no espaço dos parâmetros (**figura 2.3**).

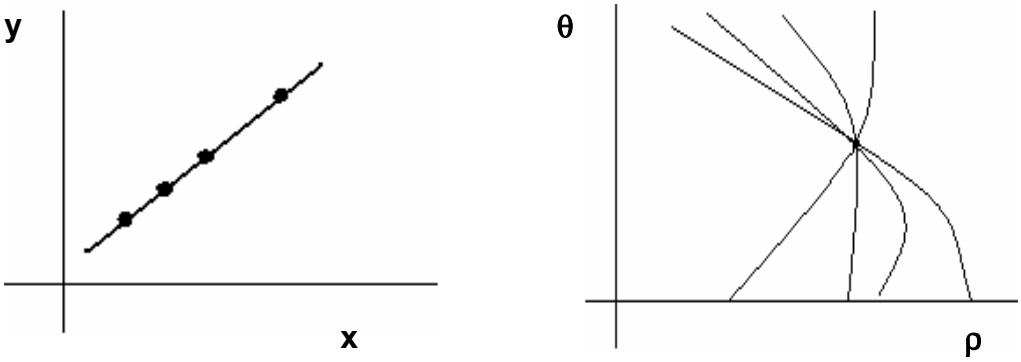

Figura 2.3- Cada ponto **P(x,y)** no espaço da imagem, corresponde a uma senóide **S(ρ,θ)** no espaço de parâmetros.

2.1- Discretização da matriz acumuladora

Para determinar o tamanho e os limites da matriz acumuladora, devem-se observar todos os parâmetros que compõem o espaço dos parâmetros. No caso das retas, é necessário ter a informação dos limites angulares de **θ** e dos valores limite de **ρ**. Neste trabalho utilizamos como padrão o sentido dos eixos e o intervalo mostrado na **figura 2.4** com valores de **θ** entre $\left(-90^{o},90^{o}\right]$ ou $\left(-\pi/2,\pi/2\right]$. Este intervalo abrange tanto retas com tangentes crescentes como retas com tangentes decrescentes. Nas **figuras 2.5** e **2.6** observa-se um exemplo de reta decrescente, onde **θ** apresenta-se negativo, devido ao intervalo padrão utilizado.

Para uma imagem descrita entre os limites **(0,0)** e **(N_1,N_2)** é estabelecido que o parâmetro **ρ** se apresente em um intervalo de $-\sqrt{N_1^2+N_2^2}$ a $\sqrt{N_1^2+N_2^2}$. O parâmetro **ρ** assume valores negativos quando **θ** apresenta valores negativos e **ysinθ** for maior do que os valores de **xcosθ**. Para entender melhor por que **ρ** pode assumir valores negativos vamos partir da fórmula:

$$\rho = x\cos\theta + y\sin\theta,$$

considerando valores negativos para **θ**, teremos; **θ =- |θ|** assim:

$$\rho = x\cos(-|\theta|) + y\sin(-|\theta|)$$

Como

$$\cos(-|\theta|) = \cos(|\theta|) \quad \text{e} \quad \sin(-|\theta|) = -\sin(|\theta|),$$

considera-se então:

$$\rho = x\cos|\theta| + y(-\sin|\theta|) \Rightarrow \rho = x\cos|\theta| - y\sin|\theta| \textbf{.}$$

Em imagens digitais **x**,**y** serão sempre positivos, assim:

$$\rho = x\cos|\theta| - y\sin|\theta| = |x\cos\theta| - |y\sin\theta|$$

Usando esta última fórmula, podemos chegar a duas condições:

Se

$$|x\cos\theta| < |y\sin\theta|, \text{ então: } \boldsymbol{\rho < 0},$$

ou seja, **ρ** assume valores entre $-\sqrt{N_1^2 + N_2^2}$ e **0** , mas se

$$|x\cos\theta| > |y\sin\theta|, \text{ então: } \boldsymbol{\rho > 0},$$

ou seja , **ρ** assume valores entre **0** e $\sqrt{N_1^2 + N_2^2}$.

Através da argumentação acima podemos notar porque no intervalo angular usado, a transformada de Hough pode apresentar parâmetro de distância negativa. A **figura 2.6** mostra um caso em que **ρ** é negativo .

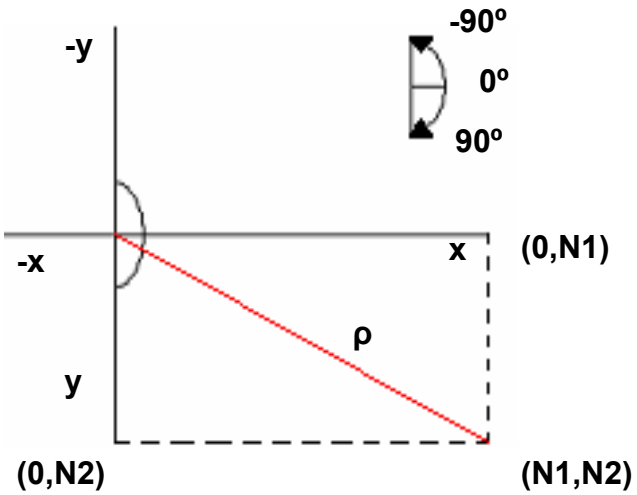

Figura 2.4- Limites de θ e ρ em uma imagem definida entre (0,0) e (N1,N2). Direção considerada positiva dos eixos x y neste trabalho.

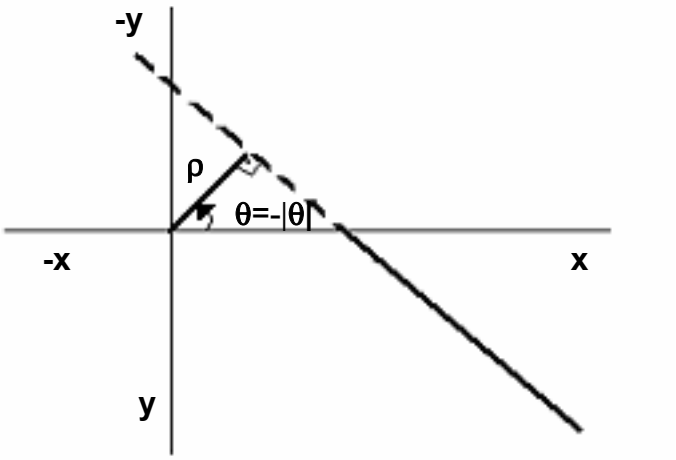

Figura 2.5- Exemplo de comportamento dos parâmetros **ρ** e **θ** em retas decrescentes.

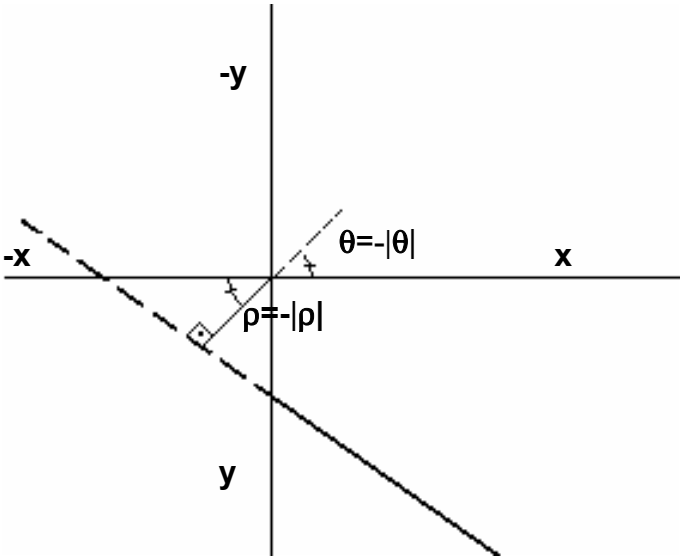

Figura 2.6- Exemplo de reta decrescente, com **ρ** negativo.

Note que na **figura 2.6**, o ângulo **θ** é mapeado para o intervalo padrão $(-90^o, 90^o]$ ou $(-\pi/2, \pi/2]$ o que produz **ρ** com valores negativos.

A precisão da detecção das retas varia conforme os intervalos e os incrementos estabelecidos tanto para **θ** quanto para **ρ**, ou seja, conforme **Δθ e Δρ**. Por exemplo, se o intervalo de **θ** é $(-90^o, 90^o]$ ou $(-\pi/2, \pi/2]$ então existem 180 células na matriz acumuladora para discretização de ângulo de grau em grau, mas se a discretização é setada de 4 em 4 graus**,** quer dizer que ao invés de 180 células a matriz pode ter apenas 45 células.

2.2 -Algoritmo de Hough para retas

O algoritmo da transformada de Hough para retas usando a forma polar pode ser descrito como segue, considerando:

I(x,y)-valor do pixel **(x,y)**da imagem em tom de cinza.
M(ρ,θ) – matriz de parâmetros

hough();
Inicializar matriz **M(ρ,θ)** com zero
Para cada **x,y** pertencente à imagem com **I(x,y)** =**1** *
Para **θ= -π/2+1** até **θ=π/2** com incremento de **Δθ**
$\rho = x\cos\theta + y\sin\theta$
Incrementar de uma unidade a célula **M(ρ,θ)** a matriz acumuladora
Faça laço até **θ** não pertencer mais ao intervalo $(-\pi/2, \pi/2]$
Faça laço até o final da imagem.

Algoritmo 2.1-Algoritmo de Hough para retas na forma polar. [HOU1962]

* É necessário aplicar um limiar (nível de cinza máximo) à imagem original para considerar se o pixel está aceso ou não. Se o pixel da imagem for maior que o nível de cinzamáximo permitido recebe 1 senão recebe zero.

No **algoritmo 2.1** é percorrida toda a imagem **I,** verificando se o pixel em questão está aceso. Caso esteja aceso, calcula os **ρ**´s possíveis para cada ângulo entre $(-90^o, 90^o]$ ou $(-\pi/2, \pi/2]$. Tendo-se tanto o **θ** quanto **ρ,** incrementa-se o valor da célula **M(ρ,θ)** de uma unidade. Quando o intervalo **Δρ**, ou seja, o tamanho da célula da matriz acumuladora for muito grande, são identificadas muitas retas falsas, já que num intervalo **Δρ** maior se abrigam mais retas em cada célula da matriz acumuladora, enquanto que quando se aplica um intervalo pequeno, a precisão se torna cada vez maior e as retas se distribuem pelas várias células existentes.

Ao final da varredura da imagem, teremos uma matriz preenchida com votos para cada par de coordenadas **M(ρ,θ)**. A célula com maior valor indica que seus índices são parâmetros da melhor reta. Caso existam **n** retas a serem identificadas, serão escolhidos os **n** maiores valores na matriz acumuladora.

2.3- Definição da reta pelos parâmetros encontrados

Antes de executar o processo inverso da transformada de Hough , que se resume a ir do espaço de parâmetros para o espaço de imagem, é necessário identificar o maior valor ou os maiores valores (quando se tratam de várias retas a serem identificadas) aplicando um limiar à matriz acumuladora para que seja resgatado somente a(s) reta(s) requerida(s). Mas como saber o valor que representa adequadamente as retas? Uma idéia para solucionar o problema, de qual valor deve ser dado ao limiar, é ordenar os maiores contadores de acordo com o número de retas a serem detectadas. O **algoritmo 2.2** mostra esta possibilidade.

Utilizando a matriz dos maiores votos, o limiar será o menor valor da matriz dos maiores votos. Dessa maneira será mostrado apenas o número de retas requeridas.

maiores[ ]-matriz que contém os maiores votos;
menor-menor valor presente na matriz **maiores[ ]**;
busca_menor-retorna o menor valor na matriz **maiores[ ]**;
aux-variável que recebe valor da matriz acumuladora.

```
buscar_maiores()
    Inicializar maiores[ ], menor
    menor=busca_menor();
    Para todo M(ρ,θ)
        Aux= M(ρ,θ)
        Se menor<aux então
        maiores[ ]=aux
        menor=busca_menor();
     Executar loop até o fim a matriz
```

Algoritmo 2.2- Ordenação dos contadores

Para executar o processo inverso da transformada de Hough, isto é, mostrar no espaço da imagem as retas detectadas, pode-se usar o procedimento

do **algoritmo 2.4** com o auxilio do **algoritmo 2.3**, neste procedimento considera-se que os elementos que seguem têm o significado abaixo:

IS(x,y)-imagem de saída;

x_menor e **y_menor**-coordenadas **x** e **y** do ponto inicial da reta;

x_maior e **y_maior**-coordenadas **x** e **y** do ponto final da reta;

cont-contador, se for igual a zero , significa o primeiro ponto da reta;

ap-aproximação, taxa de erro para encontrar pixel aceso;

busca_aceso-função que recebe **true** se através da aproximação **ap** encontrar um pixel aceso na imagem original, recebe **false** se não encontrar pixel aceso na vizinhança de **ap**;

w-contador;

linha-número de retas identificadas. Total de linhas que a matriz armazenadora de pontos iniciais e finais pode ter;

retas-matriz armazenadora de pontos iniciais e finais;

Ni,**Nj-** limites de **x** e **y** da imagem.

```
busca_aceso()
Variável ap recebe o tamanho da vizinhança, i e j
recebem as coordenadas do pixel de onde será calculada a
vizinhança.
 Para n= -ap até n= ap
    Para m= -ap até m= ap
        Se o pixel I(i+n , j+m) estiver aceso então
            busca_aceso=true, sai do procedimento
     Faça laço até m=ap
  Faça laço até n=ap
  busca_aceso=false
```

Algoritmo 2.3- algoritmo usado no **algoritmo 2.4**, linha 26,para encontrar pixel aceso caso ocorra um desvio **ap**

```
inversa_hough();
    Inicializar w,x_menor, x_maior,y_maior,y_menor
    Para cada célula M(ρ,θ)
        Se M(ρ,θ)==1* então
           Se sinθ==0 então
                Inicializar cont
                Para y=0 até y< Nj
                     x= ρ/cosθ
                     Se (x>0) && (x<Ni) então
                        Se I(x,y)==1 então
                            Se cont==0 então
                                x_menor=x, y_menor=y,
                               cont=cont+1
                            x_maior=x, y_maior=y
                            IS(x,y)=aceso
                 Executar Loop 6-14 até Nj da imagem
            Senão
                 Inicializar cont
                 Para x=0 até x<Ni
                      y = ρ − x cos θ/sin θ
                      Se (y<0) && (y<Ni) então
                          Se I(x,y)==1 então
                              Se cont==0 então
                                  x_menor=x, y_menor=y
                                  cont=cont+1
                              x_maior=x, y_maior=y
                              IS(x,y)=aceso
                          Senão
                              aceso=busca_aceso(ap,x,y)
                              Se (aceso==true) então
                                  Se (cont==0) então
                                      x_menor=x, y_menor=y
                                      cont=cont+1
                                 x_maior=x, y_maior=y
                                 IS(x,y)=aceso
                  Executar Loop até Ni
            Se (w<linha) então
                Retas[w*linha]=x_menor
                Retas[w* linha +1]=y_menor
                Retas[(w+1)* linha]=x_maior
                Retas[(w+1)* linha +1]=y_maior
                w=w+2
    Executar Loop até Ni
```

Algoritmo2.4- Definição das retas no espaço da Imagem

* É necessário aplicar um limiar (voto mínimo) à matriz acumuladora para considerar os parâmetros das retas mais votadas. Se o voto for maior que o voto mínimo a célula recebe 1 senão recebe 0.

Após a execução do **algoritmo 2.1**, temos uma matriz acumuladora preenchida com votos. A execução do **algoritmo 2.2** resulta na identificação do maior ou dos maiores votos. Em seguida aplica-se um limiar para que, na matriz acumuladora, todas as células de interesse apresentem valor 1 e as demais apresentem valor zero.

A célula **(ρ,θ)** com maior(es) voto(s) é identificada e em seguida extrai-se suas características para que possam ser calculados **x** e **y**.

Para cada célula da matriz acumuladora, com valor 1, se o **sinθ** for igual a zero (**algoritmo 2.4**) tem-se significado que a reta é horizontal, sendo utilizada a seguinte expressão, varrendo o parâmetro **y**:

$$x = \rho / \cos\theta$$

Caso **sinθ** seja diferente de zero (**algoritmo 4**), a reta pode ser vertical ou inclinada, sendo **x** e **y** calculados através da seguinte expressão, varrendo o parâmetro **x**:

$$y = \rho - x\cos\theta / \sin\theta$$

Após calculados **x** e **y** é verificado se, na imagem original, o pixel com as coordenadas (**x**,**y**) está aceso através do **algoritmo 2.3**. Esse procedimento é uma comprovação de que a reta, encontrada na matriz acumuladora, realmente existe na imagem original. Se o pixel estiver aceso na imagem original este mesmo pixel será aceso na imagem de saída. Também podemos verificar se este é o primeiro ponto da reta. Se este for o caso, então, gravam-se as suas coordenadas (**x**,**y**) como ponto inicial, se não for, gravam-se as suas coordenadas (**x**,**y**) como um ponto qualquer pertencente a reta.

Como não temos a informação do tamanho da reta, devemos percorrê-la a fim de encontrar o ponto final. O ponto corrente sempre será escolhido como o ponto final, mas se existir um próximo ponto, as coordenadas do ponto final são substituídas pelas coordenadas do último ponto percorrido e assim sucessivamente até não haver mais próximo ponto. Identificando assim os

pontos de início e fim de reta, que devem ser armazenados para vetorização das figuras.

Para retas com inclinação, existem casos em que a reta não fica totalmente perfeita sob a reta da imagem original, ou seja, há um pequeno desvio de inclinação. Desse modo, percorrendo a reta identificada, se o pixel corrente aceso não estiver também aceso na imagem original, ainda deve-se verificar na vizinhança se existe um pixel que pelo menos esteja aceso na imagem original, caso exista esse pixel, as suas coordenadas passam a fazer parte da matriz de pontos da reta identificada. Assim, caso ocorra um desvio em alguns pixels na imagem detectada, este algoritmo procura os pixels corretos. Isso é feito através de uma simples comparação entre imagem original e imagem detectada.

A vizinhança é considerada dependendo da tolerância de aproximação (**ap**) dada, quanto maior a tolerância maior será o tamanho da vizinhança. Para explicar melhor, vejamos o exemplo representado pela **figura 2.7** em que o pixel da imagem detectada não coincide com o pixel da imagem original.

Supondo que o pixel corrente na imagem detectada seja o mostrado com a cor cinza e que o pixel na imagem original que está mais próximo está em vermelho na **figura 2.7**. O pixel em cinza não pode ser gravado como um ponto pertencente à imagem detectada, pois não coincide com a imagem original. Dessa maneira o pixel em vermelho substituirá o pixel em cinza e será inserido na imagem detectada.

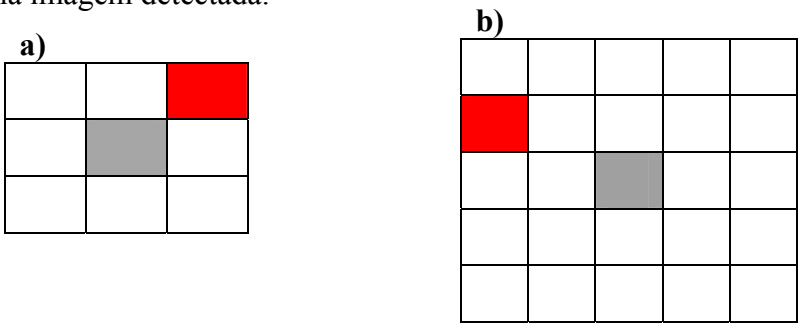

Figura 2.7 – Imagem detectada sob imagem original. **a)**Vizinhança com aproximação de 1 pixel; **b)**Vizinhança com aproximação de 2 pixels.

Identificados os pontos final e inicial de cada reta, eles são armazenados. Toda vez que for requerido, as retas já vetorizadas serão plotadas do ponto inicial ao ponto final.

2.4- Testes com retas

Para verificar a eficácia na vetorização de retas utilizando a transformada de Hough um programa foi desenvolvido na linguagem C com base no algoritmo já mostrado para detecção de retas. Foram realizados diversos testes usando o programa implementado e compilado no C++ Builder 5.0. Nesta seção, as imagens de teste com exceção do molde, foram geradas através do programa PaintBrush.

A discretização dos testes que seguem são de 4 graus para θ e 1 pixel para ρ. A seguir são mostradas algumas imagens e o resultado obtido para detecção das retas que as compõem.

O primeiro teste visa verificar se o programa identifica retas simples, mas com inclinações diferentes. Ele é representado pelas **figuras 2.8** e **2.9.** Foi utilizado 1 pixel de tolerância a erro, devido à existência de uma reta inclinada. Este teste mostra que apesar da imagem com aliasing, é possível identificar uma única reta e não segmentos separados. Os dados das três retas detectadas estão na **tabela 2.1.**

Tabela 2.1- Características extraídas de três retas (**teste 1**).

θ	ρ	**Ponto inicial**	**Ponto final**
-40^o	18	(26,0)	(161,158)
0^o	97	(98,2)	(98,210)
90^o	115	(0,116)	(161,116)

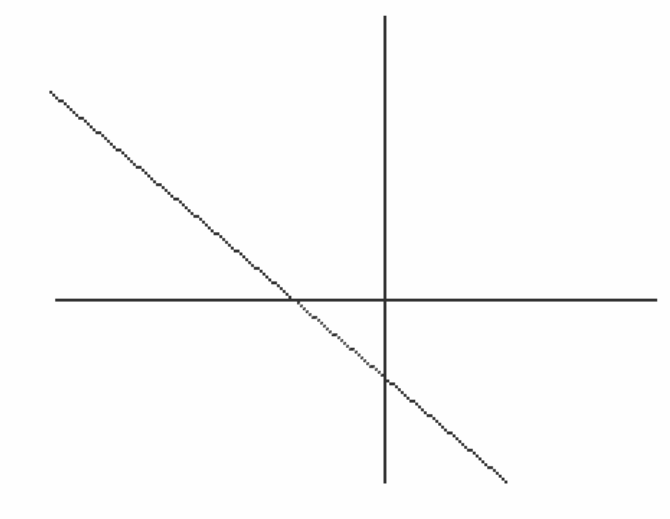

Figura 2.8- Teste1: Imagem de entrada com 3 retas.

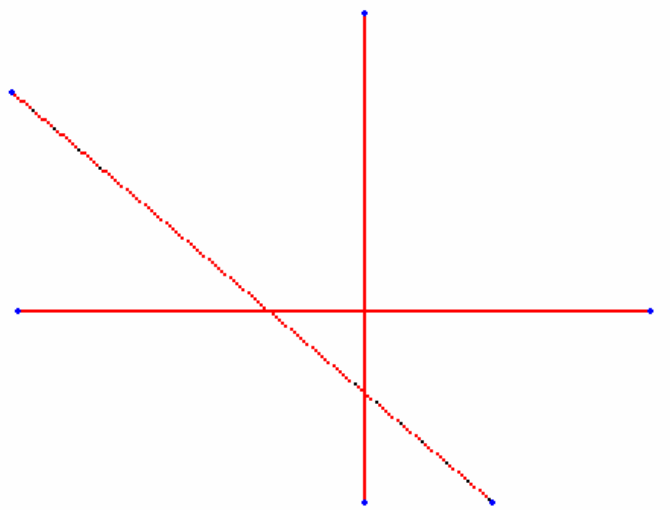

Figura 2.9- Teste 1: 3 retas. Imagem das 3 retas detectadas

O segundo teste (**figura 2.10, 2.11),** mostra que mesmo pequenos segmentos de retas podem ser detectados, com exceção para as imagens que apresentam um alto número de ruídos. A discretização utilizada foi de 0,5 pixel

para **ρ** e 4 graus para **θ** e não foi utilizado pixel de tolerância a erros. A **figura 2.11**, mostra que o programa também identifica pontos iniciais e finais de cada reta podendo assim fazer vetorização de retas. Os dados detectados do teste 2 estão presentes na **tabela 2.2**.

Tabela 2.2- Características extraídas de 4 retas (**teste 2**).

θ	ρ	**Pontos inicial**	**Ponto final**
0^o	62	(63,102)	(63,177)
0^o	83	(84,52)	(84,127)
0^o	93	(94,17)	(94,49)
0^o	97	(98,2)	(98,14)

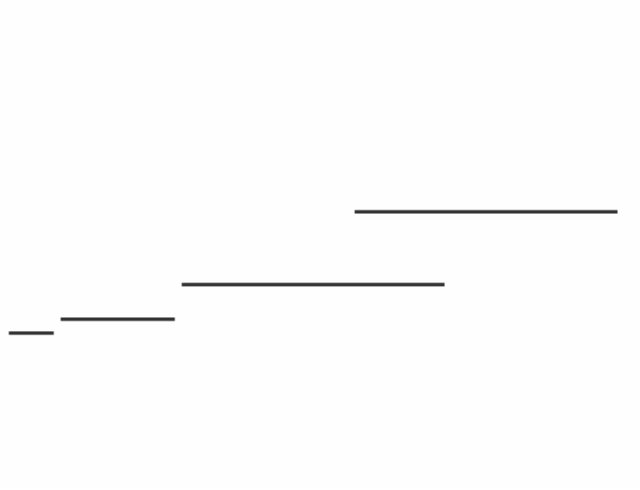

Figura 2.10- Teste 2:Imagem de entrada com 4 retas horizontais.

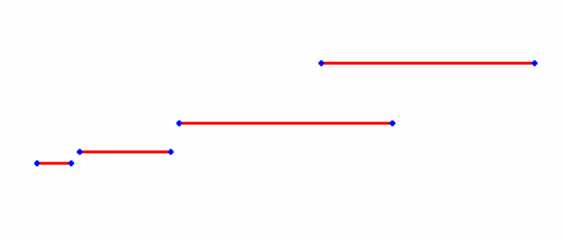

Figura 2.11- Teste 2: Resultado obtido das 4 retas horizontais.

O terceiro teste mostra individualmente o comportamento da detecção de uma reta inclinada a partir da **figura 2.12**, com resultado mostrado na **figura 2.13**. Foram utilizados 2 pixels de tolerância a erros, intervalo de **ρ**=1 pixel e pontos inicial e final de (1,0) e (160,208).

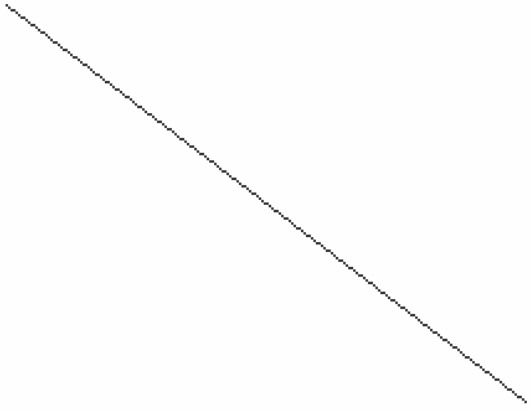

Figura 2.12- Teste 3: Imagem de entrada com uma reta a 154º ou -26º.

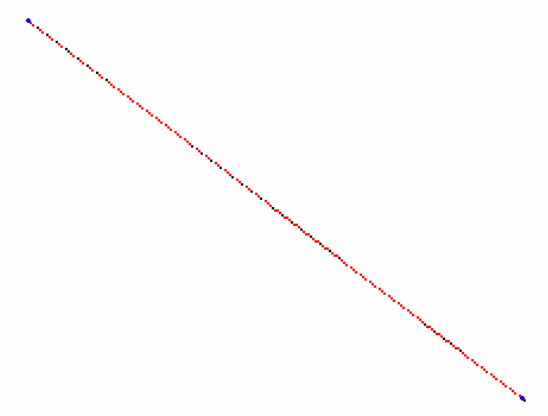

Figura 2.13- Teste 3: Resultado obtido de uma reta inclinada.

É importante ressaltar que este programa não detecta formas completas e sim, as várias retas pertencentes à forma geométrica que se deseja detectar. No teste 4, foram identificadas 4 retas. Não foi utilizada tolerância a erros, já que a **figura 2.15** trata-se somente de retas totalmente horizontais e verticais. A **tabela 2.3** resume melhor as características extraídas do teste 4.

Tabela 2.3- Características extraídas de um retângulo (**teste 4**).

θ	ρ	**Pontos inicial**	**Ponto final**
0	26	(26,40)	(26,154)
0	116	(116,40)	(116,154)
90	40	(26,40)	(116,40)
90	154	(26,154)	(116,154)

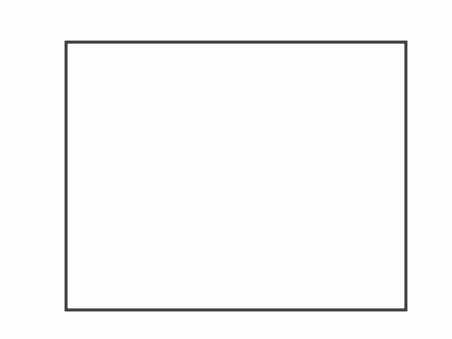

Figura 2.14- Teste 4: Imagem de entrada com um retângulo.

Figura 2.15- Teste 4: Resultado obtido de um retângulo.

O teste mostrado a seguir foi aplicado a uma imagem de molde de vestuário digitalizado em escanner.

No teste 5, representado pelas **figuras 2.16**, **2.17** e **2.18**, foram identificadas 5 retas sob o molde, a **figura 2.17** mostra as retas identificadas sem limites inicial e final, já a **figura 2.18** mostra a imagem com limites inicial e final de acordo com o molde original. O teste utiliza 1 pixel de tolerância a erros.

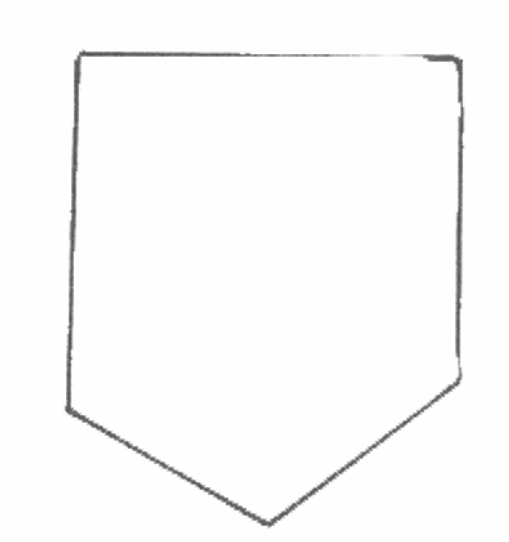

Figura 2.16- Teste 5: Imagem de entrada com um molde (corte de bolso).

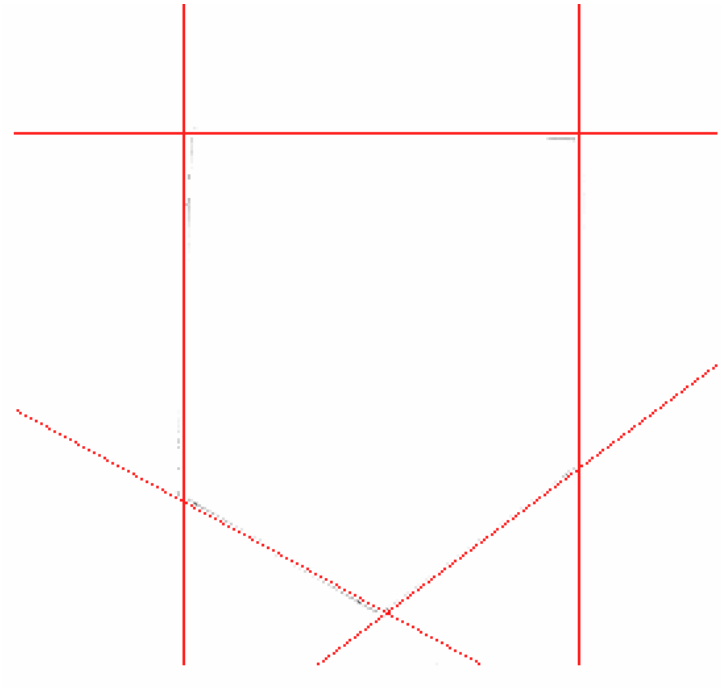

Figura 2.17- Teste 5: resultado obtido do molde sem limites.

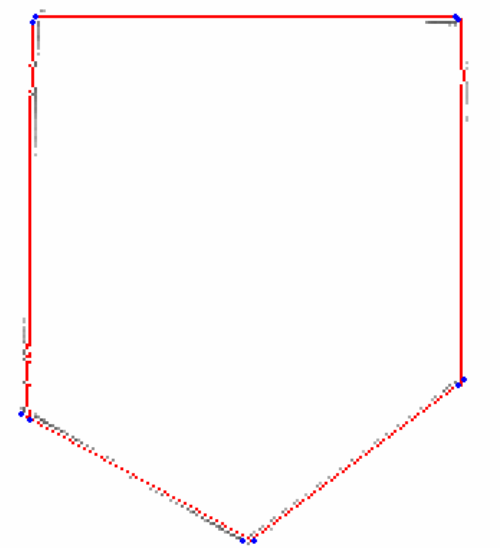

Figura 2.18- Teste 5: Resultado obtido do molde com limites.

Todos os testes durante a execução de detecção, geram um arquivo que possui todos os dados da imagem com intuito de reconstruí-las, a chamada vetorização. Mas mostraremos apenas o arquivo referente ao último teste apresentado. O arquivo apresentado pela **figura 2.19** mostra os dados gerados pela execução do teste 5. Cada linha de dados de uma reta encontrada significa respectivamente: o ângulo de inclinação **θ**, a distância perpendicular de reta à origem **ρ** e dois pares de pontos extremos.

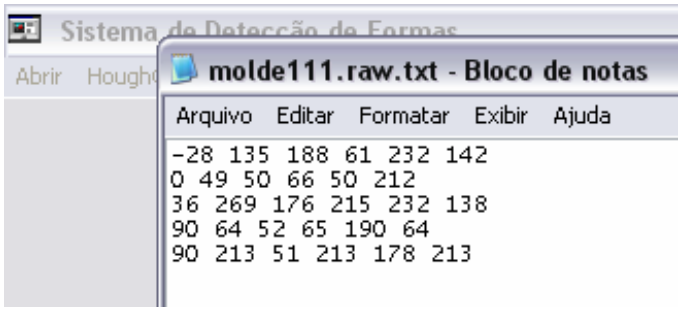

Figura 2.19- Arquivo gerado através da execução do teste 5.

O tempo de execução e o tamanho de cada uma das imagens desses testes podem ser observados na **tabela 2.4**.

Tabela 2.4- Dimensões das imagens e tempo de execução de cada teste.

Teste	Dimensão	Tempo de execução(s)
Teste 1	211x162	0,031
Teste 2	211x162	0,063
Teste 3	211x162	0,016
Teste 4	233x165	0,063
Teste 5	267x252	0,047

2.5- Conclusão

Os algoritmos desenvolvidos mostraram-se na maioria das vezes muito eficientes na detecção de retas, como mostrado nos testes.

Na existência de mais de duas retas possuindo ρ e θ iguais, e tendo diferença apenas no deslocamento, é identificado apenas como se fosse uma única reta. A identificação de retas inclinadas com tangentes próximas a ± 1 é a que apresenta maiores discrepâncias. A detecção dos pontos iniciais e finais que possibilitam uma vetorização simples dos moldes compostos por segmentos de retas no entanto pode identificar uma única reta ao invés de vários segmentos de retas com valores ρ e θ muito próximos, tanto no caso em que houver falha no contorno quanto para retas com tangentes em torno de ± 1.

Capítulo 3- Detecção de formas cônicas

Retas não são as únicas formadoras dos moldes. A etapa seguinte seria a detecção das curvas que as compõem.

As cônicas (**figura 3.1**) são curvas obtidas pela interseção de um plano com um cone circular. Se o plano for perpendicular ao eixo, sem passar pelo vértice, obtemos como resultado do corte uma **circunferência**. Se o plano de corte for paralelo a uma geratriz do cone, sem passar pelo vértice, obtemos uma **parábola**. Se o plano de corte for paralelo ao eixo e não passar pelo vértice, obtemos uma **hipérbole**. Se o plano não for paralelo ao eixo, nem a nenhuma geratriz e não passar pelo vértice, obtemos uma **elipse**.

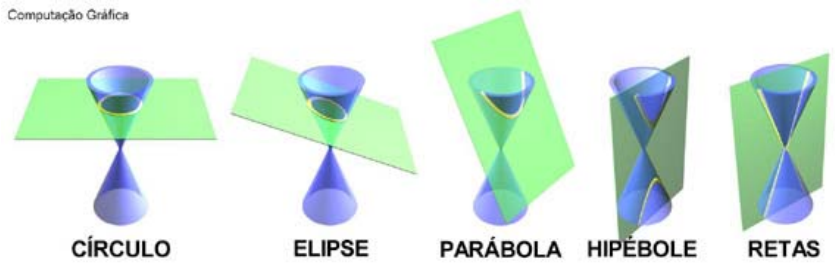

Figura 3.1 – Cônicas.[AZE2003]

Com o intuito de identificar partes curvas dos moldes de vestuário, começamos pela cônica mais simples, depois das retas, círculos. Então, como fazer a aplicação da transformada de Hough também para círculos é o primeiro assunto considerado neste capítulo

Progressivamente iremos aplicar a transformada de Hough em formas curvas mais complexas a fim de encontrar a forma geométrica mais compatível com o tipo de curva presente nos moldes.

3.1- Transformada de Hough para formas circulares

A circunferência é o lugar geométrico dos pontos do plano eqüidistantes de um ponto fixo, chamado centro, a distância entre o centro e um ponto qualquer da circunferência é o raio.

Na detecção de círculos [KIM1975], usando a transformada de Hough, podemos usar a fórmula implícita:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

onde $\mathbf{x_0}$ e $\mathbf{y_0}$ são as coordenadas cartesianas do centro do círculo e **r** é o seu raio.

Sendo que agora ao invés de usar uma matriz acumuladora bidimensional, como no caso de retas, devemos usar uma matriz acumuladora tridimensional contendo os parâmetros: $\mathbf{x_0}$, $\mathbf{y_0}$ e **r**. A eficiência computacional neste caso piora em relação ao algoritmo para detecção de retas, devido ao maior número de parâmetros.

Com a fórmula implícita $(x - x_0)^2 + (y - y_0)^2 = r^2$, é difícil evidenciar as coordenadas do centro $\mathbf{x_0}$ e $\mathbf{y_0}$ como função das demais variáveis, sendo mais adequado utilizar a forma polar.

$$x_0 = x - \rho\cos\theta$$

$$y_0 = y - \rho\sin\theta$$

Nesta forma para obter os valores **x** e **y** já conhecendo as coordenadas do centro do círculo, basta explicitar **x** e **y**.

$$x = x_0 + \rho\cos\theta$$

$$y = y_0 + \rho\sin\theta\,\mathbf{,}$$

onde ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) são coordenadas polares de um círculo.

Para utilizar a transformada de Hough com matriz acumuladora de 3 dimensões, devemos previamente encontrar o valor real do seu raio de círculo ou pelo menos ter uma idéia de valor a ser dado ao intervalo de raio.

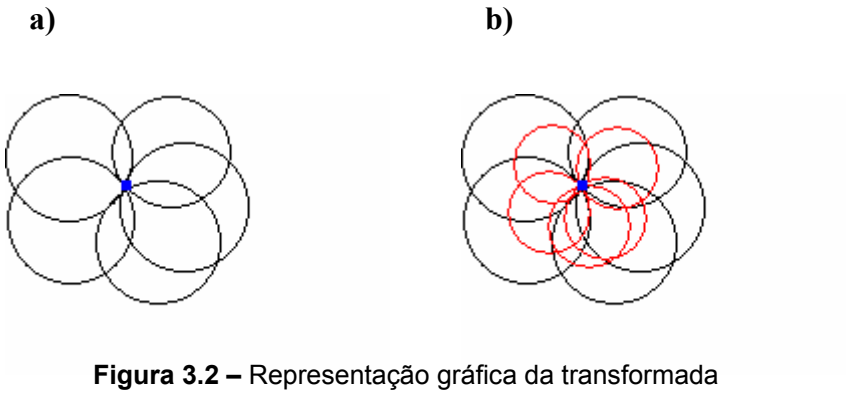

Figura 3.2 – Representação gráfica da transformada de Hough para círculos.

Para cada ponto da imagem é considerado um intervalo de raios e para cada raio são calculadas as coordenadas cartesianas do centro do círculo,como mostrado na **figura 3.2**.

Na **figura 3.2**, o ponto em azul representa um ponto da imagem e para cada ponto são simulados todos os círculos que possam passar por ele com um determinado raio. Em **a)** considerando apenas um tamanho de raio e em **b)** simulando dois tamanhos diferentes de raio.

Cada centro encontrado associado a um raio, é incrementado de uma unidade na matriz acumuladora. O maior valor na matriz acumuladora é encontrado através do **algoritmo 2.2- buscar_maiores**(), já apresentada na detecção de retas. E assim extrai-se seus respectivos parâmetros, $\mathbf{x_0}$, $\mathbf{y_0}$ e $\boldsymbol{\rho}$.

3.1.1-Discretização da matriz acumuladora para círculos

Como no caso de retas, para curvas também é necessário estabelecer um intervalo limite para todos os três parâmetros. A coordenada **a** deve ser definida em um intervalo limite de **0** à **Ni** (coordenada máxima da figura no eixo **x)**. A coordenada **b** deve ser definida em um intervalo limite de **0** à **Nj** (coordenada máxima da figura no eixo **y)**. O raio possui tamanho conforme estabelecido pelo usuário do programa de detecção de círculos. Poderá ser

fornecido apenas um raio ou um intervalo de raios, onde será detectado o círculo com mais votos dentro desse intervalo.

3.1.2-Algoritmo de Hough para círculos

A seguir é apresentado o **algoritmo 3.1** que usa "força bruta" para detecção de círculos.

De acordo com a necessidade de detecção de um determinado tipo de círculo, neste algoritmo pode ser utilizado um único raio ou um intervalo de raios. Para cada tamanho de raio, são calculados possíveis coordenadas de centro, para ângulos no intervalo $(0^o, 360^o]$ ou $(0, 2\pi]$

Da matriz acumuladora obtém-se as coordenadas do centro e seu respectivo raio. Incrementando de uma unidade a posição correspondente cada conjunto de parâmetros encontrado.

I(x,y)-valor do pixel **(x,y)**da imagem em tom de cinza;

M(x_0,y_0, ρ)-matriz de parâmetros;

minimo,**maximo**- limites do intervalo de tamanho de raio, esse dado é inserido pelo usuário.

```
circulo_hough()
    Guardar todos os pixels num array de uma dimensão
    Limpar acumulador M(x0,y0,ρ)
    Para pixel I(x,y), se este pixel I(x,y) estiver aceso na
    imagem
        Para ρ>minimo até ρ<maximo
            Para θ=0 até θ<2π
                x0 = x − ρ cos θ
                y0 = y − ρ sin θ
                M(x0,y0, ρ)=M(x0,y0, ρ)+1
            Faça para θde 0 à 2π
        Faça para todos os raios de mínimo à máximo
    Faça para todos os pixels da imagem
```

Algoritmo 3.1- Detecção dos parâmetros do círculo

Assim como na detecção de retas, antes de prosseguir na obtenção dos parâmetros do círculo é necessário ordenar os maiores votos usando o algoritmo buscar_maiores (**algoritmo 2.2**), anteriormente mencionado.

3.1.3 -Definição de círculo pelos parâmetros encontrados

Assim como na detecção de retas, deve-se voltar ao espaço da imagem identificando os parâmetros encontrados para aí vetorizá-los.

No algoritmo que segue é considerado que:

IS(x,y)-imagem de saída
x_menor e **y_menor**- coordenadas **x** e **y** respectivamente do ponto inicial do círculo ou arco de círculo
x_maior e **y_maior**- coordenadas **x** e **y** respectivamente do ponto final do círculo ou arco de círculo
ap-aproximação, taxa de erro para encontrar pixel aceso
busca_aceso- função que recebe **true** se através da aproximação **ap** encontrar pixel aceso na imagem original, recebe **false** se não encontrar pixel aceso na vizinhança **ap**
w-contador
linha-número de círculos a serem identificados. Total de linhas que a matriz armazenadora de pontos iniciais e finais pode ter.
circulo_coord- matriz armazenadora de pontos iniciais e finais
dentro- será **false** se o pixel em questão não estiver no arco de círculo, será **true** se o pixel estiver no arco de círculo.

circulo_inversa()
Inicializar **x_menor**, **y_menor**,**x_maior**,**y_maior**,**anterior_x**, **anterior_y**
Para cada célula na matriz acumuladora **M(x_0,y_0, ρ)**
Para **θ =0** até **θ = 2π**

$$x = x_0 + \rho \cos\theta$$

$$y = y_0 + \rho \sin\theta$$

Se (**x>0**) e (**x<Ni**) e (**y>0**)e (**y<Nj**), ou seja, se estiver no domínio da imagem
Se o pixel **I(x,y)** estiver aceso na imagem original então
Se a variável **dentro** for igual a **false**, significa que o pixel não está sobre o círculo então
x_menor = **x**, **y_menor** = **y**
dentro=true
O **pixel IS(x,y)** na imagem de saída é aceso
Senão, se o pixel **I(x,y)** não estiver aceso na imagem original então
aceso=busca_aceso()
Se **aceso** for igual a **true então**
Se a variável **dentro** for igual a **false** então
x_menor = **x**, **y_menor** = y
dentro=true
O **pixel IS(x,y)** na imagem de saída é aceso
Senão, se **aceso** for igual a **false** e a variável dentro for igual a **true** então
x_maior = **anterior_x**, **y_maior** = **anterior_y**
dentro=false
anterior_x=x, **anterior_y=y**
Faça o laço para todos **θ** de **0** a **2π**
Para cada círculo identificado
Armazenar os pontos de um arco de círculo **(x_menor,y_menor)** e **(x_maior,y_maior)**
Faça o laço para todos os círculos identificados
Faça o laço para todos as células da matriz acumuladora.

Algoritmo 3.2- Definição dos elementos do círculo no espaço da imagem

O **algoritmo 3.2** circulo_inversa obtém as coordenadas do centro do círculo e seu respectivo raio. Obtendo as coordenadas dos pontos de contorno do círculo, através da equação do círculo na forma polar, passa-se a verificar se o pixel encontrado existe na imagem original, caso exista, o pixel com tais coordenadas passa a fazer parte da imagem de saída, caso não exista, é feita uma procura na vizinhança, por um pixel aceso, se na vizinhança existir, este é adotado como pixel aceso na imagem de saída, se mesmo assim nenhum pixel na vizinhança estiver aceso, o loop continua com outro ângulo.

Para encontrar pontos iniciais e finais de arcos de círculo, foi adotado o seguinte método: gerenciar o início e término de arcos de círculo (**figura 3.3**) é utilizada a variável **dentro** que recebe **true** quando está percorrendo o arco e o primeiro pixel no sentido horário a partir do ângulo zero, é marcado como ponto inicial. Quando o arco termina, dentro recebe **false** e o pixel anterior é marcado como ponto final.

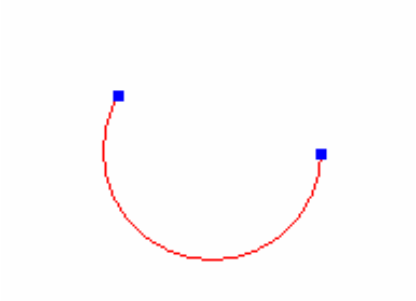

Figura 3.3 - Arco de círculo.

3.1.4-Testes com círculos e arcos de círculos

Foi implementado um programa para detecção de arcos de círculos utilizando os **algoritmos 3.1** e **3.2**. O programa funciona também para imagens com falhas. Para testar os algoritmos de círculos e arcos de círculo foram realizados **7** testes. As imagens de resultado são mostradas na cor vermelha.

Para todas as formas circulares foi aplicada uma discretização de intervalo de 1 pixel para $\mathbf{x_0}$ e para $\mathbf{y_0}$. As imagens apresentadas nos testes a seguir exceto no sétimo, foram desenhadas no programa PaintBrush e o valor do raio ou de um intervalo de raios,é inserido pelo usuário

A **figura 3.4** mostra a imagem de entrada do primeiro teste e a **figura 3.5** o seu resultado obtido. Neste teste que ilustra a identificação de um círculo, foi usado raio=69, sem tolerância a erros. O centro do círculo identificado, é mostrado na **figura 3.5** pelo ponto de coordenadas (95,102).

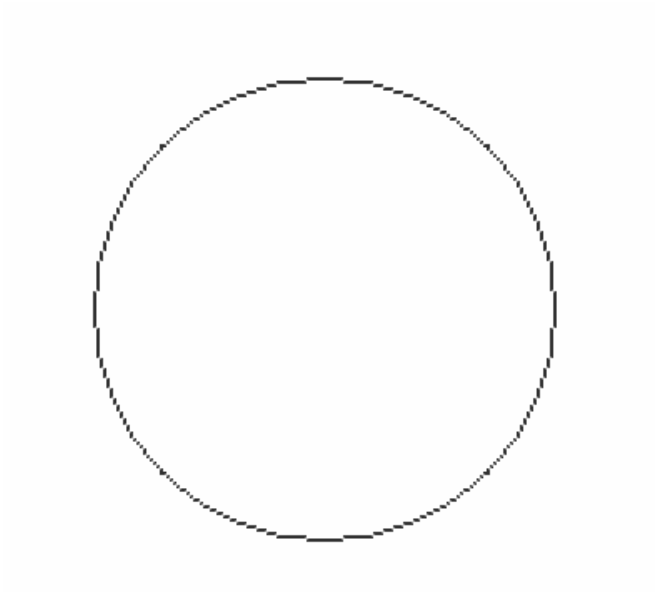

Figura 3.4- Teste 1:Imagem de entrada com único círculo completo.

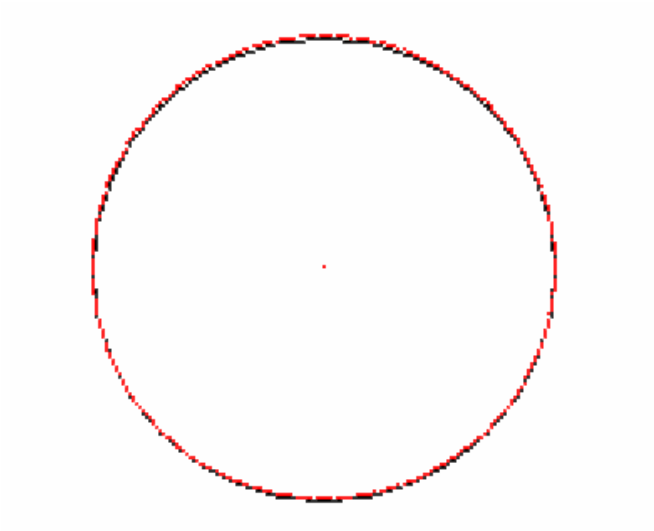

Figura 3.5-Teste 1:Imagem de resultado do círculo detectado.

O segundo teste (**figuras 3.6** e **3.7**) realizado para círculos, é muito semelhante à imagem do primeiro teste, mas o segundo apresenta falhas no contorno. É importante diferenciar falhas e arcos de círculo, pois o programa desenvolvido no caso de um círculo com várias falhas, não irá identificar pontos extremos de cada segmento, apenas poderá ser utilizado para identificar o círculo por completo. Já no caso de arco de círculo, poderão ser identificados seus pontos extremos. Este teste utilizou raio=69, sua tolerância a erros foi nula.e as coordenadas de centro do círculo são (95,102).

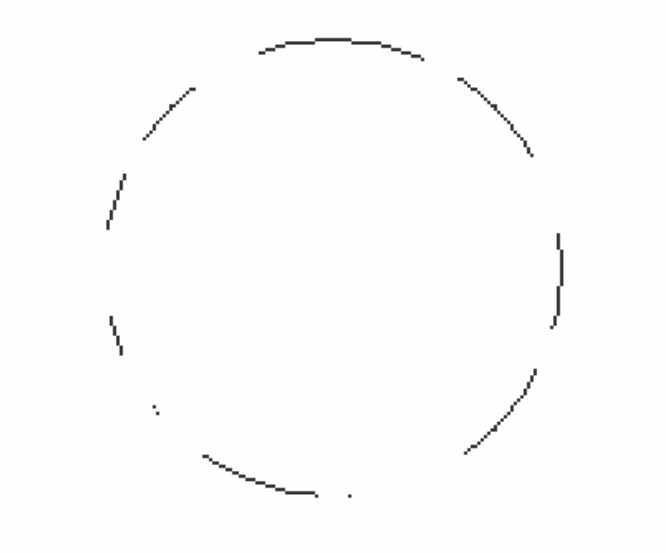

Figura 3.6- Teste 2: círculo com falhas.

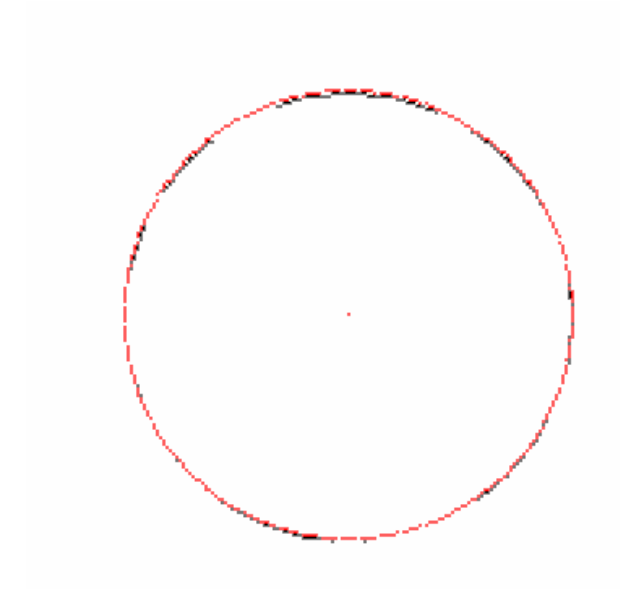

Figura 3.7- Teste 2: resultado do círculo com falhas.

O terceiro teste considera a detecção de 3 círculos (**figura 3.8**), mostrando a capacidade de identificar mais de um círculo por execução. Para que os três círculos pudessem ser identificados (**figura 3.9)**, foi utilizado um intervalo de 11 a 27 pixels para raio , sem tolerância a erros. As informações extraídas encontram-se na **tabela 3.1**.

Tabela 3.1- Características extraídas de 3 círculos (**teste 3**).

ρ	Centro	Ponto inicial	Ponto final
11	(24,144)	(0,0)	(0,0)
23	(39,37)	(0,0)	(0,0)
27	(107,129)	(0,0)	(0,0)

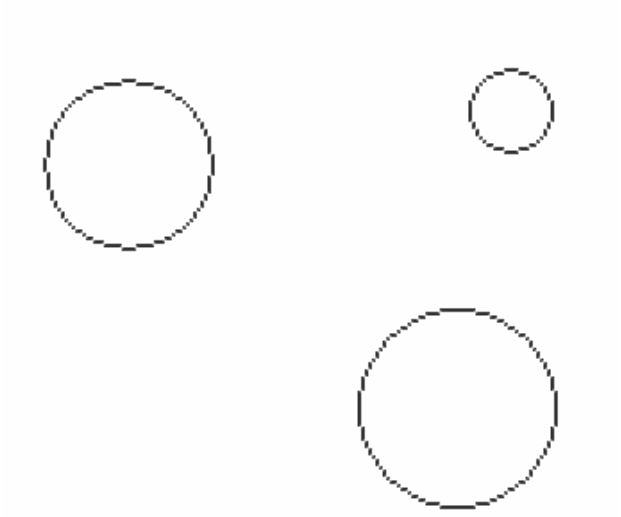

Figura 3.8- Teste 3: imagem de entrada com 3 círculos.

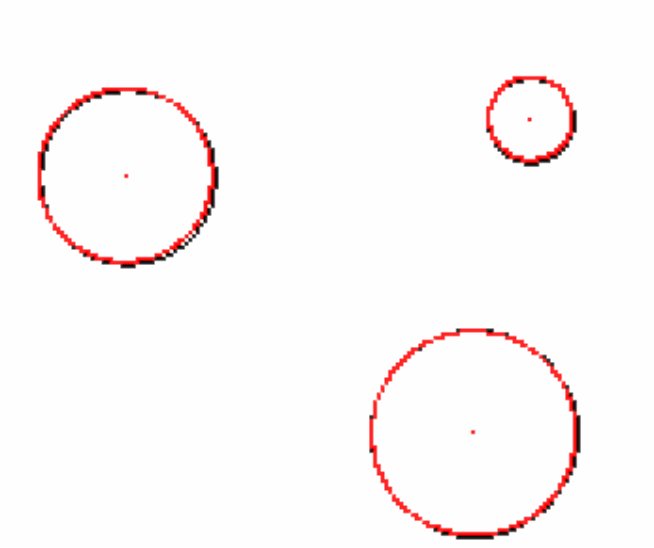

Figura 3.9- Teste 3: Resultado da detecção de 3 círculos.

O quarto teste ilustra a identificação de arcos de círculo verticais e horizontais. As **figuras 3.10** e **3.11** não consistem em apenas um arquivo e sim dois arquivos diferentes. O resultado obtido pode ser visto na **figura 3.11**, onde os pontos inicial e final são identificados pela cor azul. Foi usada tolerância a erros igual a 1 pixel. Os dados detectados de cada arco foram: arco horizontal-centro (63,59) e pontos extremos: (64,7) e (65,111). Arco vertical-centro (62,66) e pontos extremos (116,67) e (8,67).

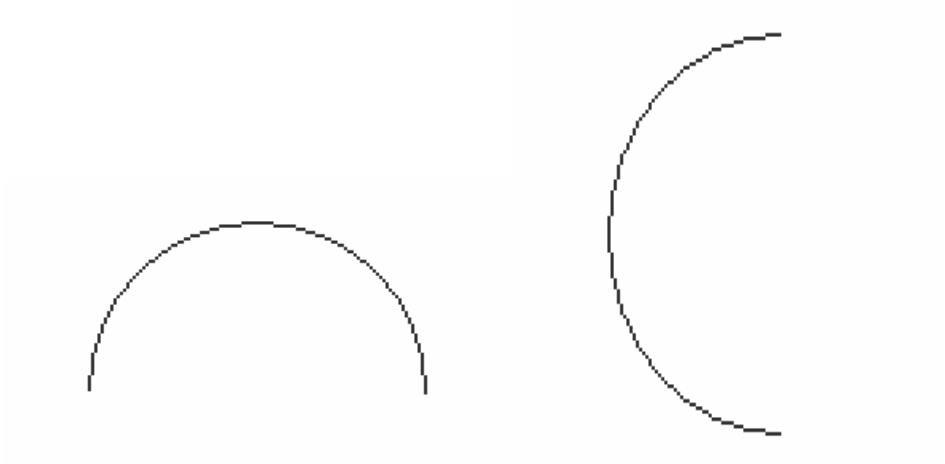

Figura 3.10- Teste 4: Duas Imagens de entrada com arcos de círculos.

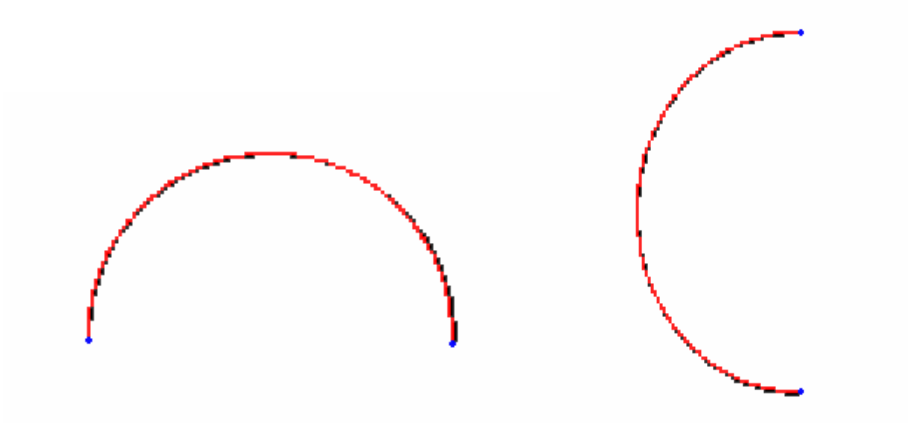

Figura 3.11- Teste 4: Resultado da detecção de arcos de círculos.

No quinto teste é ilustrada a detecção de um arco de círculo com ângulo menor que 180 graus. Foi utilizado raio=68 e tolerância de 1 pixel para erros. O centro de arco encontrado tem coordenadas (95,101) e seus pontos extremos são: (117,37) e (33,129).

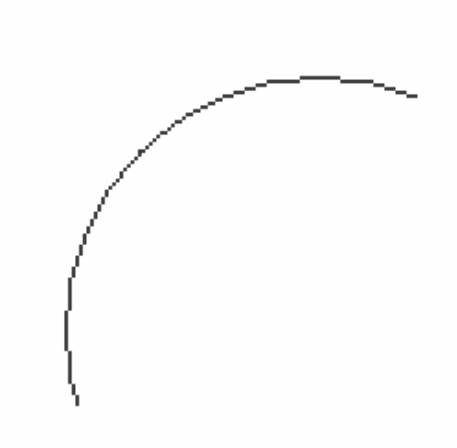

Figura 3.12- Teste 5: Imagem de entrada com arco de círculo de ângulo qualquer.

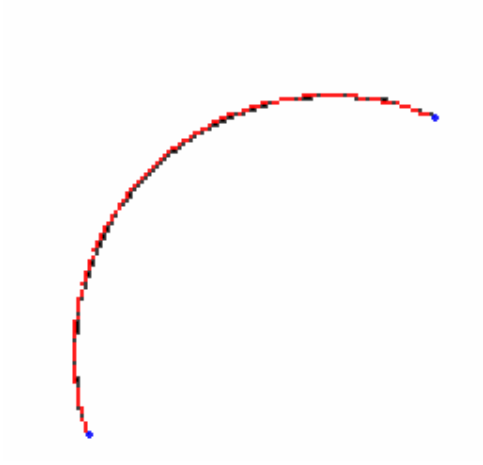

Figura 3.13- Teste 5: Imagem do resultado da detecção de um arco de círculo de ângulo menor que 180 graus.

O sexto teste mostra como o programa desenvolvido se aplica a arcos de círculo maiores que 180 graus. Para este teste foram utilizados também raio=69 e tolerância a erros de 1 pixel. As coordenadas do centro do arco de círculo são (95,102) e seus pontos extremos são: (81,169) e (26,107).

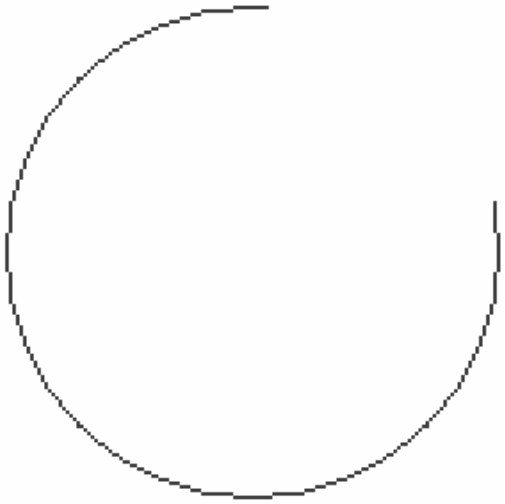

Figura 3.14- Teste 6: Imagem de entrada com arco de círculo de ângulo maior que 180 graus.

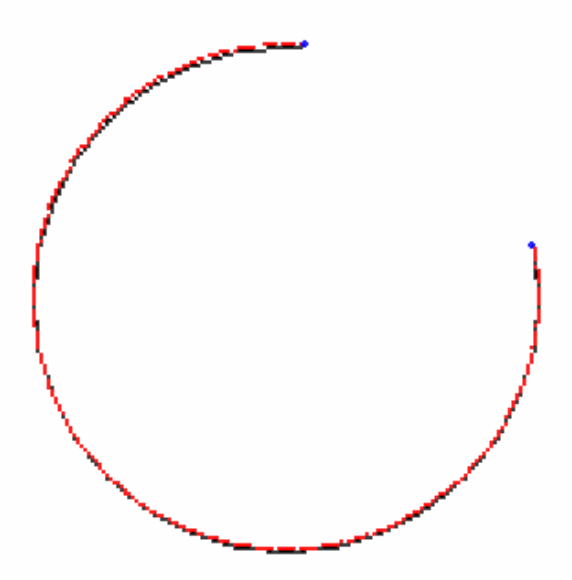

Figura 3.15- Teste 6: Resultado da detecção de um arco de círculo de ângulo maior que 180 graus.

O teste 7 trata-se da aplicação do algoritmo de detecção de círculos em um molde, neste caso foram detectados somente os círculos para posteriormente no capítulo 4 ser mostrada a detecção completa desse molde.

Esta imagem foi digitalizada em escanner e este molde trata-se de um corte de saia.

A detecção de cada círculo foi feita separadamente, pois a utilização do intervalo entre os dois raios causaria grande aumento no tempo de execução. As informações extraídas do molde estão apresentadas na **tabela 3.2**:

Tabela 3.2- Características extraídas de 2 arcos de círculo (**teste 7**).

ρ	Centro	Ponto inicial	Ponto final
51	(109,46)	(5,140)	(216,136)
140	(109,32)	(74,70)	(144,69)

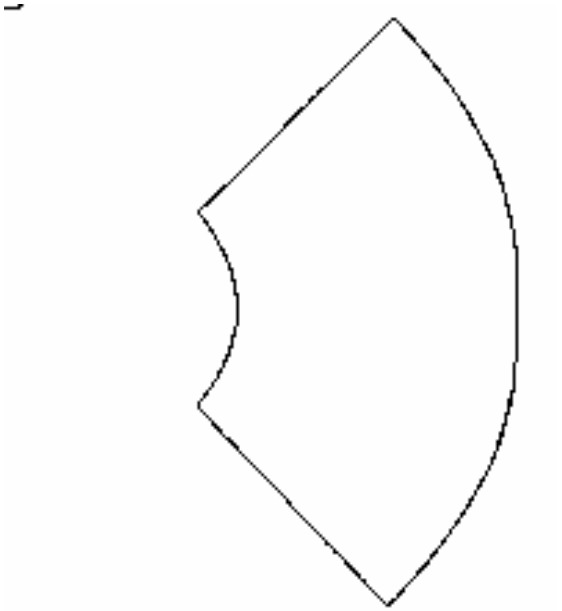

Figura 3.16- Teste 7: Imagem de entrada de um molde (saia).

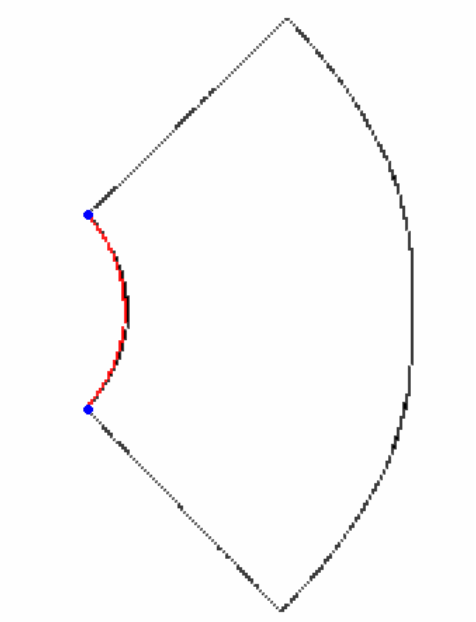

Figura 3.17- Teste 7: Resultado da detecção de um arco de círculo de raio 51 pixels.

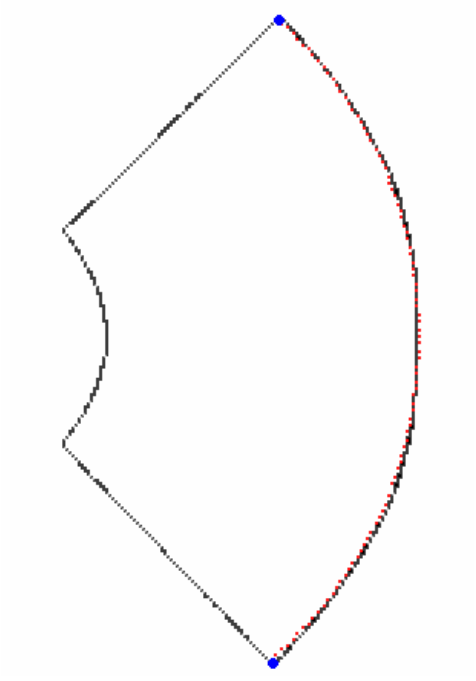

Figura 3.18- Teste 7: Resultado da detecção de um arco de círculo de raio 140 pixels.

Os dados para vetorização desses arcos a partir do molde, podem ser encontrados num arquivo produzido pelo sistema . Esse arquivo pode ser visualizado na **figura 3.19**.

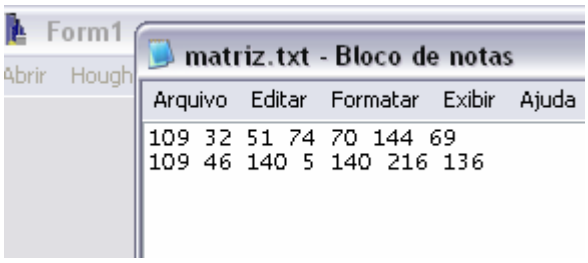

Figura 3.19- Dados da vetorização do teste 7. Respectivamente: coordenadas **x** e **y** do centro, raio, e pares de pontos inicias e finais dos arcos.

A partir desses dados armazenados é possível reconstruir os arcos de círculo identificados anteriormente.

Todas as imagens testadas são do tipo . raw com dimensão e tempo de execução conforme a **tabela 3.3**.

Tabela 3.3- Dimensões das imagens e tempo de execução de cada teste.

Teste	Dimensão	Tempo de execução(s)
Teste 1	249x175	0,094
Teste 2	249x175	0,062
Teste 3	192x140	1,157
Teste 4	192x140	0,047
	192x140	0,047
Teste 5	249x175	0,047
Teste 6	249x175	0,094
Teste 7	200x220	0,10+0,11=0,21

3.1.5-Conclusão

Através dos testes efetuados concluímos que a transformada de Hough funciona perfeitamente para círculos, arcos de círculos e círculos com falhas. Podendo ser útil mais tarde no processo de detecção de moldes completos. O sistema porém não foi projetado para identificar vários segmentos ao longo de uma forma circular com falhas, o resultado relevante será apenas o conjunto centro e raio de um círculo completo.

3.2- Transformada de Hough para elipses

Elipse (**figura 3.20**) é uma curva plana, definida como o lugar geométrico dos pontos do plano para os quais a soma das distâncias a dois pontos fixos desse plano $\mathbf{F_1}$ e $\mathbf{F_2}$ é uma constante.

O eixo $\mathbf{S_1S_2}$ é denominado **eixo maior** da elipse e seu raio maior **s** é igual a metade do eixo maior. O eixo $\mathbf{T_1T_2}$ é denominado **eixo menor** da elipse e seu raio menor **t** é igual a metade do eixo menor. A distância **c** é igual a distância do centro aos focos ($\mathbf{F_1}$ ou $\mathbf{F_2}$).

Os pontos $\mathbf{F_1}$ e $\mathbf{F_2}$ são denominados focos e a distância $\mathbf{F_1F_2}$ é conhecida como distância focal da elipse. O quociente **c/s** é conhecido como excentricidade da elipse. Como, por definição, **s > c**, podemos afirmar que a excentricidade de uma elipse é um número positivo menor que a unidade.

Seja **P(x, y)** um ponto qualquer de uma elipse e sejam $\mathbf{F_1}(c,0)$ e $\mathbf{F_2}(-c,0)$ os seus focos. Sendo **2s** o valor constante com **c < s**, como vimos acima, podemos escrever:

$$PF_1 + PF_2 = 2s$$

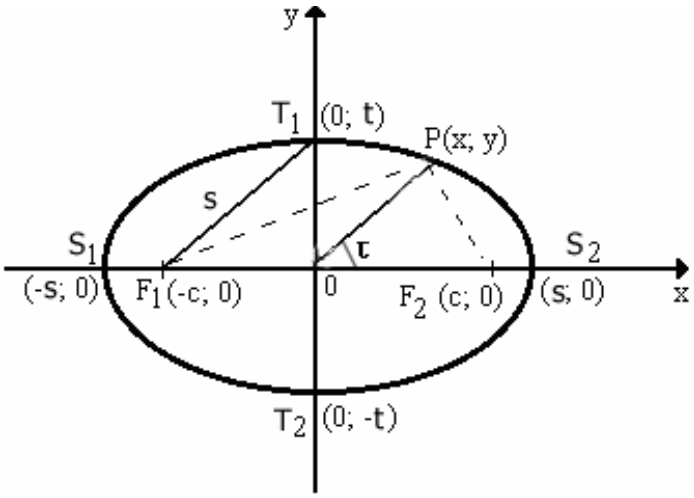

Figura 3.20- Elipse

Uma elipse é expressa da seguinte forma:

$$\frac{x^2}{s^2} + \frac{y^2}{t^2} = 1,$$

para o caso do eixo maior estar no eixo dos x e

$$\frac{x^2}{t^2} + \frac{y^2}{s^2} = 1,$$

para o caso do eixo maior estar no eixo dos y .

Para a equação polar da elipse (**figura 3.21**) tem-se:

$$\rho^2 = \frac{s^2 t^2}{s^2 \sin^2 \tau + t^2 \cos^2 \tau},$$

onde **s** é a metade do eixo maior, **t** é a metade do eixo menor, **ρ** a distância entre o centro e um determinado ponto na borda da elipse, e **τ** representa o ângulo que **ρ** faz com o eixo horizontal, sendo **(ρ,τ)** coordenadas polares da elipse.

Para desenhar uma elipse com sua inclinação em relação ao eixo horizontal usa-se uma matriz de rotação (**tabela 3.4**).

Tabela 3.4- Matriz de rotação.

$$\begin{vmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{vmatrix}$$

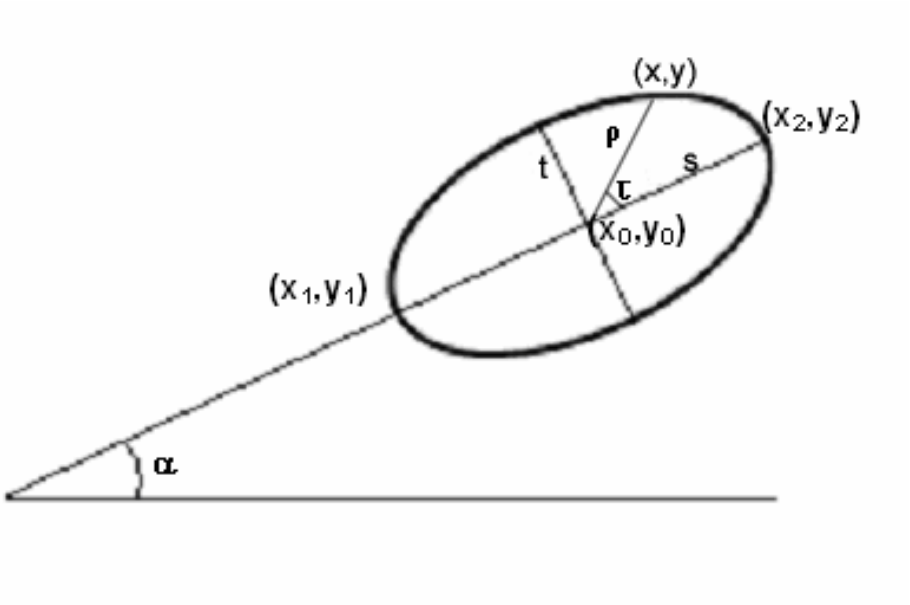

Figura 3.21- Elipse inclinada com seus cinco parâmetros e coordenadas inicial e final de seu maior eixo.

Cada coordenada **(x,y)** da elipse é transformada por essa matriz como mostra as equações abaixo.

$$x' = x\cos\alpha + y\sin\alpha$$

$$y' = y\cos\alpha - x\sin\alpha\,.$$

Um ponto da elipse pode ser determinado através de coordenadas polares da seguinte forma:

$$x = \rho\cos\tau$$

$$y = \rho\sin\tau\,,$$

como numa elipse: $\mathbf{[t \le \rho \le s]}$, então:

$$x = s\cos\tau$$

$$y = t\sin\tau\,,$$

operando a rotação de um ponto juntamente com uma translação $\mathbf{(x_0,y_0)}$ temos:

$$x' = x_0 + s\cos\tau\cos\alpha + t\sin\tau\sin\alpha$$

$$y' = y_0 + t\sin\tau\cos\alpha - s\cos\tau\sin\alpha$$

Para que uma elipse seja detectada a partir de um espaço de parâmetros, é necessário que todos os parâmetros mostrados na **figura 3.21** sejam calculados ($\mathbf{x_1,y_1}$) e ($\mathbf{x_2,y_2}$) representam as coordenadas que definem o limite do eixo maior da elipse, ($\mathbf{x_0,y_0}$) o centro deste eixo, **s** a metade do eixo maior , **t** a metade do eixo menor e α sua inclinação ao eixo horizontal.

Esses parâmetros podem ser calculados da seguinte forma:

$$x_0 = \frac{x_1 + x_2}{2}$$

$$y_0 = \frac{y_1 + y_2}{2}$$

$$s = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{2}$$

$$\alpha = \arctan\left[\frac{(y_2 - y_1)}{(x_2 - x_1)}\right]$$

Já para o cálculo do quinto parâmetro, **t** (metade do eixo menor), é necessário utilizar a equação polar da elipse, usando o termo **t** em evidência:

$$t^2 = \frac{s^2 \rho^2 \sin^2 \tau}{s^2 - \rho^2 \cos^2 \tau}$$

Para melhor entendimento, os termos desta fórmula estão representados na **figura 3.22.**

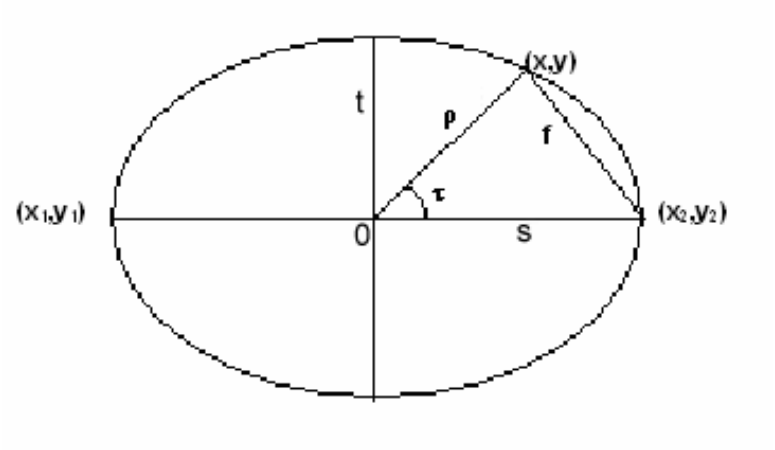

Figura 3.22- Triângulo não retângulo inscrito numa elipse.

Na **figura 3.22**, **f**, que ainda era um argumento desconhecido, representa a distância entre um ponto qualquer do contorno da elipse (**x,y**) ao ponto limite do eixo maior ($\mathbf{x_2,y_2}$).

O parâmetro **s**, já foi calculado anteriormente, **ρ** pode ser calculado a partir da seguinte expressão:

$$\rho = \sqrt{(x - x_0)^2 + (y - y_0)^2} ,$$

faltando apenas seno e cosseno do ângulo **τ**. Como o triângulo mostrado na **figura 3.22** é um triângulo não-retângulo, o **cosτ** pode ser calculado usando a **lei do cosseno:** $f^2 = \rho^2 + s^2 - 2\rho s \cos\tau$, colocando **cos τ** em evidência temos:

$$\cos\tau = \frac{s^2 + \rho^2 - f^2}{2\rho^2 s}$$

Utilizando a propriedade trigonométrica:

$$\cos^2\tau + \sin^2\tau = 1,$$

obtemos finalmente $\sin^2\tau$**:**

$$\sin^2\tau = 1 - \cos^2\tau$$

Com os valores, **ρ**,**s**, $\sin^2\tau$ e $\cos^2\tau$**,** obtidos**,** o parâmetro **t**, poderá ser calculado.

Já tendo calculado o último parâmetro **t**, é possível pensar sobre o mecanismo de armazenamento desses parâmetros e como a transformada de Hough será utilizada para elipses.

Primeiramente estabelece-se um valor para o eixo maior. Diante disso, são procurados na elipse dois pontos de contorno (x_1,y_1) e (x_2,y_2) que tenham a mesma distância do eixo maior, esta procura está representada na **figura 3.23**. Para cada dois pontos encontrados, sendo esses, possíveis pontos extremos do eixo maior, são calculados o centro da elipse (x_0,y_0), a metade do eixo maior (**s**) e sua inclinação (α). Mas para calcular o eixo menor, será necessário fornecer um outro ponto qualquer (**x,y**). Em torno da elipse entre $\left(0^o,360^o\right]$ ou $\left(0,2\pi\right]$, são escolhidos vários pontos e conseqüentemente calculados vários valores de **t**, que são incrementados em uma matriz acumuladora unidimensional. Assim a cada par de candidatos a pontos extremos do eixo maior, é eleito o parâmetro **t** com maior voto, que em seguida é posto em uma matriz armazenadora de 6 dimensões chamada de **elipse[ ]**, onde ficarão também seus 4 parâmetros restantes e seu número de votos obtidos, onde só poderá armazenar no máximo 20 elipses mais votadas.

Após a varredura da imagem a matriz **elipse[ ]**, conterá todos os parâmetros das elipses mais votadas entre todos os possíveis "eixos maiores" encontrados na imagem. Assim, novamente o maior valor é calculado e a elipse mais votada é encontrada.

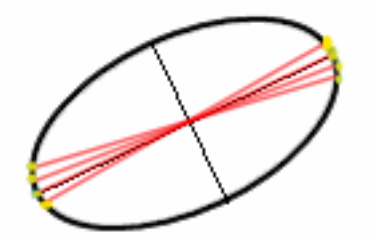

Figura 3.23- Cálculo de todos dos possíveis pontos extremos do eixo maior.

Apesar da grande dimensão da matriz armazenadora, sua restrição ao número de registros não prejudica portanto o desempenho do algoritmo. O algoritmo apresentado é baseado em [XIE2002]. A diferença é que, nesse

artigo, a cada eixo encontrado apaga-se a possível elipse encontrada, com o intuito de encontrar mais de uma elipse por imagem. Enquanto que o algoritmo apresentado no presente trabalho não procura detectar mais de uma elipse por imagem, obtendo-se uma maior precisão de valores na elipse detectada.

3.2.1-Discretização da matriz acumuladora para elipses

O eixo maior **2s** pode estar compreendido entre os limites de **2** e $\sqrt{N_i^2 + N_j^2}$, considerando **Ni x Nj** a dimensão da imagem. O eixo menor **2t** deve estar compreendido entre os limites **1** e $\sqrt{N_i^2 + N_j^2}$ -1 e ser menor que **2s**.

Neste caso a matriz acumuladora possui apenas uma dimensão com o parâmetro **t**. A matriz **elipse[ ]** é utilizada como matriz armazenadora contendo 6 dimensões, podendo ser controlada em número de registros, ou seja quantidade de eixos maiores aceitáveis.

O número de pontos no contorno da elipse entre o intervalo $\left(0^o, 360^o\right]$ ou $\left(0, 2\pi\right]$ também pode ser configurável, está configuração atua diretamente no tamanho da matriz acumuladora.

3.2.2-Algoritmo

O **algoritmo 3.3** desenvolvido para detecção de elipses é mostrado a seguir, sendo:

I(x_1,y_1)-pixel da imagem, representando um dos pontos extremos do eixo maior;

I(x_2,y_2)-pixel da imagem, representando um dos pontos extremos do eixo maior;

I(x,y)-pixel da imagem , representando um ponto qualquer na elipse;

M(t)-matriz acumuladora;

elipse[t,s,x_0,y_0,α,v]-matriz armazenadora que guarda todos os parâmetros das uma elipses mais votadas mais **v**(número de votos obtidos).

elipse_hough()

Guardar todos os pixels num array de uma dimensão

Limpar acumulador

Para cada **I(x_1,y_1)**, de **x**=0 e **y**=0 até **x**=**Ni** e **y**=**Nj**

Para cada outro **I(x_2,y_2)**, de **x**=0 e **y**=0 até **x**=**Ni** e **y**=**Nj**

Se a distância entre (x_1,y_1) e (x_2,y_2) for maior do que a distância mínima requerida para o eixo maior então

A partir de um par de pixels (x_1,y_1) e (x_2,y_2), calcular centro, inclinação e tamanho do maior eixo para a assumida elipse

$$s = \left[(x_2 - x_1) + (y_2 - y_1)^{\frac{1}{2}} / 2 \right]$$

$$x_0 = (x_1 + x_2)/2$$

$$y_0 = (y_1 + y_2)/2$$

$$\alpha = \arctan[(y_2 - y_1)/(x_2 - x_1)]$$

Para cada terceiro pixel **I(x,y)**, de **x**=0 e **y**=0 até **x**=**Ni** e **y**=**Nj**

$\rho=((x-x_0)^2 + (y-y_0)^2)^{1/2}$

Se ρ for maior do que zero e menor que **s** então

Calcule o tamanho do menor eixo:

$t=(s^2 \rho^2 sin^2\tau)/(s^2-\rho^2 cos^2\tau)$

Incremente o acumulador **M(t)** para este tamanho de eixo menor de uma unidade.

Faça laço até final da imagem.

Encontre o máximo elemento no acumulador. O índice do acumulador é o possível tamanho do menor eixo.

Armazene os cinco parâmetros da elipse mais votada no array **elipse[t,s,x_0,y_0,α,v]**e seu respectivo voto.

Faça laço até final da imagem.

Faça laço até final da imagem

Encontrar maior voto no array **elipse[t,s,x0,y0,α,v]**

Resgatar de **elipse[t,s,x_0,y_0,α,v]** os 5 parâmetros da elipse com maior voto

Para cada ângulo, de τ=0 até $\tau<2\pi$

Calcular cada **x** e **y** da elipse detectada

$$x' = s\cos\tau, \quad y' = t\sin\tau$$

$$x = x_0 + x'\cos\alpha - y'\sin\alpha$$

$$y = y_0 + x'\sin\alpha - y'\cos\alpha$$

Faça laço até $\tau=2\pi$

Mostrar elipse detectada

Algoritmo 3.3- Detecção dos parâmetros das elipses.

Diferentemente dos algoritmos anteriores, o algoritmo para detecção de elipses tanto armazena na matriz acumuladora quanto executa o processo inverso, ou seja, calcula o conjunto de parâmetros mais votado e mostra na tela a elipse detectada em uma só etapa. Para encontrar o maior voto, foi utilizado o procedimento **busca_maiores**() (**algoritmo 2.2**).

3.2.3-Testes com elipses

Foi desenvolvido um programa para detecção de elipses utilizando o **algoritmo 3.3**.A seguir comentaremos alguns testes realizados para verificar a correção do algoritmo, neles a imagem detectada é mostrada em vermelho. Para todos os testes é usado o intervalo de 1 em 1 pixel para **t (eixo menor)**.

No primeiro teste a imagem de entrada, uma elipse horizontal, é mostrada na **figura 3.24**. No resultado obtido na **figura 3.25**, foi feita definido o valor de 124 pixels para **eixo maior**. Foram detectados **eixo menor** de 60 pixels, coordenadas de centro (70,94) e zero grau de inclinação.

Figura 3.24 -Teste 1:Imagem de entrada com uma elipse com zero grau de inclinação

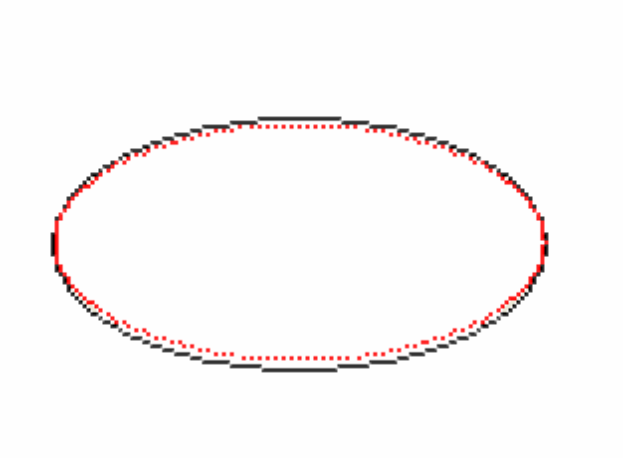

Figura 3.25- Teste 1:Imagem de resultado

O segundo teste identifica imagem de uma elipse inclinada mostrada na **figura 3.26** A **figura 3.27** mostra seu resultado com inclinação de 16 graus com 103 pixels de **eixo maior**, **eixo menor** com 42 pixels e coordenadas de centro de (82,92).

Figura 3.26-Teste 2:Imagem de entrada com uma elipse rotacionada.

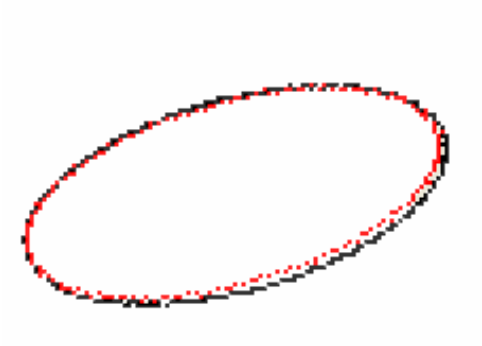

Figura 3.27 - Teste2:Imagem de resultado da detecção de uma elipse rotacionada

No terceiro teste é usada a imagem da **figura 3.28**, que trata-se de uma elipse vertical. A elipse detectada é mostrada na **figura 3.29**. Na sua execução, foi informado **eixo maior** de 99 pixels e a partir disso foram detectados **eixo menor** de 36 pixels, coordenadas de centro de (141,69) e 89 graus de inclinação.

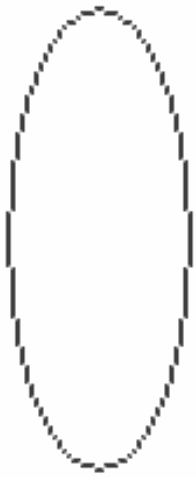

Figura 3.28- Teste3:Imagem de entrada de uma elipse vertical.

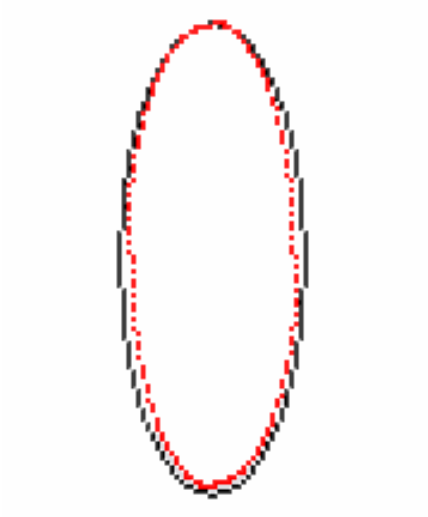

Figura 3.29- Teste3:Resultado da detecção de uma elipse vertical.

No quarto teste busca-se verificar a detecção com diferentes direções em relação aos eixos de definição da imagem. A imagem de entrada é a mostrada na **figura 3.30**. A detecção obtida é vista na **figura 3.31**, é usado **eixo maior** de 86 pixels, **eixo menor** de 48 pixels, centro da elipse (64,62) e 138 graus ou -42 graus de inclinação.

Figura 3.30- Teste4:resultado da detecção de uma outra elipse rotacionada.

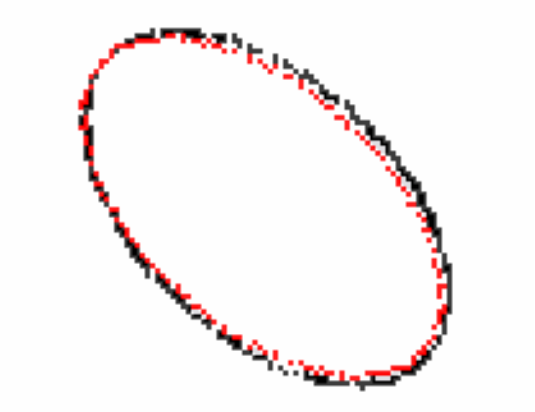

Figura 3.31- Teste4:Imagem de entrada de uma outra elipse rotacionada.

Os dados oriundos da execução de cada imagem desta seção de testes são armazenados em arquivo, como mostra a **figura 3.32**, onde está sendo apresentado apenas o arquivo do teste 4. Com as informações extraídas da **figura 3.30**.

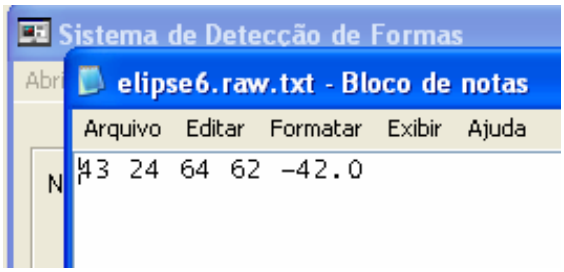

Figura 3.32- Dados de vetorização do teste 4. Respectivamente: metade do eixo maior, metade do eixo menor, coordenadas de centro, e a inclinação da elipse.

O quinto teste usa a imagem da **figura 3.33** e o resultado é mostrado na **figura 3.34**. Este teste mostra que é possível detectar elipses com falhas. No entanto se estas falhas se apresentam na região dos pontos extremos do **eixo maior**, a detecção correta não ocorrerá. Isso se deve ao algoritmo adotado

nesse trabalho que calcula o centro da elipse e demais parâmetros através dos pontos extremos do **eixo maior**.

Neste último teste foi informado um **eixo maior** de 124 pixels, e detectados **eixo menor** de 60 pixels, centro da elipse de (70,94) com zero graus de inclinação.

Figura 3.33- Teste 5:Imagem de entrada de uma elipse com falhas.

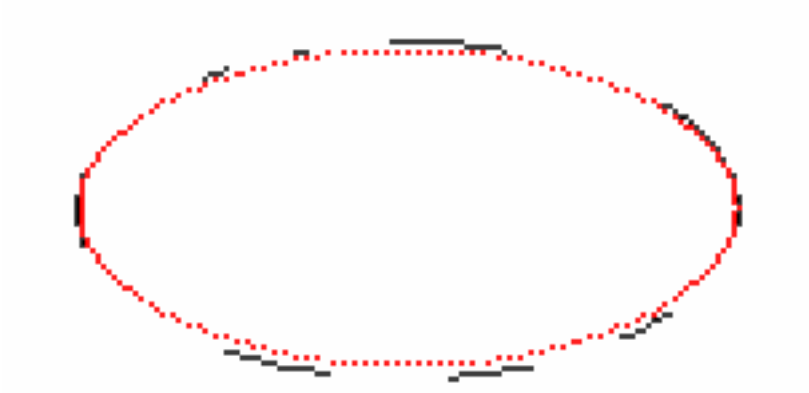

Figura 3.34- Teste 5:Resultado da detecção de uma elipse com falhas.

Todas as imagens testadas foram desenhadas usando o programa PaintBrush e as rotações foram feitas no programa Photoshop 7.0. Elas são do tipo . raw com dimensão e tempo de execução conforme a **tabela 3.5**.

Tabela 3.5- Dimensões das imagens e tempo de execução de cada teste.

Teste	Dimensão	Tempo de execução(s)
Teste 1	192x140	0,094
Teste 2	192x140	0,078
Teste 3	192x140	0,062
Teste 4	192x140	0,078
Teste 5	192x140	0,047

3.2.4-Conclusão

Os algoritmos desenvolvidos nesta seção mostraram-se muito eficientes na detecção de elipses, como mostrado nos testes. Sua eficiência também foi comprovada no caso de elipses com muitas falhas. Apenas não detecta corretamente caso a falha esteja na região de pontos extremos do **eixo maior** da elipse. Da forma que foi implementado este algoritmo não é adequado para detecção de mais de uma elipse. Para ser adequado a falhas no contorno também no eixo maior, os fórmulas devem ser modificadas para se referirem tanto ao eixo maior quanto ao eixo menor. Para que seja possível detectar mais de uma elipse por vez pode-se usar o desenvolvimento original de [XIE2002] conforme comentado no final da seção 3.2.

3.3- Transformada de Hough para parábolas

Parábola (**figura 3.35**) é uma curva plana, lugar geométrico dos pontos de um plano que são eqüidistantes de um ponto fixo **F** e de uma reta fixa **k**.

No plano cartesiano, pode ser definida como a curva plana formada pelos pontos **P(x,y)**, tais que:

$$p = PP' = PF \text{ ,}$$

onde:**p** = distância entre os pontos **P** e **F** ou à distância entre o ponto **P** e a reta **k** (diretriz).

Outro **parâmetro** é a distância do foco **F** à reta diretriz **k** (**2d).**

Esta forma cônica também pode ser definida como o gráfico de uma função polinomial do segundo grau do tipo $y = gx^2 + hx + i$ **ou** $x = gy^2 + hy + i$, com **g≠0**. Sempre que **g>0**, a parábola terá concavidade voltada para cima ou para direita, e quando **g<0** a parábola terá concavidade voltada para baixo ou para esquerda.

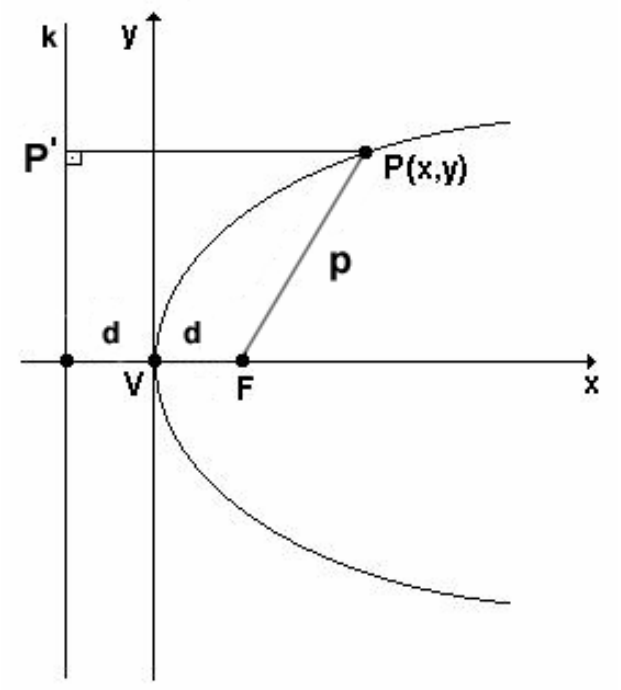

Figura3.35- Elementos de definição da parábola

No caso de uma equação reduzida da parábola de eixo horizontal e vértice na origem (**figura 3.35**), consideremos os pontos: **F(d, 0)** - foco da parábola, e **P(x,y)** - um ponto qualquer da parábola. Usando a fórmula da distância entre pontos no plano cartesiano:

$$\sqrt{(x-d)^2+(y-0)^2} = \sqrt{(x+d)^2+(y-y)^2}$$

Desenvolvendo e simplificando a expressão acima, chegaremos à equação reduzida da parábola de eixo horizontal e vértice na origem, a saber:

$$y^2 = 4dx,$$

onde **d** é a distância focal da parábola.

Para parábolas de eixo horizontal e vértice em um ponto qualquer, ou seja, se o vértice da parábola não estiver na origem e, sim, num ponto ($\mathbf{x_0}$, $\mathbf{y_0}$), a equação acima se torna:

$$(y-y_0)^2 = 4d(x-x_0)$$

Para parábola de eixo vertical e vértice na origem, se a parábola tiver vértice na origem e eixo vertical, a sua equação reduzida será:

$$x^2 = 4dy$$

Para parábola de eixo vertical e vértice no ponto ($\mathbf{x_0}$, $\mathbf{y_0}$), ou seja, se o vértice da parábola não estiver na origem, e, sim, num ponto ($\mathbf{x_0}$, $\mathbf{y_0}$), a equação fica:

$$(x-x_0)^2 = 4d(y-y_0)$$

Uma parábola também pode ser representada na forma polar através da seguinte expressão:

$$\rho = \frac{2d}{1-\cos\beta},$$

onde **d**= distância do foco ao vértice (**v**), **ρ**= distância do foco a cada ponto da parábola, sendo referente ao **p** na **figura 3.35**, e **β**= ângulo que **ρ** faz com o eixo horizontal, sendo **(ρ,β),** coordenadas polares da parábola.

A parábola pode ser desenhada através das seguintes expressões

$$x = \rho \cos \beta$$

$$y = \rho \sin \beta \text{ ,}$$

onde $\mathbf{x_f}$ e $\mathbf{y_f}$ são coordenadas do foco da parábola, e **x** e **y** são coordenadas de um ponto qualquer da parábola. Mas para desenhar uma parábola com inclinação (**figura 3.36**), cada ponto deve ser multiplicado pela matriz de rotação (**tabela 3.2**), neste caso o **α** está sendo substituído por **ω**.

$$x' = x \cos \omega + y \sin \omega$$

$$y' = y \cos \omega - x \sin \omega \text{ ,}$$

e para transladar é necessário adicionar a coordenada de centro do foco ($\mathbf{x_f}$, $\mathbf{y_f}$)**.**

$$x' = x_f + \rho \cos \beta \cos \omega - \rho \sin \beta \sin \omega$$

$$y' = y_f + \rho \cos \beta \sin \omega + \rho \sin \beta \cos \omega$$

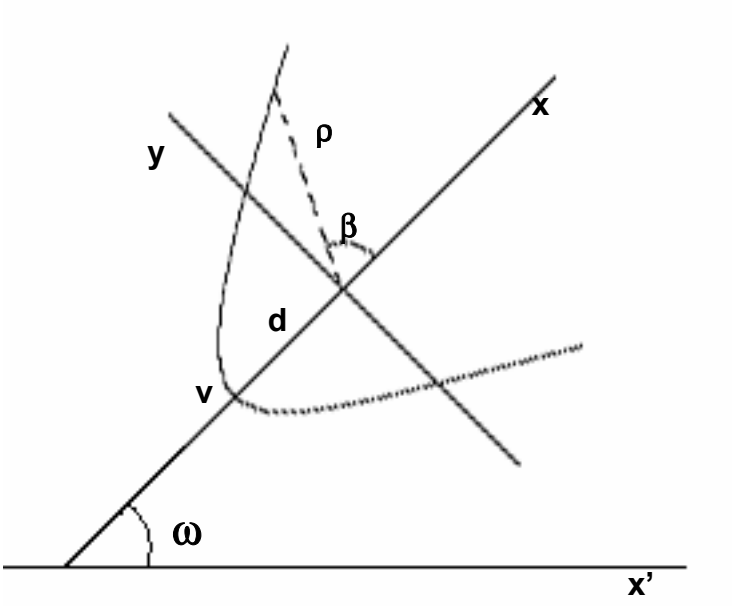

Figura 3.36- Parábola rotacionada.

Na detecção de parábolas por transformada de Hough, seria necessário calcular todos os parâmetros de uma parábola e passar essas informações para o espaço de parâmetros. No caso deste tipo de forma cônica seria necessário obter uma matriz acumuladora de 4 dimensões com os seguintes itens: $(\mathbf{x_f}, \mathbf{y_f})$**-**foco**, d-**distância do vértice ao foco e **ω** inclinação.

Para transformar uma parábola em dados no espaço de parâmetros, seguem se os seguintes passos:

1- define-se o intervalo de **d**, ou seja, os valores mínimo e máximo da distância entre o foco e o vértice.

2- através da equação na forma polar é calculado o raio para cada ponto aceso da imagem.

3- possuindo o valor do raio, calculam-se as coordenadas de foco multiplicando cada ponto da parábola pelo ângulo de inclinação num intervalo de $\left(0^o, 360^o\right]$ graus ou $(0, 2\pi]$ radiano de acordo com a fórmula abaixo:

$$x_f = x - \rho\cos\beta\cos\omega + \rho\sin\beta\sin\omega$$

$$y_f = y - \rho\cos\beta\sin\omega - \rho\sin\beta\cos\omega$$

A **figura 3.37** mostra uma ilustração da metodologia aplicada a cada ponto da imagem, considerando três funções: variação de foco, variação de inclinação e variação de tamanho. Respectivamente da esquerda para direita: ponto na imagem original, variação de foco, variação de inclinação e variação de tamanho.

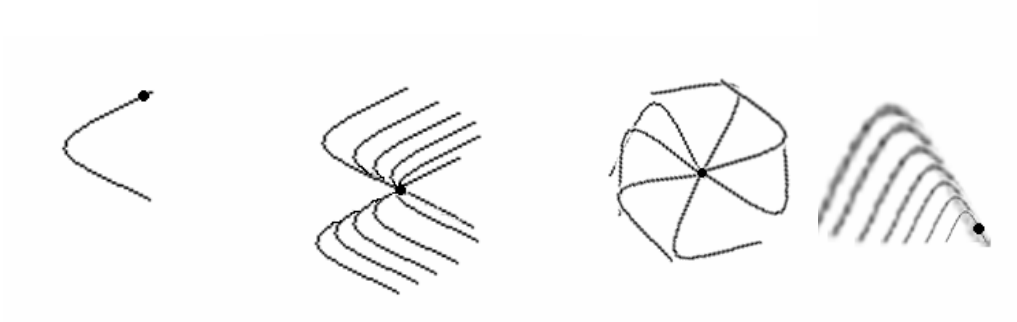

Figura 3.37- Fases executadas na transformada de Hough para parábolas.

3.3.2-Discretização da matriz acumuladora para parábolas

São atribuídos às coordenadas de foco valores entre 0 e a dimensão da imagem. À distância do foco ao vértice, **d**, é atribuído um intervalo de um valor mínimo a um valor máximo dependendo da entrada do usuário, ou um intervalo pré-estabelecido pelo próprio algoritmo. O parâmetro de inclinação da parábola, ou seja, seu ângulo de rotação, varia entre 0 e 360 graus ou de 0 a 2π radiano. Da mesma forma que em retas, círculos e elipses, a precisão da identificação de uma forma dependerá do tamanho desses intervalos, ou seja, quanto menor o tamanho dos intervalos, maior será a precisão e o tempo de execução.

3.3.3-Algoritmo

O **algoritmo 3.4** descreve a forma para detecção de parábolas por força bruta que foi utilizado neste trabalho:

I(x,y)- pixel da imagem.
M(x_0,y_0,d, ω)- matriz de parâmetros;

parabola_hough()
Guardar todos os pixels num array de uma dimensão
Limpar acumulador **M(x_0,y_0,d, ω)**
Para pixel **I(x,y)**, se este pixel **I(x,y)** estiver aceso na imagem
Para **d**>mínimo até **d**<máximo
Para β=0 até β<2π
$\rho = 2d/1-\cos\beta$
Para ω=0 até ω<2π
$x_f = x - \rho\cos\beta\cos\omega + \rho\sin\beta\sin\omega$
$y_f = y - \rho\cos\beta\sin\omega - \rho\sin\beta\cos\omega$
M(x_0, y_0,d, ω)=M(x_0, y_0,d, ω)+1
Faça para os ângulos de β=0 à 2π
Faça para os ângulos de ω=0 à 2π
Faça para todos os **d**´s de mínimo à máximo
Faça para todos os pixels da imagem

Algoritmo 3.4- Detecção dos parâmetros de parábolas

O algoritmo de detecção de parábolas é semelhante ao de círculos, a diferença está na variação de raio, por isso, passam a existir inclinações diferentes e conseqüentemente, um quarto parâmetro que forma um espaço de parâmetros de quatro dimensões.

Inicialmente é fornecido ao programa um intervalo de valores relacionados à distância **d** entre o foco e o vértice (tamanho). A partir disso, são calculados na **linha 6,** todos os **ρ's** possíveis para cada ponto aceso da imagem no intervalo $0^o \leq \beta < 360^o$ ou $0 \leq \beta < 2\pi$ por conseguinte nas **linhas 8** e **9** são calculadas todas as coordenadas de foco possíveis para cada ρ calculado no intervalo $0^o \leq \omega < 360^o$ ou $0 \leq \omega < 2\pi$. A cada interação todos esses valores são

calculados e a célula de índice correspondente a esse conjunto de valores é incrementada de uma unidade. Mantendo o laço até não existirem mais pixels acesos na imagem.

O Algoritmo abaixo extrai o conjunto de parâmetros mais votado, a célula da matriz acumuladora mais votada é identificada através do **algoritmo 2.2** apresentado anteriormente.

IS(x,y)-pixel imagem de saída
x_menor-coordenada horizontal do primeiro ponto limite da parábola;
y_menor- coordenada vertical do primeiro ponto limite da parábola;
x_maior- coordenada horizontal do segundo ponto limite da parábola;
y-maior-coordenada vertical do segundo ponto limite da parábola;
dentro-será **false** se o pixel em questão não estiver na parábola, será **true** se o pixel estiver na parábola.
busca_aceso (algoritmo 2.3)-função que recebe **true** se através de uma aproximação fornecida encontrar-se um pixel aceso na imagem original, recebe **false** de não encontrar pixel aceso na vizinhança;

parabola_inversa()
Inicializar **x_menor**, **y_menor**,**x_maior**,**y_maior**,**anterior_x**, **anterior_y**
Para cada célula na matriz acumuladora **M(x_0,y_0,d, ω)**
Para **β =0** até **β = 2π**
ρ=(2*d)/(1-cosβ)

$$x = x_f + \rho\cos\beta\cos\omega - \rho\sin\beta\sin\omega$$

$$y = y_f + \rho\cos\beta\sin\omega + \rho\sin\beta\cos\omega$$

Se (**x,y**) estiver no domínio da imagem então
Se o pixel **I(x,y)** estiver aceso na imagem original então
Se a variável **dentro** for igual a **false**, significa que o pixel não está sobre a parábola
x_menor = **x**, **y_menor** = **y**
dentro=true
O **pixel IS(x,y)** na imagem de saída é aceso
Senão, se o pixel **I(x,y)** não estiver aceso na imagem original então
aceso=busca_aceso()
Se **aceso** for igual a **true** então
Se a variável **dentro** for igual a **false** então
x_menor = **x**, **y_menor** = **y**
dentro=true
O **pixel IS(x,y)** na imagem de saída é aceso
Senão, se **aceso** for igual a **false** e a variável dentro for igual a **true** então
x_maior = **anterior_x**, **y_maior** = **anterior_y**
dentro=false
anterior_x=x, **anterior_y=y**
Faça o laço para todos os ângulos **β** de 0 a 2π
Para cada parábola identificada
Armazenar os pontos limite da parábola **(x_menor,y_menor)** e **(x_maior,y_maior)**
Faça o laço para todas as parábolas identificadas
Faça o laço para todas as células da matriz acumuladora.

Algoritmo 3.5- Definição dos elementos da parábola no espaço da imagem.

Para cada conjunto de parâmetros mais votado são calculados os pixels pertencentes à parábola que mais coincide com a imagem de entrada. Não seria necessário desenhar a parábola logo que encontrada, seus parâmetros

poderiam apenas ser armazenados em um banco de dados e ser desenhado sempre que fosse necessário.

Para gerenciar o início e término de parábolas também é utilizada a variável **dentro** como no algoritmo de círculos que recebe **true** quando está percorrendo parábola e o primeiro pixel no sentido horário a partir do ângulo zero, é marcado como ponto inicial. Quando a parábola termina dentro recebe **false** e o pixel anterior é marcado como ponto final.

3.3.4-Testes com parábolas

Foi desenvolvido um programa para detecção de parábolas utilizando os **algoritmos 3.4** e **3.5**, o programa funciona também para imagens com falhas. As imagens detectadas estão apresentadas na cor vermelha, para todos foram utilizados intervalos de 10 graus para **ω** (inclinação) e intervalos de 1 pixel para $\mathbf{x_f}$ e $\mathbf{y_f}$. O valor ou um intervalo de **d** (distância do foco ao vértice) deve ser inserido pelo usuário. As imagens de teste são curvas semelhantes a parábolas, e são geradas através do programa PaintBrush.

No primeiro teste, a imagem de entrada é mostrada na **figura 3.38** e seu resultado é apresentado na **figura 3.39** com **d**=11 pixels, inclinação de 277 graus e coordenadas de foco de (83,95) e pontos inicial e final de (46,138) e (41,39), com 4 pixels de tolerância a erros.

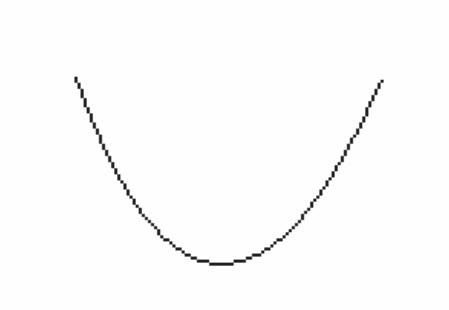

Figura 3.38-Teste 1:Imagem de entrada com uma parábola vertical

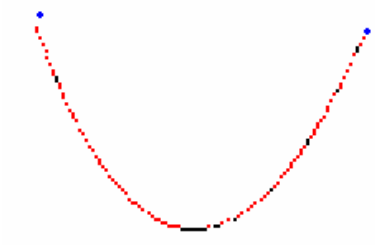

Figura 3.39-Teste 1:Resultado da detecção de uma parábola vertical com concavidade voltada para cima.

No segundo teste a parábola mostrada na **figura 3.40** é imagem de entrada, sendo o resultado obtido representado pela **figura 3.41,** foram detectados para **d**=13: inclinação de 353 graus, coordenadas de foco de (64,76) e pontos inicial e final de (130,118) e (7,119), com 3 pixels de tolerância a erros.

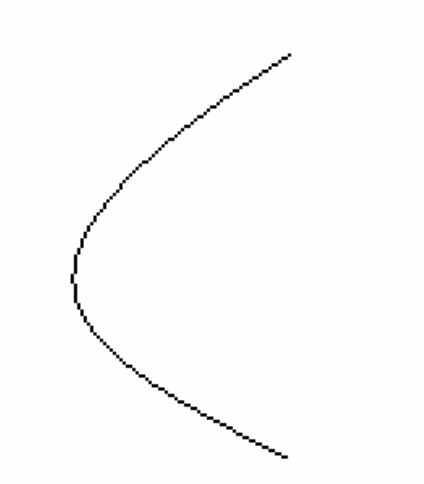

Figura 3.40-Teste 2:Imagem de entrada com uma parábola horizontal

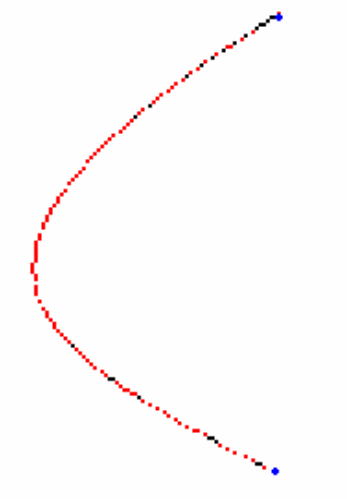

Figura 3.41-Teste 2:Resultado da detecção de uma parábola horizontal.

No terceiro teste considera-se uma parábola com direção qualquer. A imagem de entrada é a mostrada na **figura 3.42.** Foram detectados para **d**=7:

inclinação de 223 graus e coordenadas de foco de (67,62), pontos limite de (22,69) e (62,18), utilizando 2 pixels de tolerância a erros.

Figura 3.42-Teste 3:Imagem de entrada com uma parábola horizontal

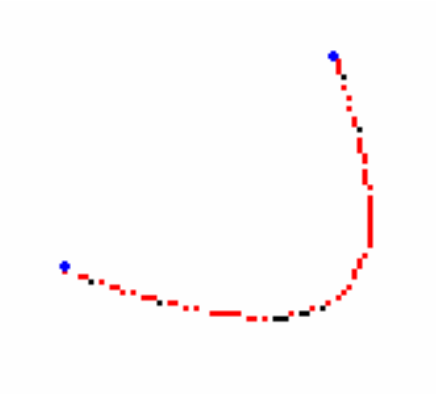

Figura 3.43-Teste 3:Resultado da detecção de uma parábola inclinada com concavidade voltada para cima.

No quarto teste foi identificada uma parábola com extremidades de tamanho diferente. A imagem de entrada é representada pela **figura 3.44** e o resultado mostrado na **figura 3.45.** Para d=12 pixels foram detectados: 353 graus de inclinação, coordenadas de foco de (53,99) e pontos limite de (131,73) e (23,128). A tolerância a erros foi de 2 pixels.

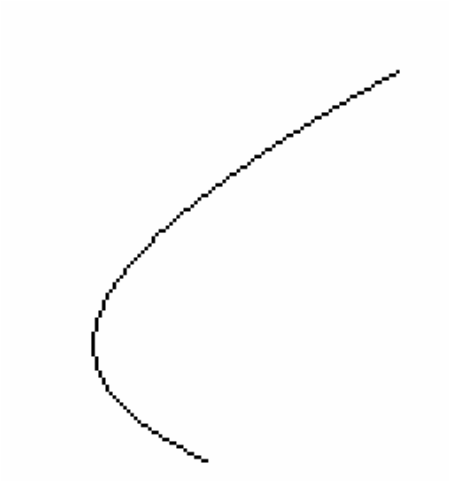

Figura 3.44-Teste 4 - Imagem de uma parábola com extremidades de tamanho diferente.

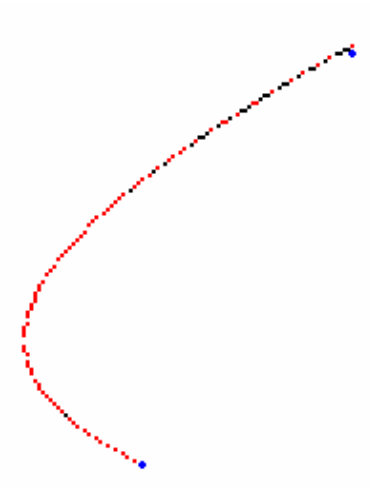

Figura 3.45-Teste 4:Resultado da detecção de uma parábola de extremidades diferentes.

No quinto teste foram detectados para **d**=5: inclinação de 333 graus e coordenadas de foco de (79,74), pontos limite de (75,159) e (24,107), e a tolerância a erros para este teste é de 3 pixels.

Figura 3.46-Teste 5:Imagem de entrada com uma parábola horizontal

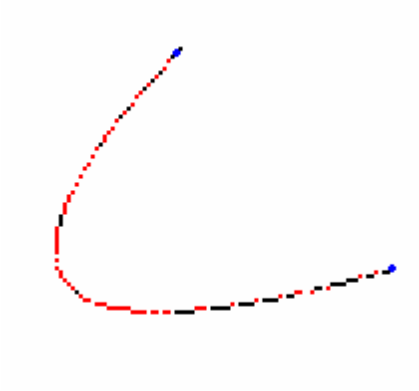

Figura 3.47-Teste 5:Resultado da detecção de uma parábola vertical com concavidade voltada para cima.

Como em todas as seções de teste deste trabalho, para parábolas também é apresentado um exemplo de arquivo (**figura 3.48)** gerado pelo sistema e que possui dados da forma geométrica encontrada para que seja possível a reconstrução da mesma.

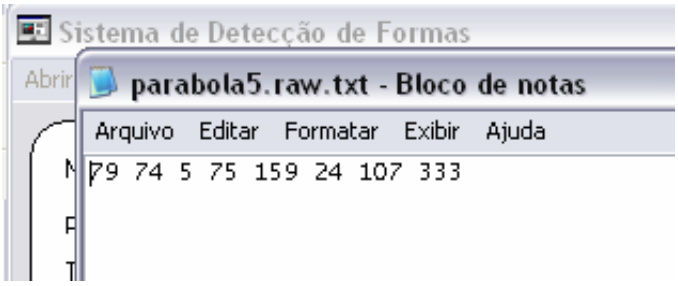

Figura 3.48- Dados de vetorização do teste 5. Respectivamente: coordenadas do foco, raio mínimo, pontos limite e por último sua inclinação.

No teste 6 uma parábola é identificada a partir da imagem de um molde de blusa na **figura 3.49** e o resultado é mostrado na **figura 3.50**. Seus dados são: coordenadas de foco (135,84), **d**=51 pixels, inclinação de 323 graus e pontos limite de (129,144) e (19,62), com 3 pixels de tolerância a erros. A identificação das retas desse molde é mostrada no capítulo 4.

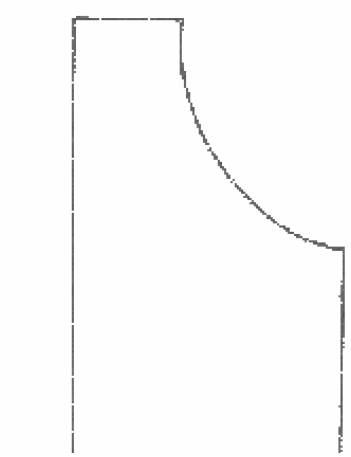

Figura 3.49- Teste 6- Imagem de um molde (corte de blusa)

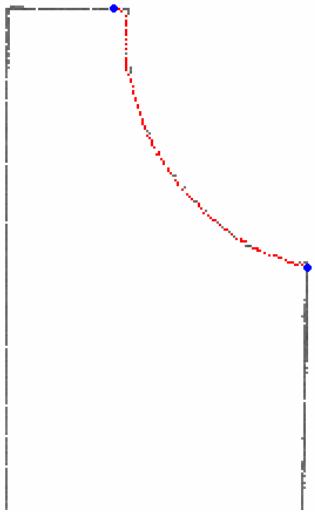

Figura 3.50- Teste 6-Resultado da detecção de uma parábola em um molde.

Todas as imagens testadas são do tipo . raw com dimensão e tempo de execução conforme a **tabela 3.6**.

Tabela 3.6- Dimensões das imagens e tempo de execução de cada teste.

Teste	Dimensão	Tempo de execução(s)
Teste 1	192x140	5,39
Teste 2	192x140	6,062
Teste 3	192x140	4,453
Teste 4	192x140	5,281
Teste 5	192x140	5,125
Teste 6	150x284	21,73

3.3.5-Conclusão

Os algoritmos desenvolvidos nesta seção mostraram-se muito eficazes na detecção de parábolas com qualquer inclinação, mesmo que a curva fornecida claramente não seja de uma parábola, como ilustrado nos testes.

No entanto, para que o algoritmo torne-se eficiente é necessário otimizá-lo para que seu tempo de execução diminua. Para isso é indicado saber o ângulo de inclinação da parábola independentemente do uso da matriz acumuladora. [AGU2001] indica uma forma de saber a inclinação de uma elipse através do cálculo da segunda derivada, mas esta solução poderia também resolver a questão da inclinação de parábolas.

Não foram feitas implementações específicas para as hipérboles pois esta forma praticamente não é encontrada nos moldes de confecção, que são o objetivo final dessa linha de pesquisa.

Capítulo 4 –Detecções de várias formas na mesma imagem

Com o intuito de aperfeiçoar os algoritmos de detecção de formas mostrados nesse trabalho e construir uma ligação entre todas essas detecções, uma metodologia de tratamento de formas não primitivas é apresentada.

Esta tem como objetivo, tornar possível detectar uma imagem complexa, ou seja, uma imagem com mais de uma forma cônica presente. Neste caso a melhor forma de vetorizar uma composição de cônicas é dividir para conquistar, ou seja, a detecção de formas se dá por partes, confira o esquema:(**figura 4.1**) a seguir:

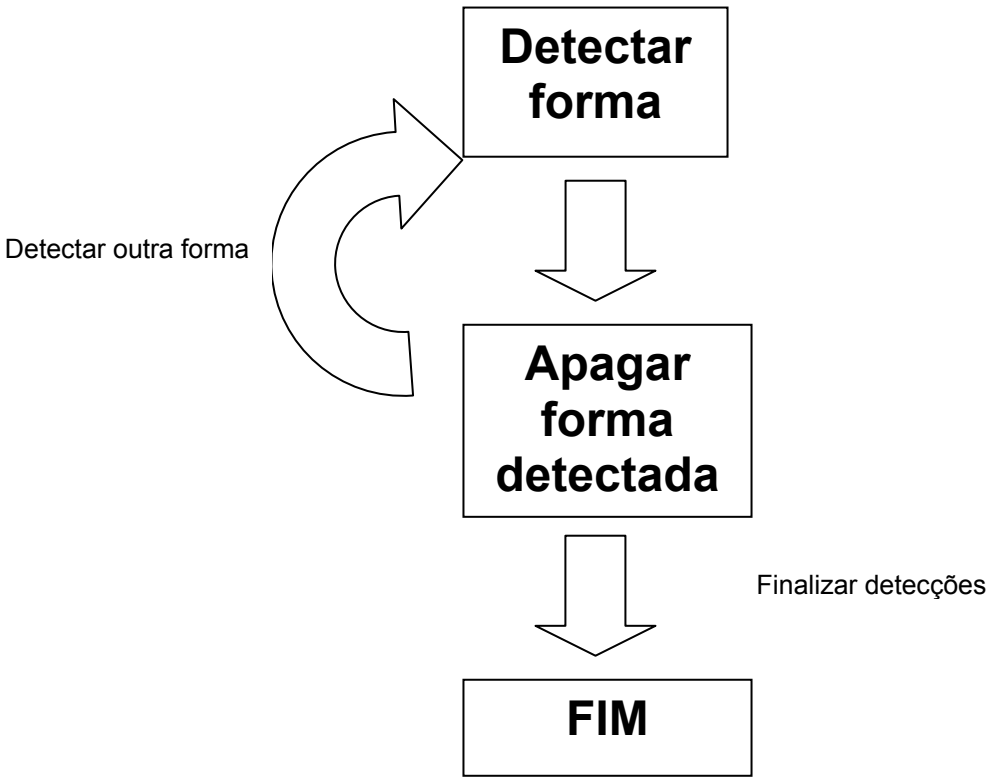

Figura 4.1-Esquema de detecção de várias formas diferentes.

Em uma imagem com retas e parábolas, por exemplo, seriam detectadas primeiramente todas as parábolas, depois essas parábolas seriam apagadas da imagem, para apagar cada pixel da forma já identificada é utilizado uma distância euclidiana ou algébrica. A seguir seriam detectadas todas as retas e posteriormente seriam apagadas da mesma forma. Os parâmetros das retas encontradas e seus pontos iniciais e finais juntamente com o conjunto de parâmetros das parábolas e seus respectivos pontos iniciais e finais seriam armazenados para que houvesse a possibilidade de redesenhá-la sempre que necessário, ou seja, seria a própria forma vetorizada da imagem, o resultado do processo.

A seguir são mostrados alguns exemplos:

O primeiro exemplo mostra uma imagem formada por parábolas e retas. As **figuras 4.2** a **4.5** mostram as seqüências de suas respectivas detecções, forma por forma. Houve a detecção de uma parábola, em seguida a mesma foi apagada para que a primeira reta pudesse ser detectada e assim por diante. Para a detecção da parábola foram usados 2 pixels de tolerância a erros e para retas 1 pixel de tolerância. A **tabela 4.1** mostra todos os dados detectados no primeiro exemplo.

Tabela 4.1- Características extraídas da imagem do exemplo 1.

Parábola					
x_f	**y_f**	**d**	**ω**	**Ponto inicial**	**Ponto final**
67	62	7	223	(22,69)	962,18)
Retas					
θ	**ρ**	**-**	**-**	**Ponto inicial**	**Ponto final**
123	40	-	-	(27,72)	(118,129)
152	154	-	-	(65,23)	(117,127)

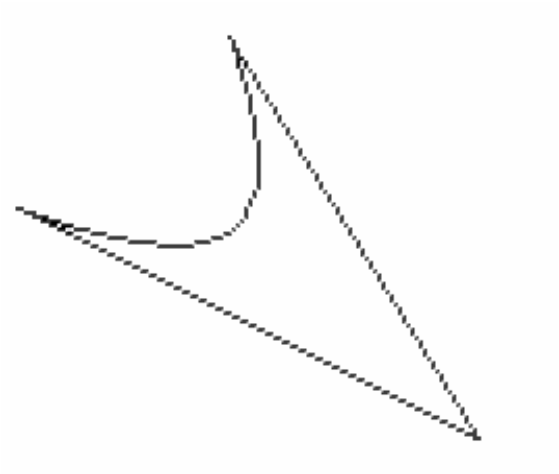

Figura 4.2- Exemplo 1- Imagem de entrada contendo uma parábola e duas retas.

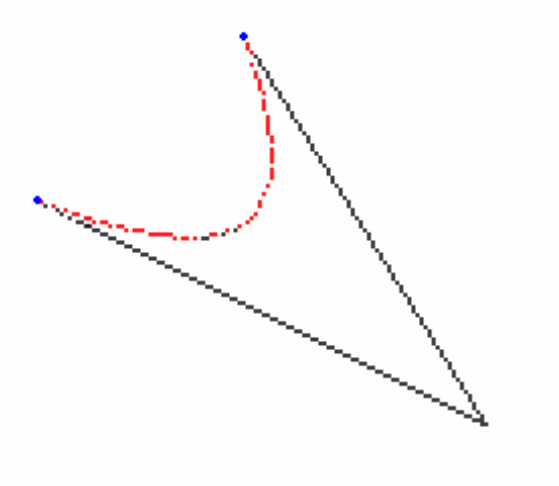

Figura 4.3- Exemplo 1- Parábola detectada.

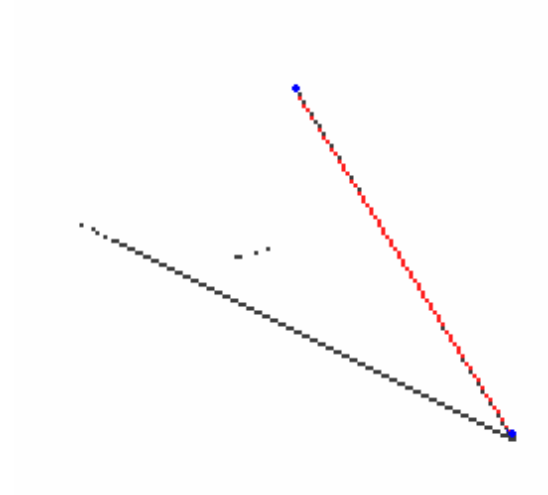

Figura 4.4- Exemplo 1- Primeira reta detectada.

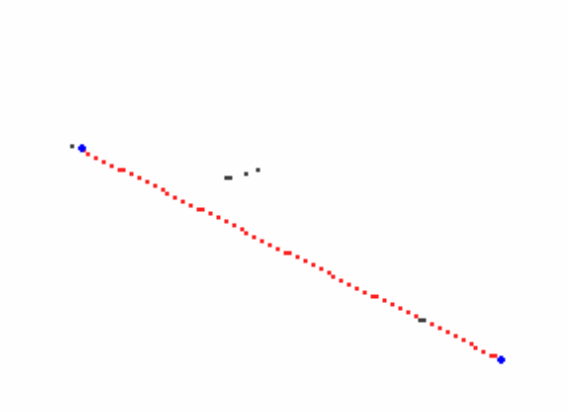

Figura 4.5- Exemplo 1- Segunda reta detectada.

Como mencionado no capítulo 3, o esquema para que tanto os círculos quanto as retas da **figura 3.16,** pudessem ser detectados, é mostrado a seguir:

No exemplo 2, o molde de uma saia é detectado através de duas formas primitivas, círculo e retas.

Os círculos foram identificados utilizando intervalos de 1 pixel para ambos parâmetros (x_0,y_0) e 1 pixel de tolerância a erro. Para as retas foram utilizados intervalos de 1 pixel para o parâmetro ρ e intervalos de 4 graus para θ. Assim como 1 pixel para tolerância a erros. Ao apagar o segundo círculo, na **figura 4.8**, restaram alguns pixels que agora são ruídos, já que não serão mais necessários, restando apenas as duas retas como pixels interessantes.

Durante toda a detecção do molde, seus parâmetros são armazenados em um arquivo mostrado pela **figura 4.10**.

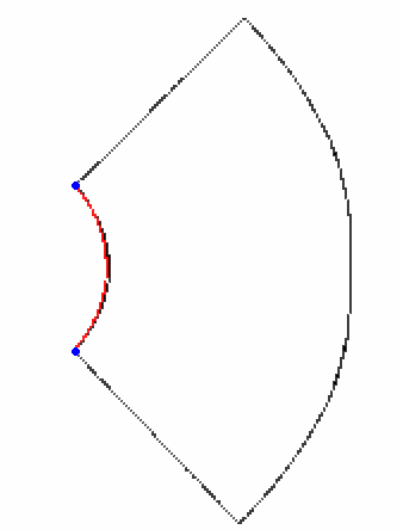

Figura 4.6- Exemplo 2- Primeiro círculo detectado.

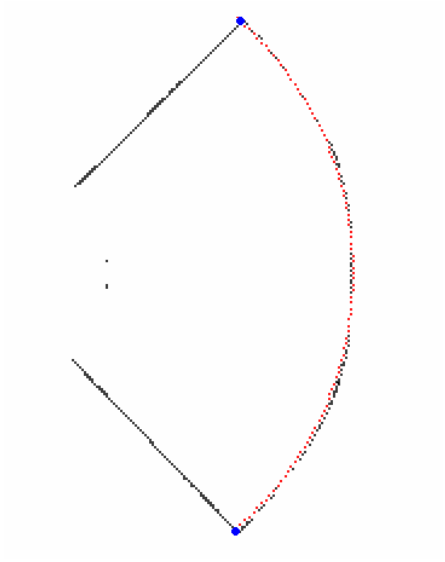

Figura 4.7- Exemplo 2- Após apagar o primeiro círculo, o segundo círculo é detectado.

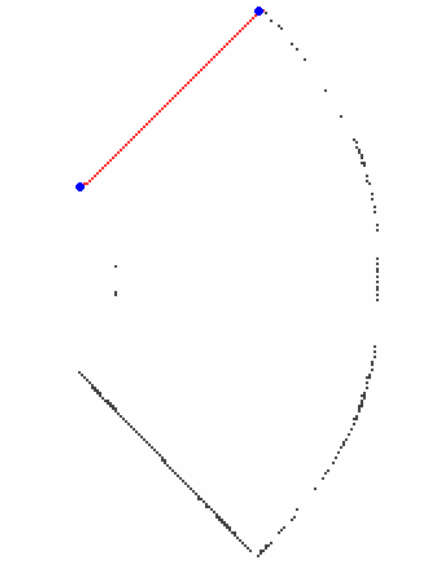

Figura 4.8- Exemplo 2- Após apagar o segundo círculo, a primeira reta é detectada.

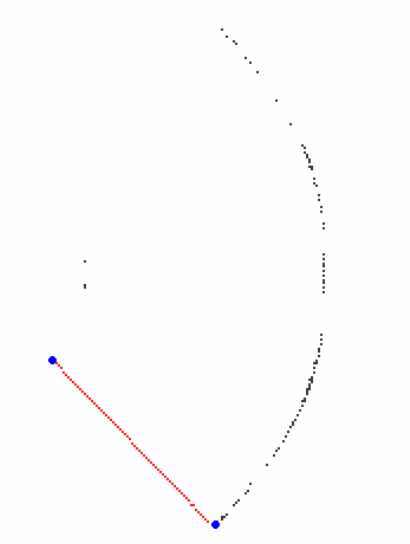

Figura 4.9- Exemplo 2- Por último, a reta restante na imagem é detectada.

A **tabela 4.2** mostra as informações extraídas ao longo de todo o processo de detecção:

Tabela 4.2- Características extraídas da imagem do exemplo 2.

Arcos de círculo				
x_0	**y_0**	**ρ**	**Ponto inicial**	**Ponto final**
109	32	51	(74,70)	(144,69)
109	46	140	(5,140)	(216,136)
Retas				
θ	**ρ**	**-**	**Ponto inicial**	**Ponto final**
45	102	-	(5,141)	(73,71)
-45	53	-	(145,70)	(215,140)

O arquivo mostrado pela **figura 4.10** indica na primeira linha, respectivamente: um par de coordenadas de centro do círculo, raio, um par de coordenadas do primeiro ponto e um par de coordenadas do segundo ponto. Na

segunda linha segue-se a mesma estrutura, mas para outro círculo. Na terceira linha é mostrado um conjunto de parâmetros de reta sendo respectivamente: ângulo de inclinação da reta, distância perpendicular da reta a origem, um par de coordenadas do primeiro ponto e um par de coordenadas do segundo ponto. A quarta linha também se refere aos parâmetros de uma reta.

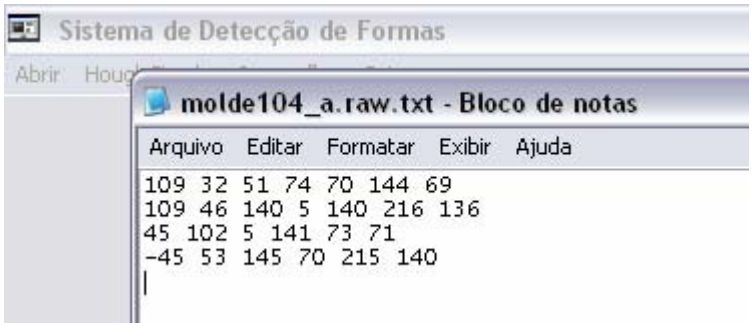

Figura 4.10-Arquivo com parâmetros oriundos da execução do exemplo 2.

O exemplo 3 combina a detecção de uma parábola com 3 retas. A imagem de entrada foi mostrada na **figura 3.49** no capítulo 3 e o resultado das detecções podem ser observados nas **figuras 4.11 a 4.12**. Para uma parábola com **d**=51 pixels, foram detectados foco(127,61), pontos limite de (129,144) e (19,62), inclinação de 323 graus e tolerância a erros de 3 pixels.

Neste caso ao invés de detectar reta por reta, foram detectadas as três retas de uma única vez. Foram considerados 2 pixels de tolerância a erros para a parábola, e para retas não foram considerados erros. A **tabela 4.3** mostra todos os dados extraídos da **figura 4.11**.

Tabela 4.3- Características extraídas da imagem do exemplo 3.

Parábola					
x_f	**y_f**	**d**	**ω**	**Ponto inicial**	**Ponto final**
127	61	51	323	(129,144)	(19,62)
Retas					
θ	**ρ**	**-**	**-**	**Ponto inicial**	**Ponto final**
90	15	-	-	(19,16)	(271,16)
90	141	-	-	(127,142)	(271,142)
0	18	-	-	(19,16)	(19,68)

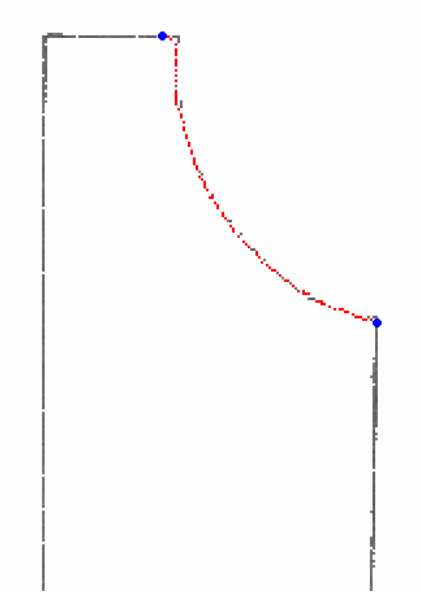

Figura 4.11- Exemplo 3- parábola detectada.

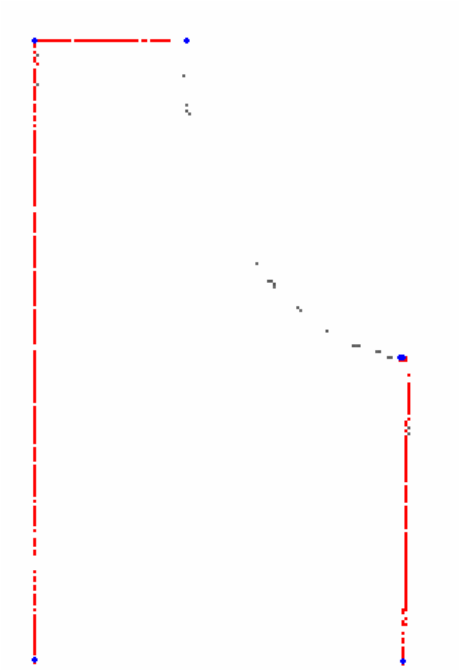

Figura 4.12- Exemplo 3-Resultado da imagem com 3 retas

Para formas mais complexas como curvas de terceiro grau, sugere-se a identificação através da combinação de duas ou mais parábolas.

Todos os exemplos utilizam imagens do tipo .raw e possuem dimensão e tempo de execução de acordo com a **tabela 4.4.**

Tabela 4.4- Dimensões das imagens e tempo de execução de cada exemplo.

Teste	Dimensão	Tempo de execução(s)
Exemplo1		
Parábola 1	192x140	8,641
Reta 1	192x140	0,047
Reta 2	192x140	0,047
Exemplo 2		
Círculo 1	200x220	0,094
Círculo 2	200x220	0,094
Reta 1	200x220	0,016
Reta 2	200x220	0,015
Exemplo 3		
Parábola	150x284	19,469
Reta 1	150x284	0,031
Reta 2	150x284	0,031
Reta 3	150x284	0,031

Capítulo 5 – Outras Aplicações

Além de moldes da área têxtil, várias outras aplicações podem utilizar a transformada de Hough para extração de características. Este capítulo mostra que a utilização da transformada de Hough para formas primitivas e suas possíveis combinações, que poderão simplificar metodologias que utilizam várias técnicas diferentes de processamento de imagem e algoritmos de visão computacional.

5.1.- Ensaio de Dureza

A detecção de círculos por transformada de Hough também é útil na aplicação de ensaios de dureza. Os ensaios de dureza [MEN2003] são feitos por aparelhos chamados durômetros que verificam o quanto um dado material é resistente à penetração. Este trabalho trata apenas dois casos , o ensaio Brinell e o ensaio Vickers.

O ensaio de dureza Brinell consiste em comprimir lentamente uma esfera de aço temperado de diâmetro **D**, sobre uma superfície plana, polida e limpa de um metal, produzindo uma calota esférica de diâmetro **D'**. Em [BER2003] é apresentado um sistema automático de medição do diâmetro **D'** propondo o uso de análise de imagem para o pré-processamento que engloba quatro etapas. Após o pré-processamento, quatro métodos de obtenção de medida são executados com intuito comparativo: Diferença de Coordenadas, Área, Perímetro e Três Pontos Eqüidistantes.

O pré-processamento da imagem usando Hough necessita de apenas duas etapas, mas é importante mencionar que o objetivo da aplicação da transformada de Hough num ensaio de dureza é de apenas mostrar que a transformada pode ser mais uma alternativa de medição tanto de diâmetros quanto de diagonais, mas esta dissertação não enfatiza em nenhum momento a precisão dos valores encontrados.

1- A imagem pode ter seu contorno realçado com o seguinte filtro mostrado na **tabela 4.5** (aplicado pelo programa Adobe Photoshop 7.0):

Tabela 4.5- Filtro passa alta

-1	-1	-1
-1	12	-1
-1	-1	-1

2- É aplicado na imagem um **limiar** de 80, ou seja, numa escala de cinza de 0 a 255, são considerados apenas os pixels com tons de 0 a 80.

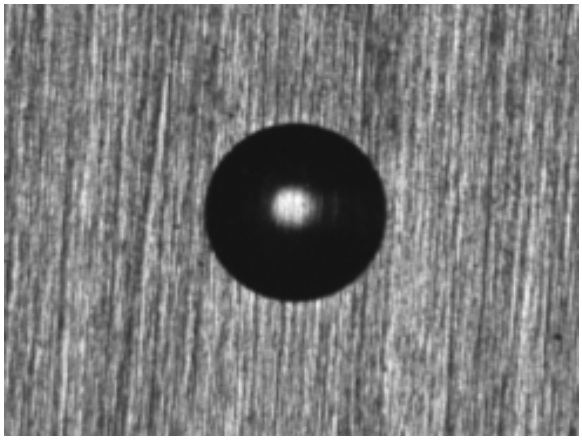

Figura 5.1- Imagem original de um ensaio de dureza Brinell

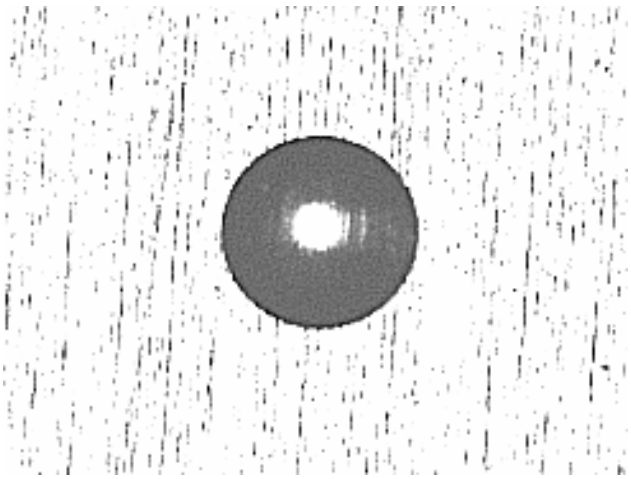

Figura 5.2- Imagem após o uso do filtro passa alta

Para identificar o diâmetro **d** do círculo marcado na imagem foram utilizados os **algoritmos 3.1** e **3.2** com discretização de 1 pixel para as coordenadas de centro, o intervalo de raios fornecido foi de 37,4 a 40,0 pixels. Em 5,14 segundos, as coordenadas de centro do círculo encontradas foram (88,123) e raio 37,4 com 2 pixel de taxa de erro, então **D'** é igual a 74,8 pixels. Consideramos resolução de 72 polegadas/pixel e tamanho da imagem de 245 x 184. O valor real do diâmetro da **figura 5.1** é de 75 pixels , erro de 0,2%.

.

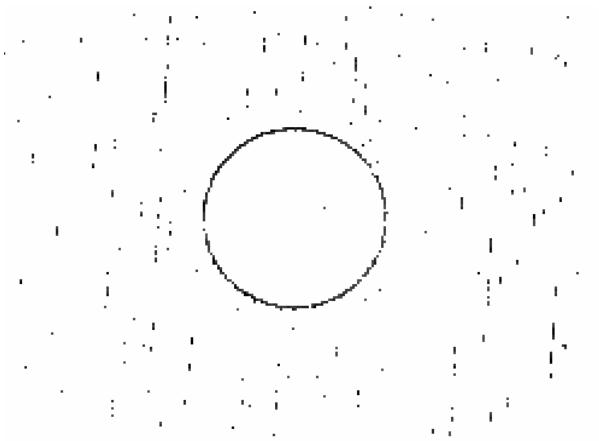

Figura 5.3- Resultado da limiarização.

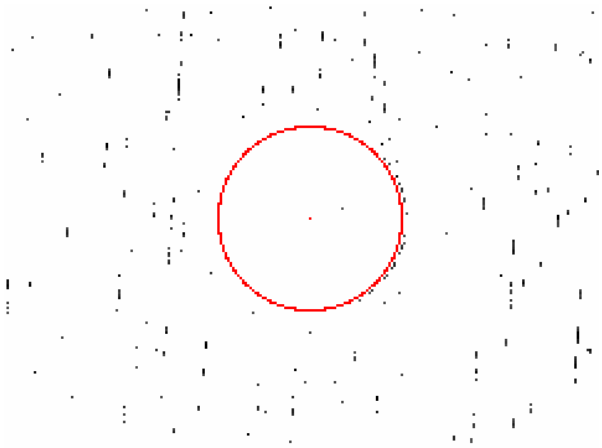

Figura 5.4- Círculo detectado por Hough em vermelho.

Os dados de vetorização desse círculo estão no arquivo apresentado pela **figura 5.5,** onde os pontos zerados significam tratar-se de um círculo completo.

Figura 5.5- Arquivo gerado pelo sistema com os dados do círculo.

Já o ensaio de dureza Vickers consiste em comprimir uma pirâmide de diamante de base quadrada e ângulo entre faces de 136º. Em [MEN2003] na fase de pré-processamento utiliza apenas um thresholding modal. E para obtenção da diagonal foram utilizados 4 métodos para comparação: Diferença de Coordenada, Área, Perímetro e Vértices do Perímetro. A partir da imagem de entrada mostrada na **figura 5.6**, o trabalho presente utiliza, para a fase de pré-processamento, um filtro **tabela 4.6** (aplicado pelo programa Adobe Photoshop 7.0) passa alta (tabela 4.5) com resultado na **figura 5.7** e limiarização de 0 (**figura 5.8**), ou seja, somente os pontos pretos (tom 0) são considerados. Para este teste foi utilizado um intervalo de 6 em 6graus para a inclinação da reta e 1 pixel de tolerância a erros. Na obtenção do valor da diagonal são usados os **algoritmos 2.1** e **2.4**

Tabela 4.6- Filtro passa alta

-1	-1	-1
-1	10	-1
-1	-1	-1

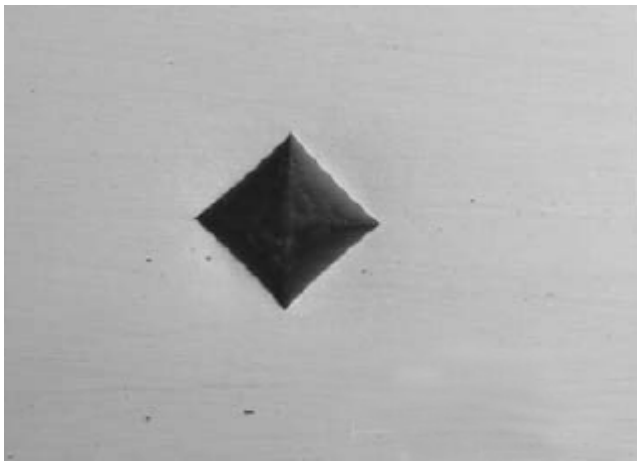

Figura 5.6- Imagem original de um teste de dureza Vickers

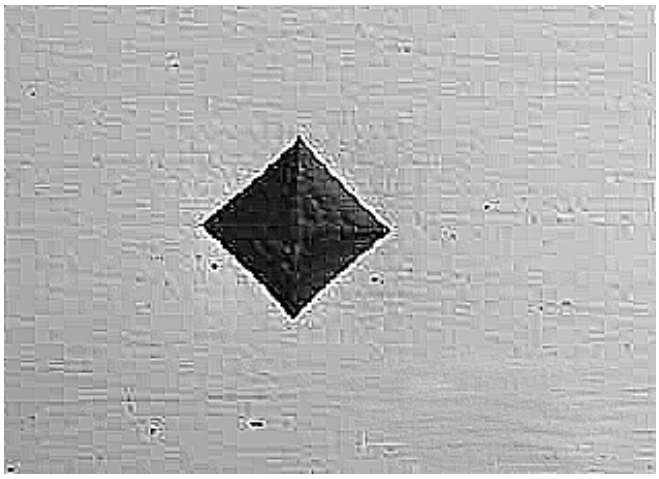

Figura 5.7- Imagem após o uso do filtro passa alta

Utilizando a metodologia do capítulo 4, cada aresta do quadrado foi identificada, juntamente com as coordenadas de cada vértice.

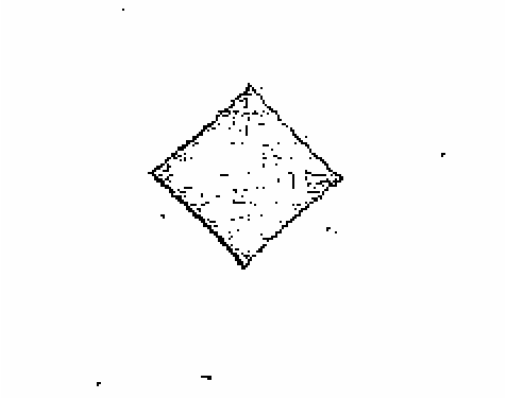

Figura 5.8- Imagem após o uso do limiar de 0

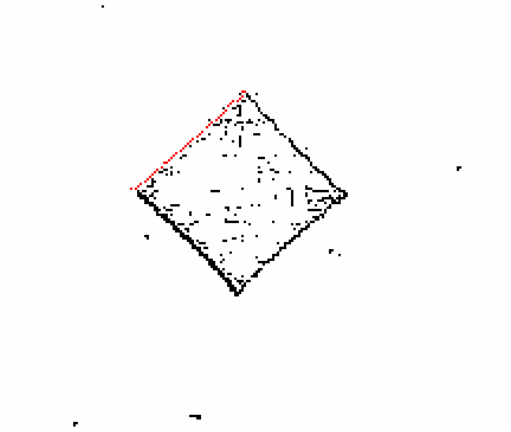

Figura 5.9- Imagem após a identificação da primeira aresta.

Figura 5.10- Imagem após a identificação da segunda aresta.

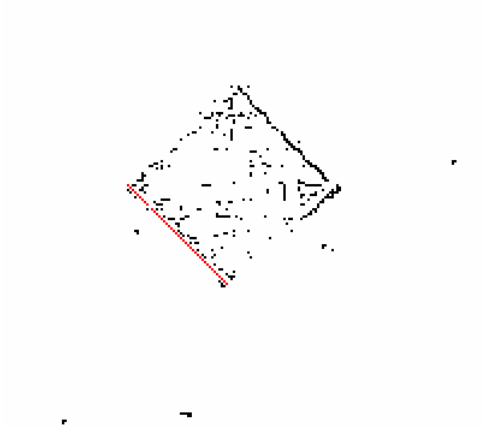

Figura 5.11- Imagem após a identificação da terceira aresta.

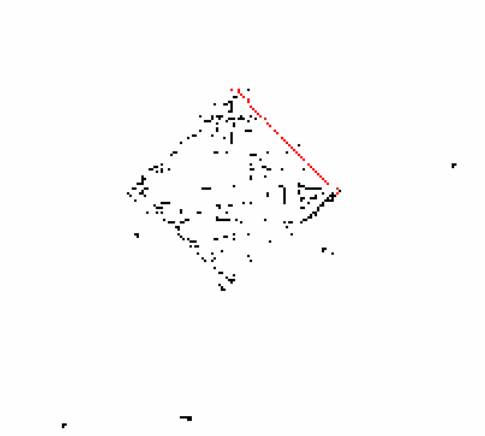

Figura 5.12- Imagem após a identificação da quarta e última aresta.

Cada uma das quatro retas foi identificada em 0,031s. Todos os dados foram armazenados em um arquivo (**figura 5.13**) sendo possível reconstruí-lo ou apenas ter a informação mais importante, ou seja, o valor da diagonal. No arquivo, segue: inclinação da reta, a distância da origem à reta, e os pares de pontos iniciais e finais.

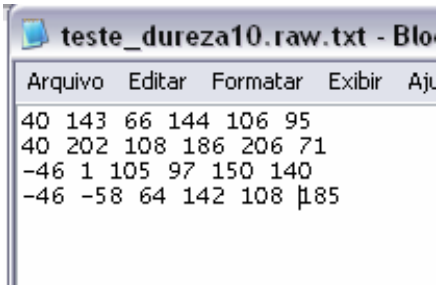

Figura 5.13- Arquivo com dados para vetorização do quadrado.

O cálculo da diagonal será feito a partir do parâmetro ρ detectado em cada aresta. A subtração dos parâmetros ρ entre as retas detectadas que possuem a mesma inclinação, indica o tamanho da aresta. Para entender melhor veja o esquema presente na **figura 5.14**.

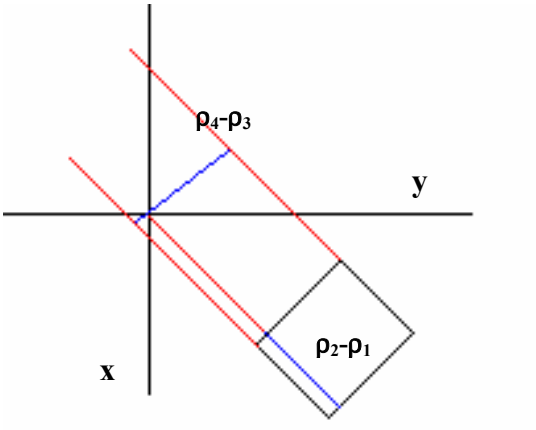

Figura 5.14- Esquema de cálculo da aresta através do parâmetro ρ.

aresta=ρ_2-ρ_1 ou **aresta=ρ_4-ρ_3**

ρ_1=143

ρ_2=202

aresta==ρ_2-ρ_1= 202-143= 59 pixels

ρ_3=1

ρ_4= -58

aresta==ρ_4-ρ_3= -58-1= 59 pixels

Diagonal=$aresta\sqrt{2}$

Diagonal= $59\sqrt{2}$

Diagonal=83,4 pixels

Consideramos resolução de 72 polegadas/pixel e tamanho da imagem de 316 x 227. O valor real encontrado na **figura 5.6** é de 86 pixels, então há um erro de 2,79%

Através da argumentação acima, foi possível identificar o valor da diagonal do quadrado do teste de dureza **Vickers**.

5.2- Controle de Qualidade

Hoje, indústrias procuram melhorar sua produção aumentando a qualidade de seus produtos e diminuindo assim prejuízos e aumentando lucros.

Dessa maneira, a pesquisa com base em controle de qualidade é questão fundamental para o provimento de regularidade e uniformidade de um produto. A indústria farmacêutica é uma das que preza pela qualidade de seus medicamentos, pois um comprimido a ser enviado ao mercado, não pode estar com tamanho fora do padrão, imperfeito ou danificado. Uma forma de automatizar o processo de seleção de comprimidos aptos a irem para o mercado, é utilizar um sistema de captura e processamento de imagem

[MAC2004] a fim de encontrar possíveis defeitos no produto. Este artigo mencionado utiliza o algoritmo de detecção de círculos mostrado nesta dissertação na **seção 3.1.**.

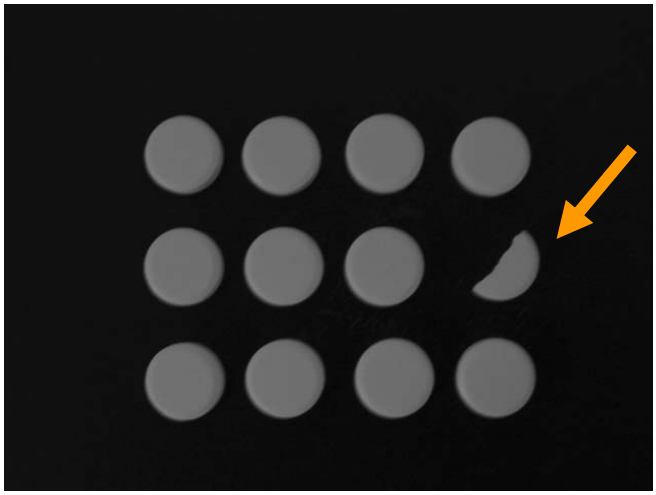

Figura 5.15-Imagem original de pílulas de forma circular, onde uma apresenta falha

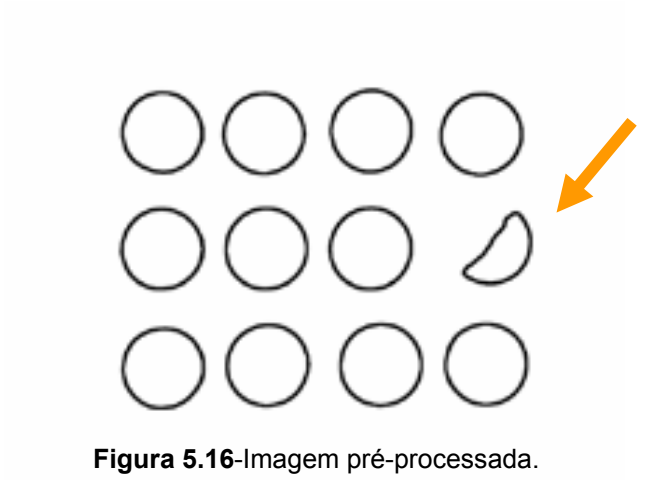

Figura 5.16-Imagem pré-processada.

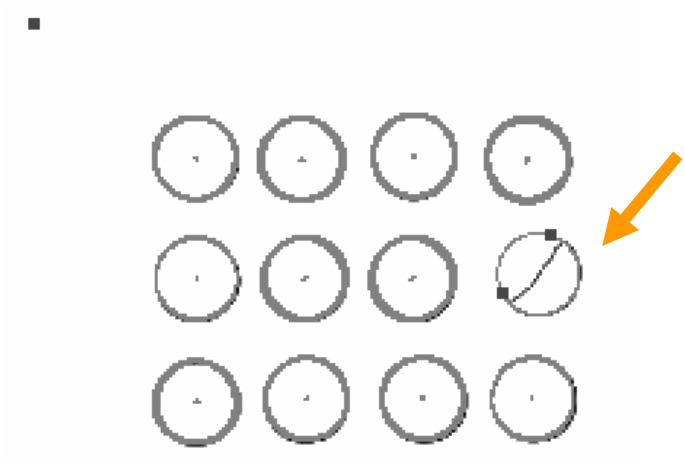

Figura 5.17-Imagem detectada. Os dois pontos no início e fim do fragmento indicam falha.

As **figuras 5.15** a **5.17** mostram uma metodologia de detecção de falhas através da detecção de cada círculo presente na imagem. Através do **algoritmo 3.2** é possível detectar círculos e identificar no caso de arcos de círculo seus pontos iniciais e finais. Caso o ponto inicial e final de um arco de círculo ao término do processo seja diferente da origem (0,0), então quer dizer que na imagem fornecida para detecção existe pelo menos um arco de círculo, ou seja, uma pílula com falha.

Capítulo 6- Conclusões

Para que um molde de vestuário possa ser vetorizado é necessário fazer sua total detecção. Como trata-se na maioria das vezes de formas arbitrárias, como meio de resolver a questão, ao longo desta dissertação são apresentados algoritmos de identificação de formas cônicas, utilizando a técnica de transformada de Hough, que somados mostram como resultado final a detecção de toda a imagem bem como a possibilidade de reconstrução do molde presente na imagem a partir de sua vetorização.

Desse modo é considerado que um molde pode ser formado por um ou mais tipos de formas e sua reconstrução é feita pela soma de todas as formas identificadas.

A pesquisa começa pela identificação de cada forma cônica com exceção de hipérboles que não se apresentam úteis em qualquer dos moldes testados. As principais barreiras aparecem no momento em que se mostra necessário aumentar a dimensão da matriz acumuladora, onde o tempo de execução aumenta significativamente.

Depois da identificação de cada forma ocorre sua exclusão da imagem original por meio de distância algébrica e suas características principais são armazenadas.

Durante a fase de testes, além dos moldes, foram testadas imagens de ensaio de dureza e de comprimidos, diferentemente do objetivo de vetorizar, no ensaio de dureza foi concluído que os algoritmos de retas e de círculos poderiam calcular medidas a partir de um pré-processamento simples. E a aplicação do algoritmo de círculos poderia também fazer controle de qualidade, identificando se uma imagem de comprimidos trata-se de círculos completos ou apenas de arcos de círculo.

A partir do sistema desenvolvido foi possível executar todos os testes com eficácia considerando um limite de tamanho de cada imagem de até 350 x 350 pixels, alcançando o objetivo principal de vetorizar moldes.

6.1- Trabalhos Futuros

Apesar do objetivo deste trabalho ter sido concluído, aprimoramentos podem ser feitos tal como a otimização do algoritmo de identificação de parábolas que utiliza um intervalo de 0 a 360 graus ou 0 a 2π de inclinação o que acrescenta o tamanho da dimensão da matriz acumuladora , aumentando por conseqüência seu tempo de execução. Uma alternativa seria achar a inclinação da reta tangente ao vértice da parábola.

A implementação do algoritmo de identificação para hipérboles, utilizando a transformada de Hough, pode ser feita posteriormente para assim completar todas as formas cônicas. Já que esta forma não foi implementada por não se adequar a nenhum molde.

Para identificar círculos é necessário que o usuário insira o tamanho do raio, ou pelo menos um intervalo, a mesma situação ocorre para elipses e parábolas. Poderia ser feito um estudo sobre a possibilidade de executar a identificação sem haver a inserção dessa informação, ou seja , automatizar completamente o processo.

No caso de duas retas ou mais, com mesmo ρ e θ, sendo uma ao lado da outra, o sistema detecta como se fosse apenas uma. Uma alternativa seria estudar como um algoritmo de identificação de lacunas resolveria essa questão.

A detecção de parábolas desenvolvida nesta dissertação pode vir a ser um método mais simples de calcular curvatura [COS2000].

Existem outras maneiras de identificar formas arbitrárias, como por exemplo usando a transformada de Hough generalizada [BAL,1981], [AGU,2000], [AGU,2002], ou a adaptativa [ECA,2004]. A identificação de um molde através da transformada de Hough generalizada, melhoraria a automatização do processo, eliminando as fases de identificação de cada forma primitiva.

Referências

[AGU1996]-A. Aguado, M. Montiel, M. Nixon. *On Using Directional Information for Parameter Space Decomposition in Ellipse Detection*. Pattern Recognition, Vol. 29, No. 3, pp. 369 381, 1996.

[AGU2000]-A. Aguado, M. Montiel, M. Nixon. *Bias Error Analysis of the Generalised Hough Transform*. Journal of Mathematical Imaging and Vision, No. 12, pp. 25-42, 2000.

[AGU2002]-A. Aguado, M. Montiel, M. Nixon. *Invariant Characterization of the Hough Transform for Pose Estimation of Arbitrary Shapes*. Pattern Recognition, Vol. 35, pp. 1083-1097, 2002.

[AZE2003]-E. Azevedo, A. Conci. *Computação Gráfica: teoria e prática*, Campus; 2003.

[BAL1981]-D. H, Ballard. *Generalising the Hough Transform to Detect Arbitrary Shapes*. Pattern recognition, Vol. 13, pp.111-122,1981.

[COS2000]-L. F, Costa. R.M.Cesar Junior. *Shape Analysis and Classification:- Theory and Practice.* ISBN: 0849334934, CRC Press , 2000.

[CHA2004]-C. Chau, W. Siu.*Generalised Hough Transform using Regions with Homogeneous Color*. International Journal of Computer Vision, Vol. 59, pp. 183-199, 2004.

[CHE2001]-T. Chen, K. Chung. *An Efficient Randomized Algorithm for Detecting Circles*. Computer Vision and Image Understanding, Vol. 83, pp. 172-191, 2001.

[DUD1972]-R.Duda, P. Hart. *Use of the Hough transformation to detect lines and curves in the pictures*. Communication Association Computing Machine, Vol. 15, pp. 11-15, Jan, 1972.

[ECA2004]-O. Ecabert, J. Thiran. *Adaptive Hough Transform for the Detection of Natural Shapes under Weak Affine Tranformations*. Pattern Recognition, Vol. 25, pp. 1411-1419, 2004.

[GON1992]-R. Gonzales. *Digital Image Processing*. Addinson-Wesley, 1992.

[GOR1976]-F. O´Gorman, M. Clowes. *Finding picture edges through collinearity os feature points*. IEEE Transaction Computer, C25(4), 449-54, 1976.

[HOU1962]-P. Hough. *Method and means for recognizing complex patterns.* U.S Patent 3 069 654, Dec, 1962.

[KAP1973]-W. Kaplan, D. Lewis. *Cálculo e Álgebra Linear 2*. Livros Técnicos e Científicos. S.A., 1973.

[KIM1975]-C. Kimme, D. Ballard, J. Sklansky. *Circles by an Array of Accumulators*. Comunications of the ACM, Vol.18, pp 120-122, Feb, 1975.

[LEI1999]-Y. Lei, K. Wong. *Ellipse detection based on symmetry*. Pattern Recognition,Vol.20, pp 41-47, 1999.

[MAC2004]-M. Macedo, A. Conci. *Um Sistema para Controle de Qualidade de Medicamentos.* Anais IX Congresso Regional de Informática e Telecomunicações, 2004.

[MEN2003]-V. B. Mendes. *Medição de Identações de Dureza Brinell e Vickers por meio de Técnicas de Visão Computacional*. Dissertação (Engenharia Mecânica). Universidade Federal Fluminense, 2003.

[OLI2003]-D. Oliveira. *Uma Proposta de Metodologia para Vetorização Controlada de Moldes para a Indústria do Vestuário.* Dissertação de Mestrado (Computação). Universidade Federal Fluminense, 2003

[OLS2001]-C. Olson. *A General Method for Geometric Feature Matching and Model Extraction.* International Journal of Computer Vision. Vol.45, pp.39–54, 2001.

[PAR1997]-J. Parker. *Algoritms for image Processing and Computer Vision*. John Wiley & Sons, Toronto, 1997.

[PRA2001]-W. Pratt. *Digital Image Processing* . Third Edition. John Wiley & Sons, 2001.

[SCH1978]-B. Schachter, A. Rosenfeld. *Some New Methods of Detecting Step Edges in Digital Pictures*. Comunications of ACM, Vol. 21, No. 21, 1978.

[SKL1978]-J. Sklansky. *On the Hough Technique for Curve Detection*. IEEE Transaction Computer, No. 10, 1978.

[TSU1978]-S. Tsuji, F. Matsumoto. *On Detecting of Ellipses by a Modified Hough Transformation*. IEEE Transaction Computer, Vol. 27, No. 8, 1978.

[WAT1998]-A. Watt,F. Policarpo. *The compute Image.* Addison-Wesley, 1998.

[XIE2002]-Y. Xie. *A New Efficient ellipse detection Method*. IEEE Transaction Computer, 2002.

[XU1993]-L. Xu, E. Oja . *Randomized Hough Transform (RHT): Basic Mechanisms, Algorithms, and Computational Complexities*. CVGIP: Image Understanding. Vol. 57, No. 2, 1993.