UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

CAIO CESAR FATTORI

Tratamento de imagens baseado no Método de Tikhonov

São Paulo
2007

Caio Cesar Fattori

Tratamento de imagens baseado no Método de Tikhonov

Monografia apresentada à
Escola Politécnica da Universidade de
São Paulo para obtenção do título de
bacharel em Engenharia.

Orientador: Alexandre Kawano

São Paulo
2007

FICHA CATALOGRÁFICA

Fattori,Caio Cesar
Tratamentos de Imagens baseados nos métodos de Tikhonov / C.C. Fattori. – São Paulo, 2007.
80p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Processamento de Imagens 2. Imagem digital (Tratamento) I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

Resumo

No caso de imagens obtidas analogicamente, o objetivo de seu tratamento é conservar seus dados, uma vez que ela carrega uma grande quantidade de informações, que podem ser perdidas com o passar do tempo. Uma forma de verificar se aqueles dados ainda são verossímeis é encontrar os contornos, ou bordas, da imagem. Mais modernamente, tanto no caso de imagens digitais como analógicas, o objetivo é o de recuperar informações difíceis de serem percebidas a olho nu. No processo de tratamento de imagens, mantemos armazenadas informações como forma de objetos contidos e, com uma simples reconstituição de cores, poderíamos regenerar a imagem.

O processo acima descrito é o processo de Detecção de Bordas (Edge Detection). Este processo pode ser usado para identificar objetos contidos no espaço. Um robô que pode traçar os contornos de um objeto e tiver um banco de dados de formas de objetos pode identificá-los no espaço. Essa detecção se baseia em ver a diferença entre as cores. Como as imagens podem conter distúrbios de um pixel para outro, é necessário efetuar uma filtragem antes de se verificar a variação de cores. O trabalho mostra a utilização do método de Tikhonov como filtro e as diferenças de não utilizar o método.

Palavras chave: Detecção de Bordas, Método de Sobel, Método de Laplace em Imagens, Método de Tikhonov, Detector de Canny.

Abstract

In the case of an image obtained analogically, the purpose of its treatment is, as an image carries a lot of information that can be lost with the passage of time, to keep that information preserved. One way to verify if these data still are true is find the contours or edges of that image. Presently, in the cases of digital and analogic images, the objectives are to recover information that could be hard to detect with naked eyes. In the process of image treatment, the information is stored as a form of objects contained within and, with a simple reconstitution of colors, we could regenerate the image. The process described above is the process of Edge Detection.

This process can be used to identify objects in the space. A robot that can trace the contours of an object and has a database of objects can identify them in the space. This detection is based in watching the difference between the colors. As the images may contain disturbances for a pixel to another, it is necessary to make a filtering before verify the variation of colors. The work shows the use of the method of Tikhonov as filter and the differences of not using the method.

Keywords: Edge Detection, Sobel Method, Laplacian Method to Images, Thikonov Method, Canny Detector.

Sumário

Capítulo 1

Introdução

Edge detection é usado para reconhecimento de orgãos humanos pela medicina. Este é um passo importante no pré-tratamento médico de imagem 3D de segmentação e reconstrução [15]. Com a identificação dos orgãos humanos é possível verificar a posição para um corte.

Outras aplicações comerciais são Sensoreamento Remoto de recursos terrestres e militares, com o método se torna possível identificar bases militares em fotos por satélite, áreas de vegetações e relevos. Usado em Aplicações Geofísicas, como Análise Sísmica, Anomalia Magnética, GPR e SAR [10].

É usado também em Biomedição de digitais, rostos e iris, com o processo pode se retirar distúrbios das imagens dos identificadores e selecionar contornos específicos que seriam registrados em banco de dados e seriam identificados mesmo com diferentes equipamentos ou condições dos identificadores das pessoas. Também usado para Segurança, Vigilância e Inspeção de Bagagens, Visão e Inspeção de máquinas [10], pelo contorno dos objetos em fotos de radiografia o sistema de inspeção pode identificar por comparação com contornos semelhantes em banco de dados.

Esse trabalho foi composto do estudo de Detecção de Bordas em diferentes métodos e posterior comparação de resultados. A Detecção de Bordas é um tratamento dado a uma imagem que compara entre pontos vizinhos se há uma mudança de elemento de cor. Em imagem de computador essa mudança de cor será representada por uma mudança de intensidade de pixels vizinhos.

Os métodos implementados no trabalho se dividem entre os que utilizam a regularização de Tikhonov e não utilizam. A regularização de Tikhonov atua como um filtro com sensibilidade escolhida para o tratamento.

Após realizar um processo de regularização da imagem, os dados filtrados são processados na busca de Bordas. Para essa busca, as funções utilizadas foram as SPlines Cúbicas Unidimensionais e Bidimensionas.

Os métodos que não utilizam a regularização de Tikhonov foram os métodos de Sobel e Laplace. Estes métodos são mais clássicos no tratamento

de imagem e usam uma operação de máscara aplicada a imagem para buscar as bordas.

Para controlar a qualidade de um Detector de Bordas, os métodos foram confrontados com o princípio de Canny, que cita 3 critérios para um detector de bordas confiável. Com a aplicação desses critérios o tratamento da imagem passa a se basear na busca de um máximo da primeira derivada da função de imagem.

Capítulo 2

Detecção de Bordas

2.1 Critério de Canny

O princípio de Canny foi desenvolvido, principalmente, para resolver problemas de detecção de bordas de imagens e indicar o comportamento de um detector para este ser considerado eficiente.

Segundo Canny, para averiguar a eficiência de um detector de bordas, este deve atender à 3 critérios básicos.

O primeiro é a Taxa de Erro, que consiste na maximização da relação sinal/ruído. Segundo Canny, quanto maior for essa relação, mais fácil será de encontrar a borda verdadeira. Isso ocorre porque o sinal se destaca na onda gerada e minimiza o efeito que o ruído pode causar, já que o ruído é uma elevação o redução no sinal e pode ser interpretado como uma borda. Um sinal característico pode ser visto na Figura 2.1.

O segundo critério trata dos pontos de borda. De acordo com este, os pontos de borda devem estar bem localizados, minimizando a distância entre este e a borda real. Este critério trata principalmente em deslocamento de posição de uma borda, pode ser interpretado como um atraso na leitura do sinal em relação a imagem real.

O terceiro critério é o da Resposta Múltipla, que diz que só deve haver uma borda onde existe uma única borda verdadeira. Neste caso, devemos reduzir ao máximo a espessura da borda da imagem, pois o verdadeiro contorno de uma imagem é uma linha de espessura de medida nula.

Pode-se concluir por esses 3 critérios que a maior probabilidade da existência de uma borda é num ponto de máximo da função derivada, pois é uma região média no crescimento da derivada da mesma (que minimiza a distância dos pontos possíveis de borda real) e neste ponto, o sinal recebido é o de maior variação (assim o sinal é bem superior um dado ruído controlado).

Os algoritmos então devem usar ferramentas matemáticas para encontrar os pontos de máxima variação da derivada do sinal de imagem, conforme descrito em [1], [11] e [2].

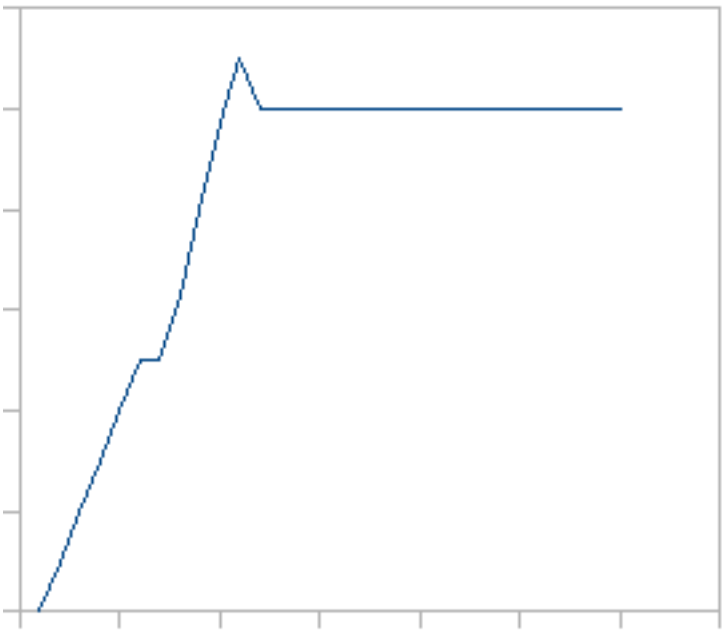

Figura 2.1: Exemplo da relação de sinal e ruído

2.2 Métodos Clássicos

Os métodos clássicos estudados foram o Método de Sobel e Laplace, descritos em [5], [8], [12] e [9].

2.2.1 Laplace

O método de Laplace se baseia na localização de um ponto de inflexão na função de imagem. Este ponto tem significado matemático de segunda derivada nula e indica um ponto de máximo ou de mínimo na derivada da função de imagem. Portanto esse método trabalha com a segunda derivada da função imagem.

O método usa a definição básica de derivadas de funções para encontrar os valores das derivadas de segunda ordem, porém adaptado a casos de dados discretos.

Pela definição, para uma função de classe C^1, temos
$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$

Aproximando para dados discretos com espaçamento dos pontos unitários:

$$f'(x) \cong f(x + 0.5) - f(x - 0.5)$$

Seguindo o mesmo raciocínio acima exposto, podemos encontrar o valor da segunda derivada como:

$$f''(x) \cong f'(x + 0.5) - f'(x - 0.5)$$

juntando as equações

$$f''(x) \cong f(x + 1) - f(x) - (f(x) - f(x - 1))$$

e, portanto

$$f''(x) = f(x + 1) - 2 * f(x) + f(x - 1)$$

Então podemos verificar a existência de um vetor no formato de uma máscara que obtêm o valor da segunda derivada de uma função, a partir de uma tabela de dados, desde que estes estejam igualmente espaçados com valor unitário.

O vetor é:

$$f''(x) = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} * \begin{bmatrix} f(x+1) \\ f(x) \\ f(x-1) \end{bmatrix}$$

Para analisarmos essa matriz independentemente da direção que possa estar o máximo de variação, expandimos o sistema nas duas direções, formamos a matriz:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Este elemento acima não se trata de uma matriz, mas sim uma máscara que deve ser utilizada, conforme descritana figura 2.2, obtida em [7], da seguinte forma:

$$b_{i,j} = \begin{array}{c} a_{i-1,j-1} * c_{i-1,j-1} + a_{i-1,j} * c_{i-1,j} + a_{i-1,j+1} * c_{i-1,j+1} + \\ a_{i,j-1} * c_{i,j-1} + a_{i,j} * c_{i,j} + a_{i,j+1} * c_{i,j+1} + \\ a_{i+1,j-1} * c_{i+1,j-1} + a_{i+1,j} * c_{i+1,j} + a_{i+1,j+1} * c_{i+1,j+1} \end{array}$$

Onde $a_{i,j}$ representa os elementos da matriz de cores, $c_{i,j}$ representa a máscara a ser aplicada e bi, j representa um elemento da resposta da aplicação da máscara aos dados.

Neste caso a matriz é capaz de verificar precisamente variações em duas direções principais. Para melhorar a precisão do método, podemos também considerar as derivadas mistas.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

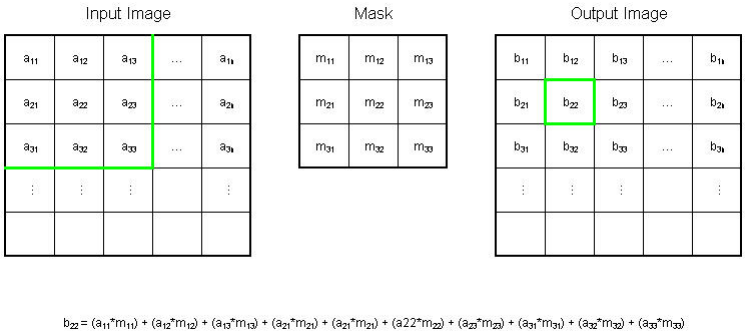

Figura 2.2: Forma de utilização da máscara

Essa alteração provoca uma mudança significativa na operação na imagem, ela passa a ser invariante às direções.

Outra forma de aprimorar o resultado é expandir o sistema para a segunda ordem, que faz com que atinjamos mais pontos adjacentes.

A nova Tabela será assim formada

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -24 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Ao se passar essa máscara por todo conteúdo da imagem, temos uma tabela que oferece a segunda derivada da função de imagem.

A segunda derivada da função é importante por nela estar armazenada a informação do comportamento da primeira derivada da função de imagem. As considerações disso são descritas no Princípio de Canny.

Então, quando ocorre uma mudança de sinal na segunda derivada da função de imagem, quer dizer que esta passou por um máximo ou um mínimo da primeira derivada e infere-se que há uma borda neste ponto.

Este método é invariante a rotação da imagem por conservar suas propriedades em todas as direções iguais, ou seja, as influências de todos os pontos adjacentes ao ponto central são iguais, como pode ser visto na tabela acima.

Um problema que esse método cria é a sensibilidade ao ruído, isso ocorre porque o ruído cria uma oscilação falsa no sistema e um ponto de máximo local. Portanto uma imagem que utiliza o tratamento de Laplace pode apresentar pontos de borda não reais.

O algoritmo deste método pode ser visto no ANEXO A.

2.2.2 Método de Sobel

O método de Sobel trabalha com a primeira derivada da função de imagem e com isso ele busca o ponto de pico ou de fundo da função derivada. O método garante que neste ponto há uma maior probabilidade de se encontrar uma borda de imagem.

Por meio do método a função deve procurar o ponto de máximo da derivada em cada curva.

Com o método podemos definir quando uma derivada pode ou não ser considerada como uma borda, como será demonstrado posteriormente.

Da definição de derivadas de funções de classe C^1, $2 * f'(x) \cong f(x+1) - f(x-1)$ que é a derivação numérica em torno do ponto central.

Esta definição é interessante por ser capaz de analisar a média das derivadas anterior e posterior ao ponto considerado, pois
$\frac{df(x)}{dx} = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ e $\frac{df(x)}{dx} = \lim_{h\to 0} \frac{f(x)-f(x-h)}{h}$ que é uma variação relevante quando tratamos de dados discretos.

Então a equação $2f'(x) = f(x+1) - f(x-1)$ pode ser obtida somando as duas definições básicas acima para espaçamento h unitário.

Com isso, montamos um vetor de convolução, em uma direção, da forma

$$f'(x) = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} f(x-h) \\ f(x) \\ f(x+h) \end{bmatrix}$$

Para adaptar os dados a uma expansão bi-dimensional, usamos um conceito estatístico de peso por influência. Uma informação cuja alteração cause maior influência sobre um resultado do que outra informação deve possuir um peso maior na elaboração desse resultado. Para o estudo dos pontos do gráfico de imagem, as linhas paralelas a estudada tem menor influência no resultado final do que a própria linha deste. Usando então a distribuição Gaussiana binomial sobre o vetor que é multiplicado aos valores da função f(x), temos que o sistema estará assim formado:

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} = & \quad 1 * (-1,0,1) * \begin{bmatrix} f(x-1,y-1) \\ f(x,y-1) \\ f(x+1,y-1) \end{bmatrix} + \\ & +2 * (-1,0,1) * \begin{bmatrix} f(x-1,y) \\ f(x,y) \\ f(x+1,y) \end{bmatrix} + \\ & +1 * (-1,0,1) * \begin{bmatrix} f(x-1,y+1) \\ f(x,y+1) \\ f(x+1,y+1) \end{bmatrix} \end{aligned}$$

Considerando que x varia na direção horizontal e y na direção vertical, a máscara a ser aplicada na direção x será:

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Generalizando, temos em outras direções as matrizes:

$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} G_{45^o} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} G_{135^o} = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$

Essas máscaras são aplicadas da mesma forma desenvolvida no método de Laplace.

Segundo o Princípio de Canny, a derivada de maior valor estará, provavelmente, num ponto de borda. Este ponto de máximo da derivada pode ser encontrado comparando-se dados adjacentes e verificar quando são crescentes ou decrescentes.

O método de Sobel opera da mesma forma que o método de Laplace, ou seja, através de uma matriz que é multiplicada aos dados da tabela de imagem como em Figura 2.2.

A diferença desse método é que a análise é feita em duas dimensões e é feita então uma soma vetorial da mesma. Portanto é um método de análise por gradiente de função.

Após multiplicados pelos pontos da tabela de dados, os vetores formados devem ser somados e seu módulo dará o valor do gradiente no ponto em que se fez a análise. Através da escolha de uma sensibilidade para a derivada, podemos determinar se um ponto possui derivada para conter uma borda ou para conter um ruído.

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} = & \begin{bmatrix} -1 & -2 & -1 \end{bmatrix} * \begin{bmatrix} f(x-1,y-1) \\ f(x,y-1) \\ f(x+1,y-1) \end{bmatrix} + \\ & \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} f(x-1,y) \\ f(x,y) \\ f(x+1,y) \end{bmatrix} + \\ & \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} f(x-1,y+1) \\ f(x,y+1) \\ f(x+1,y+1) \end{bmatrix} \end{aligned}$$

O mesmo deve ser feito usando o G_y e ao final deve ser feita a soma vetorial dos termos.

$$G = \sqrt{\frac{\partial f(x,y)}{\partial x}^2 + \frac{\partial f(x,y)}{\partial y}^2} \geq \beta \tag{2.1}$$

Onde β é o valor limite, escolhido pelo usuário, para definir a existência de uma borda, declarando como valor em módulo da derivada da função de imagem.

O algoritmo deste método pode ser visto no ANEXO B.

2.3 Regularização de Tikhonov

Este método é usado para regularizar os dados que se deseja trabalhar. Utilizando uma constante definida pelo operador, um fator regulador, pode-se alterar consideravelmente o resultado esperado do sistema.

Considerando o caso de Detecção de Bordas, a função aproximadora deve ter sua segunda derivada contínua, para ser possível encontrar os pontos de máximo e mínimo da primeira derivada, pontos que, segundo Canny, possuem bordas de imagem.

A condição gerada é que nos contornos não existem distúrbios. Isso pode ser usado uma vez que a função precisa ser ancorada em pontos de acordo com a sua dimensão. Por exemplo, são necessários dois pontos para definir um segmento de reta.

Conforme demonstrado em [13], a equação no caso unidimensional é:

$$f_{i+1}'''(x_i) - f_i'''(x_i) = \frac{1}{\alpha}\frac{x_{i+1} + x_{i-1}}{2}(\tilde{y_i} - f_{i+1}(x_i)) \tag{2.2}$$

para $i = 1, 2, 3, \ldots, n-1$, onde $\tilde{y_i}$ são os valores de entrada da tabela de dados.

Estendendo para o caso Bidimensional:

$$\frac{\partial^3 f_{i+1,j}(x_i, y_j)}{\partial x^3} - \frac{\partial^3 f_{i,j}(x_i, y_j)}{\partial x^3} = \frac{1}{\alpha}\frac{x_{i+1} + x_{i-1}}{2}(\tilde{z_{i,j}} - f_{i+1,j}(x_i, y_j)) \tag{2.3}$$

ou então:

$$\frac{\partial^3 f_{i,j+1}(x_i, y_j)}{\partial y^3} - \frac{\partial^3 f_{i,j}(x_i, y_j)}{\partial y^3} = \frac{1}{\alpha}\frac{y_{j+1} + y_{j-1}}{2}(\tilde{z_{i,j}} - f_{i,j+1}(x_i, y_j)) \tag{2.4}$$

para $i = 1, 2, 3, \ldots, n-1$ e $j = 1, 2, 3, \ldots, m-1$, onde $\tilde{z_{i,j}}$ são os valores de entrada da tabela de dados.

2.3.1 SPlines Cúbicas Unidimensionais

A transformação de sistemas discretos para contínuos pode utilizar como função adaptadora as SPlines cúbicas, para o caso unidimensional. Essas funções vão ser usadas por serem funções bem comportadas, ou seja, se adaptam facilmente as curvas.

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i \tag{2.5}$$

As funções de SPlines se adaptam às funções de imagem com as condições de contorno:

$$f_1(x_0) = \tilde{y_0} \tag{2.6}$$

$$f_n(x_n) = \tilde{y_n} \tag{2.7}$$

Para representar o surgimento do uma borda no contorno da imagem, temos que garantir que há um ponto de máximo ou de mínimo nos pontos extremos, portanto, devemos considerar a segunda derivada nula nestes pontos.

$$f_1''(x_0) = f_n''(x_n) = 0 \tag{2.8}$$

As condições de continuidade usam as condições abaixo e a condições de Tikhonov (2.2):

$$f_i(x_i) = f_{i+1}(x_i) \tag{2.9}$$

.

$$f_i'(x_i) = f_{i+1}'(x_i) \tag{2.10}$$

$$f_i''(x_i) = f_{i+1}''(x_i) \tag{2.11}$$

.

$$f_{i+1}'''(x_i) - f_i'''(x_i) = \frac{1}{\alpha}\frac{x_{i+1} + x_{i-1}}{2}(\tilde{y_i} - f_{i+1}(x_i)) \tag{2.12}$$

Essas condições são impostas para tornar as funções de classe C_2.

Uma simplificação usada para as SPlines foi a conversão de intervalos: para $x \in [x_{i-1}, x_i]$ para $i = 0, 2, \ldots, n$

$$x^* = \frac{x - x_{i-1}}{x_i - x_{i-1}} \tag{2.13}$$

Nestas condições, mudamos o intervalo de x de $[x_{i-1}, x_i]$ quaisquer, para o intervalo $[0, 1]$. Esta mudança de intervalo facilita os cálculos nos pontos do gráfico e uniformiza os elementos da matriz que calcula os coeficientes das SPlines.

Então as equações (2.6), (2.7), (2.8), (2.9), (2.10), (2.11) vão ficar na forma:

$$f_1(0) = \tilde{y_0} \tag{2.14}$$

$$f_n(1) = \tilde{y_n} \tag{2.15}$$

$$f_1''(0) = f_n''(1) = 0 \tag{2.16}$$

$$f_i(1) = f_{i+1}(0) \tag{2.17}$$

.

$$f_i'(1) = f_{i+1}'(0) \tag{2.18}$$

$$f_i''(1) = f_{i+1}''(0) \tag{2.19}$$

.

$$f_i'''(1) - f_{i+1}'''(0) = \frac{x_{i+1} - x_{i-1}}{2\alpha}(\tilde{y_i} - f_{i+1}(0)) \tag{2.20}$$

.

Calculando as condições de contorno (2.14), (2.15) e (2.16) respectivamente:

$$\begin{aligned} d_1 &= \tilde{y_0} \\ a_n + b_n + c_n + d_n &= \tilde{y_n} \\ 2b_1 = 6a_n + 2b_n &= 0 \end{aligned}$$

Por (2.17), (2.18) e (2.19) temos respectivamente:

$$\begin{aligned} a_i + b_i + c_i + d_i &= d_{i+1} \\ 3a_i + 2b_i + c_i &= c_{i+1} \\ 6a_i + 2b_i &= 2b_{i+1} \end{aligned}$$

E por (2.2) chegamos a:

$$6a_{i+1} - 6a_i = \frac{x_{i+1} - x_{i-1}}{2\alpha}(\tilde{y_i} - d_{i+1})$$

Juntando as condições de contorno, chegamos ao sistema matricial da forma:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & -1 & \dots \\ 3 & 2 & 1 & 0 & 0 & 0 & -1 & 0 & \dots \\ 6 & 2 & 0 & 0 & 0 & -2 & 0 & 0 & \dots \\ -6 & 0 & 0 & 0 & 6 & 0 & 0 & \gamma_1 & \dots \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \dots \\ 0 & 0 & 0 & 0 & 3 & 2 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 6 & 2 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & -6 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{Bmatrix} a_1 \\ b_1 \\ c_1 \\ d1 \\ \vdots \end{Bmatrix} = \begin{Bmatrix} y_0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \gamma_1 y_1 \\ 0 \\ 0 \\ 0 \\ \gamma_2 y_2 \\ 0 \\ \vdots \end{Bmatrix}$$

em que $\gamma_i = \frac{x_{i+1} - x_{i-1}}{2\alpha}$.

Esta matriz repete os 4 últimos elementos, mas com 4 colunas deslocadas para a direita.

Duas funções foram usadas como teste para as SPlines, a função triângulo, representado na Figura 3.8, e a função triângulo com distúrbios, representado na Figura 3.6. Em ambos os casos a função mostrou-se bem adaptada de acordo com os coeficientes de distúrbio.

A função triângulo pode ser definida como: $f(x) = \begin{Bmatrix} x+1 & sex < 0 \\ 1-x & sex \geq 0 \end{Bmatrix}$.

Este estudo deve ser feito nas duas direções para que não haja perda de informações em nenhuma direção e o resultado das derivadas nas duas direções deve ser somado vetorialmente em cada ponto.

O código gerado para este tratamento pode ser encontrado no APÊNDICE A

2.3.2 SPlines Cúbicas Bidimensionais

Há algumas semelhanças entre as teorias de SPlines Cúbicas e SPlines Bidimensionais, ambas estudam uma tabela de dados através da obtenção de uma função aproximadora e ambas usam funções definidas para o fazer. A diferença entre elas é que no caso das SPlines cúbicas, esse estudo é feito em cada direção separadamente, podendo provocar erros por não possuir todas as informações adjacentes necessárias. As SPlines Bidimensionais utilizam elementos de área, enquanto as SPlines Unidimensionais utilizam elementos de linha.

Para possuir uma boa precisão, a função aproximadora estudada neste método foi o polinômio de terceiro grau misto na forma:

$$f(x,y) = a_{3,3}x^3y^3 + a_{3,2}x^3y^2 + a_{3,1}x^3y + a_{3,0}x^3 +$$

$$a_{2,3}x^2y^3 + a_{2,2}x^2y^2 + a_{2,1}x^2y + a_{2,0}x^2 + a_{1,3}xy^3 + a_{1,2}xy^2 + a_{1,1}xy + a_{1,0}x + a_{0,3}y^3 + a_{0,2}y^2 + a_{0,1}y + a_{0,0}$$

Neste caso temos para cada elemento de área da malha de dados 16 incógnitas a serem determinadas. A malha de dados será dividida em quadrados de dimensões unitárias.

As condições de contorno são semelhantes as do caso unidimensional, mas são adaptadas para o caso bidimensional. As condições de contorno se aplicam a todos os pontos da borda na forma:

$$f(x_i, y_j) = z_{i,j} \tag{2.21}$$

onde $i = 0$ ou n e $j = 0$ ou m.

A outra condição de contorno é:

$$\begin{aligned} \frac{\partial^2 f_{1,j}(x_0, y)}{\partial x^2} &= 0 \\ \frac{\partial^2 f_{i,1}(x, y_0)}{\partial y^2} &= 0 \\ \frac{\partial^2 f_{n,j}(x_n, y)}{\partial x^2} &= 0 \\ \frac{\partial^2 f_{i,m}(x, y_m)}{\partial y^2} &= 0 \end{aligned} \tag{2.22}$$

Neste caso, em toda uma linha do elemento de área a segunda derivada da função deverá ser nula. Isso ocorre, pois é uma necessidade do contorno da imagem que todos os possíveis pontos intermediários tenham a mesma condição de contorno.

As condições de continuidade também são semelhantes as do caso unidimensional. Deve se ressaltar que existem condições agora em duas direções. Assim como o caso da condição de contorno de segunda derivada nula, estas condições devem ser aplicadas a todos os pontos que limitam o elemento de área. Então:

$$\begin{aligned} f_{i,j}(x_i, y) &= f_{i+1,j}(x_i, y) \\ f_{i,j}(x, y_j) &= f_{i,j+1}(x, y_j) \end{aligned} \tag{2.23}$$

$$\begin{aligned} \frac{\partial f_{i,j}(x_i, y)}{\partial x} &= \frac{\partial f_{i+1,j}(x_i, y)}{\partial x} \\ \frac{\partial f_{i,j}(x, y_j)}{\partial x} &= \frac{\partial f_{i,j+1}(x, y_j)}{\partial x} \end{aligned} \tag{2.24}$$

$$\frac{\partial f_{i,j}(x_i, y)}{\partial y} = \frac{\partial f_{i+1,j}(x_i, y)}{\partial y}$$
$$\frac{\partial f_{i,j}(x, y_j)}{\partial y} = \frac{\partial f_{i,j+1}(x, y_j)}{\partial y} \tag{2.25}$$

$$\frac{\partial^2 f_{i,j}(x_i, y)}{\partial x^2} = \frac{\partial^2 f_{i+1,j}(x_i, y)}{\partial x^2}$$
$$\frac{\partial^2 f_{i,j}(x, y_i)}{\partial x^2} = \frac{\partial^2 f_{i,j+1}(x, y_i)}{\partial x^2} \tag{2.26}$$

$$\frac{\partial^2 f_{i,j}(x_i, y)}{\partial y^2} = \frac{\partial^2 f_{i+1,j}(x_i, y)}{\partial y^2}$$
$$\frac{\partial^2 f_{i,j}(x, y_i)}{\partial y^2} = \frac{\partial^2 f_{i,j+1}(x, y_i)}{\partial y^2} \tag{2.27}$$

$$\frac{\partial^2 f_{i,j}(x_i, y)}{\partial x \partial y} = \frac{\partial^2 f_{i+1,j}(x_i, y)}{\partial x \partial y}$$
$$\frac{\partial^2 f_{i,j}(x, y_i)}{\partial x \partial y} = \frac{\partial^2 f_{i,j+1}(x, y_i)}{\partial x \partial y} \tag{2.28}$$

para $i = 1, 2, 3, \ldots, n$ e $j = 1, 2, 3, \ldots, m$.

Considerando que as condições de contorno e de continuidade devem ser válidas em todo segmento de reta, os termos constantes devem ter valores semelhantes para variáveis semelhantes, como no caso:

$$\begin{aligned} a_3 z^3 + a_2 z^2 + a_1 z + a_0 &= b_3 z^3 + b_2 z^2 + b_1 z + b_0 \\ a_3 &= b_3 \\ a_2 &= b_2 \\ a_1 &= b_1 \\ a_0 &= b_0 \end{aligned}$$

Para facilitar os cálculos, a mesma mudança de base feita no caso unidimensional será feita agora, mas considerando que as variáveis agora são x e y.

$$x^* = \frac{x - x_{i-1}}{x_i - x_{i-1}}$$
$$y^* = \frac{y - y_{j-1}}{y_j - y_{j-1}}$$

Essa transformação de base faz com que o domínio de cada elemento de área seja um quadrado de coordenadas $[0,0]$; $[0,1]$; $[1,0]$ e $[1,1]$.

Pela equação (2.21) com a mudança de base acima proposta, podemos concluir que:

$$\begin{aligned} f(0,0) &= a_{0,0} = \tilde{z_{0,0}} \\ f(0,1) &= a_{0,3} + a_{0,2} + a_{0,1} + a_{0,0} = \tilde{z_{0,1}} \\ f(1,0) &= a_{3,0} + a_{2,0} + a_{1,0} + a_{0,0} = \tilde{z_{1,0}} \\ f(1,1) = & \begin{array}{c} a_{3,3} + a_{3,2} + a_{3,1} + a_{3,0} + a_{2,3} + a_{2,2} + a_{2,1} + a_{2,0} + \\ a_{1,3} + a_{1,2} + a_{1,1} + a_{1,0} + a_{0,3} + a_{0,2} + a_{0,1} + a_{0,0} = \tilde{z_{1,1}} \end{array} \end{aligned} \tag{2.29}$$

Já pelas equações presentes em (2.22) com a mudança de base, podemos concluir que:

$$\begin{aligned} \frac{\partial^2 f_{1,j}(0,y)}{\partial x^2} &= 2a_{2,3}y^3 + 2a_{2,2}y^2 + 2a_{2,1}y + 2a_{2,0} = 0 \\ \frac{\partial^2 f_{i,1}(x,0)}{\partial y^2} &= 2a_{3,2}x^3 + 2a_{2,2}x^2 + 2a_{1,2}x + 2a_{0,2} = 0 \\ \frac{\partial^2 f_{n,j}(1,y)}{\partial x^2} &= (6a_{3,3}+2a_{2,3})y^3 + (6a_{3,2}+2a_{2,2})y^2 + (6a_{3,1}+2a_{2,1})y + (6a_{3,0}+2a_{2,0}) = 0 \\ \frac{\partial^2 f_{i,m}(x,1)}{\partial y^2} &= (6a_{3,3}+2a_{3,2})x^3 + (6a_{2,3}+2a_{2,2})x^2 + (6a_{1,3}+2a_{1,2})x + (6a_{0,3}+2a_{0,2}) = 0 \end{aligned} \tag{2.30}$$

Com isso formamos as 16 equações abaixo:

$$\begin{aligned} 2a_{2,3} &= 0 \\ 2a_{2,2} &= 0 \\ 2a_{2,1} &= 0 \\ 2a_{2,0} &= 0 \\ 2a_{3,2} &= 0 \\ 2a_{2,2} &= 0 \\ 2a_{1,2} &= 0 \\ 2a_{0,2} &= 0 \\ 6a3,3 + 2a_{2,3} &= 0 \\ 6a3,2 + 2a_{2,2} &= 0 \\ 6a3,1 + 2a_{2,1} &= 0 \\ 6a3,0 + 2a_{2,0} &= 0 \\ 6a3,3 + 2a_{3,2} &= 0 \\ 6a2,3 + 2a_{2,2} &= 0 \\ 6a1,3 + 2a_{1,2} &= 0 \\ 6a0,3 + 2a_{0,2} &= 0 \end{aligned} \tag{2.31}$$

No caso das condições de contorno (2.23) com a mesma mudança de base:

$$f_{u,v}(1,y) = f_{u+1,v}(0,y) =$$
$$= \overbrace{\sum_{i=0}^{3} a_{i,3}y^3 + \sum_{i=0}^{3} a_{i,2}y^2 + \sum_{i=0}^{3} a_{i,1}y + \sum_{i=0}^{3} a_{i,0}}^{u,v} = \overbrace{a_{0,3}y^3+a_{0,2}y^2+a_{0,1}y+a_{0,0}}^{u+1,v}$$
$$f_{u,v}(x,1) = f_{u,v+1}(x,0) =$$
$$= \overbrace{\sum_{j=0}^{3} a_{3,j}x^3 + \sum_{j=0}^{3} a_{2,j}x^2 + \sum_{j=0}^{3} a_{1,j}x + \sum_{j=0}^{3} a_{0,j}}^{u,v} = \overbrace{a_{3,0}x^3+a_{2,0}x^2+a_{1,0}x+a_{0,0}}^{u,v+1} \quad (2.32)$$

Com isso podemos formar as 8 equações abaixo:

$$\begin{aligned}
\overbrace{a_{3,3} + a_{2,3} + a_{1,3} + a_{0,3}}^{u,v} &= \overbrace{a_{0,3}}^{u+1,v} \\
\overbrace{a_{3,2} + a_{2,2} + a_{1,2} + a_{0,2}}^{u,v} &= \overbrace{a_{0,2}}^{u+1,v} \\
\overbrace{a_{3,1} + a_{2,1} + a_{1,1} + a_{0,1}}^{u,v} &= \overbrace{a_{0,1}}^{u+1,v} \\
\overbrace{a_{3,0} + a_{2,0} + a_{1,0} + a_{0,0}}^{u,v} &= \overbrace{a_{0,0}}^{u+1,v} \\
\overbrace{a_{3,3} + a_{3,2} + a_{3,1} + a_{3,0}}^{u,v} &= \overbrace{a_{3,0}}^{u,v+1} \\
\overbrace{a_{2,3} + a_{2,2} + a_{2,1} + a_{2,0}}^{u,v} &= \overbrace{a_{2,0}}^{u,v+1} \\
\overbrace{a_{1,3} + a_{1,2} + a_{1,1} + a_{1,0}}^{u,v} &= \overbrace{a_{1,0}}^{u,v+1} \\
\overbrace{a_{0,3} + a_{0,2} + a_{0,1} + a_{0,0}}^{u,v} &= \overbrace{a_{0,0}}^{u,v+1}
\end{aligned} \quad (2.33)$$

No caso das condições de contorno (2.24) com a mesma mudança de base:

$$\frac{\partial f_{u,v}(1,y)}{\partial x} = \frac{\partial f_{u+1,v}(0,y)}{\partial x} =$$
$$= \overbrace{\sum_{i=1}^{3} ia_{i,3}y^3 + \sum_{i=1}^{3} ia_{i,2}y^2 + \sum_{i=1}^{3} ia_{i,1}y + \sum_{i=1}^{3} ia_{i,0}}^{u,v} = \overbrace{a_{1,3}y^3+a_{1,2}y^2+a_{1,1}y+a_{1,0}}^{u+1,v}$$
$$\frac{\partial f_{u,v}(x,1)}{\partial x} = \frac{\partial f_{u,v+1}(x,0)}{\partial x} =$$

$$= \overbrace{\sum_{j=0}^{3} 3a_{3,j}x^2 + \sum_{j=0}^{3} 2a_{2,j}x + \sum_{j=0}^{3} a_{1,j}}^{u,v} = \overbrace{3a_{3,0}x^2+2a_{2,0}x+a_{1,0}}^{u,v+1} \quad (2.34)$$

Com isso podemos formar as 7 equações abaixo:

$$\begin{aligned}
\overbrace{3a_{3,3} + 2a_{2,3} + a_{1,3}}^{u,v} &= \overbrace{a_{1,3}}^{u+1,v} \\
\overbrace{3a_{3,2} + 2a_{2,2} + a_{1,2}}^{u,v} &= \overbrace{a_{1,2}}^{u+1,v} \\
\overbrace{3a_{3,1} + 2a_{2,1} + a_{1,1}}^{u,v} &= \overbrace{a_{1,1}}^{u+1,v} \\
\overbrace{3a_{3,0} + 2a_{2,0} + a_{1,0}}^{u,v} &= \overbrace{a_{1,0}}^{u+1,v} \\
\overbrace{3a_{3,3} + 3a_{3,2} + 3a_{3,1} + 3a_{3,0}}^{u,v} &= \overbrace{3a_{3,0}}^{u,v+1} \\
\overbrace{2a_{2,3} + 2a_{2,2} + 2a_{2,1} + 2a_{2,0}}^{u,v} &= \overbrace{2a_{2,0}}^{u,v+1} \\
\overbrace{a_{1,3} + a_{1,2} + a_{1,1} + a_{1,0}}^{u,v} &= \overbrace{a_{1,0}}^{u,v+1}
\end{aligned} \quad (2.35)$$

No caso das condições de contorno (2.25) com a mesma mudança de base:

$$\frac{\partial f_{u,v}(1,y)}{\partial y} = \frac{\partial f_{u+1,v}(0,y)}{\partial y} =$$

$$= \overbrace{\sum_{i=0}^{3} 3a_{i,3}y^2 + \sum_{i=0}^{3} 2a_{i,2}y + \sum_{i=0}^{3} a_{i,1}}^{u,v} = \overbrace{3a_{0,3}y^2+2a_{0,2}y+a_{0,1}}^{u+1,v}$$

$$\frac{\partial f_{u,v}(x,1)}{\partial y} = \frac{\partial f_{u,v+1}(x,0)}{\partial y} =$$

$$= \overbrace{\textstyle\sum_{j=1}^{3} ja_{3,j}x^3+\sum_{j=1}^{3} ja_{2,j}x^2+\sum_{j=1}^{3} ja_{1,j}x+\sum_{j=1}^{3} ja_{0,j}}^{u,v} = \overbrace{a_{3,1}x^3+a_{2,1}x^2+a_{1,1}x+a_{0,1}}^{u,v+1} \quad (2.36)$$

Com isso podemos formar as 7 equações abaixo:

$$\begin{aligned}
\overbrace{3a_{3,3} + 3a_{2,3} + 3a_{1,3} + 3a_{0,3}}^{u,v} &= \overbrace{3a_{0,3}}^{u+1,v} \\
\overbrace{2a_{3,2} + 2a_{2,2} + 2a_{1,2} + 2a_{0,2}}^{u,v} &= \overbrace{2a_{0,2}}^{u+1,v} \\
\overbrace{a_{3,1} + a_{2,1} + a_{1,1} + a_{0,1}}^{u,v} &= \overbrace{a_{0,1}}^{u+1,v}
\end{aligned}$$

$$\begin{aligned}
\overbrace{3a_{3,3}+2a_{3,2}+a_{3,1}}^{u,v} &= \overbrace{a_{3,1}}^{u,v+1} \\
\overbrace{3a_{2,3}+2a_{2,2}+a_{2,1}}^{u,v} &= \overbrace{a_{2,1}}^{u,v+1} \\
\overbrace{3a_{1,3}+2a_{1,2}+a_{1,1}}^{u,v} &= \overbrace{a_{1,1}}^{u,v+1} \\
\overbrace{3a_{0,3}+2a_{0,2}+a_{0,1}}^{u,v} &= \overbrace{a_{0,1}}^{u,v+1}
\end{aligned} \tag{2.37}$$

No caso das condições de contorno (2.26) com a mesma mudança de base:

$$\begin{aligned}
\frac{\partial^2 f_{u,v}(1,y)}{\partial x^2} &= \frac{\partial^2 f_{u+1,v}(0,y)}{\partial x^2} = \\
= \overbrace{\textstyle\sum_{i=2}^{3} i(i-1)a_{i,3}y^3 + \sum_{i=2}^{3} i(i-1)a_{i,2}y^2 + \sum_{i=2}^{3} i(i-1)a_{i,1}y + \sum_{i=2}^{3} i(i-1)a_{i,0}}^{u,v} &= \\
= \overbrace{2a_{1,3}y^3 + 2a_{1,2}y^2 + 2a_{1,1}y + 2a_{1,0}}^{u+1,v} & \\
\frac{\partial^2 f_{u,v}(x,1)}{\partial x^2} &= \frac{\partial^2 f_{u,v+1}(x,0)}{\partial x^2} = \\
= \overbrace{\sum_{j=0}^{3} 6a_{3,j}x + \sum_{j=0}^{3} 2a_{2,j}}^{u,v} &= \overbrace{6a_{3,0}x + 2a_{2,0}}^{u,v+1}
\end{aligned} \tag{2.38}$$

Com isso podemos formar as 6 equações abaixo:

$$\begin{aligned}
\overbrace{6a_{3,3}+2a_{2,3}}^{u,v} &= \overbrace{2a_{2,3}}^{u+1,v} \\
\overbrace{6a_{3,2}+2a_{2,2}}^{u,v} &= \overbrace{2a_{2,2}}^{u+1,v} \\
\overbrace{6a_{3,1}+2a_{2,1}}^{u,v} &= \overbrace{2a_{2,1}}^{u+1,v} \\
\overbrace{6a_{3,0}+2a_{2,0}}^{u,v} &= \overbrace{2a_{2,0}}^{u+1,v} \\
\overbrace{6a_{3,3}+6a_{3,2}+6a_{3,1}+6a_{3,0}}^{u,v} &= \overbrace{6a_{3,0}}^{u,v+1} \\
\overbrace{2a_{2,3}+2a_{2,2}+2a_{2,1}+2a_{2,0}}^{u,v} &= \overbrace{2a_{2,0}}^{u,v+1}
\end{aligned} \tag{2.39}$$

No caso das condições de contorno (2.27) com a mesma mudança de base:

$$\frac{\partial^2 f_{u,v}(1,y)}{\partial y^2} = \frac{\partial^2 f_{u+1,v}(0,y)}{\partial y^2} =$$

$$= \overbrace{\sum_{i=0}^{3} 6a_{i,3}y + \sum_{i=0}^{3} 2a_{i,2}}^{u,v} = \overbrace{6a_{0,3}y + 2a_{0,2}}^{u+1,v}$$

$$\frac{\partial^2 f_{u,v}(x,1)}{\partial y^2} = \frac{\partial^2 f_{u,v+1}(x,0)}{\partial y^2} =$$

$$= \overbrace{\sum_{j=2}^{3} j(j-1)a_{3,j}x^3 + \sum_{j=2}^{3} j(j-1)a_{2,j}x^2 + \sum_{j=2}^{3} j(j-1)a_{1,j}x + \sum_{j=2}^{3} j(j-1)a_{0,j}}^{u,v} =$$

$$= \overbrace{2a_{3,2}x^3 + 2a_{2,2}x^2 + 2a_{1,2}x + 2a_{0,2}}^{u,v+1} \quad (2.40)$$

Com isso podemos formar as 6 equações abaixo:

$$\overbrace{6a_{3,3} + 6a_{2,3} + 6a_{1,3} + 6a_{0,3}}^{u,v} = \overbrace{6a_{0,3}}^{u+1,v}$$

$$\overbrace{2a_{3,2} + 2a_{2,2} + 2a_{1,2} + 2a_{0,2}}^{u,v} = \overbrace{2a_{0,2}}^{u+1,v}$$

$$\overbrace{6a_{3,3} + 2a_{3,2}}^{u,v} = \overbrace{2a_{3,2}}^{u,v+1}$$

$$\overbrace{6a_{2,3} + 2a_{2,2}}^{u,v} = \overbrace{2a_{2,2}}^{u,v+1}$$

$$\overbrace{6a_{1,3} + 2a_{1,2}}^{u,v} = \overbrace{2a_{1,2}}^{u,v+1}$$

$$\overbrace{6a_{0,3} + 2a_{0,2}}^{u,v} = \overbrace{2a_{0,2}}^{u,v+1} \quad (2.41)$$

No caso das condições de contorno (2.28) com a mesma mudança de base:

$$\frac{\partial^2 f_{u,v}(1,y)}{\partial x \partial y} = \frac{\partial^2 f_{u+1,v}(0,y)}{\partial x \partial y} =$$

$$= \overbrace{\sum_{i=1}^{3} 3ia_{i,3}y^2 + \sum_{i=1}^{3} 2ia_{i,2}y + \sum_{i=1}^{3} ia_{i,1}}^{u,v} = \overbrace{3a_{1,3}y^2 + 2a_{1,2}y + a_{1,1}}^{u+1,v}$$

$$\frac{\partial^2 f_{u,v}(x,1)}{\partial x \partial y} = \frac{\partial^2 f_{u,v+1}(x,0)}{\partial x \partial y} =$$

$$\overbrace{\sum_{j=1}^{3} 3ja_{3,j}x^2 + \sum_{j=1}^{3} 2ja_{2,j}x + \sum_{j=1}^{3} ja_{1,j}}^{u,v} = \overbrace{3a_{3,1}x^2 + 2a_{2,1}x + a_{1,1}}^{u,v+1} \quad (2.42)$$

Com isso podemos formar as 6 equações abaixo:

$$
\begin{aligned}
\overbrace{9a_{3,3} + 6a_{2,3} + 3a_{1,3}}^{u,v} &= \overbrace{3a_{1,3}}^{u+1,v} \\
\overbrace{6a_{3,2} + 4a_{2,2} + 2a_{1,2}}^{u,v} &= \overbrace{2a_{1,2}}^{u+1,v} \\
\overbrace{3a_{3,1} + 2a_{2,1} + a_{1,1}}^{u,v} &= \overbrace{a_{1,1}}^{u+1,v} \\
\overbrace{9a_{3,3} + 6a_{3,2} + 3a_{3,1}}^{u,v} &= \overbrace{3a_{3,1}}^{u,v+1} \\
\overbrace{6a_{2,3} + 4a_{2,2} + 2a_{2,1}}^{u,v} &= \overbrace{2a_{2,1}}^{u,v+1} \\
\overbrace{3a_{1,3} + 2a_{1,2} + a_{1,1}}^{u,v} &= \overbrace{a_{1,1}}^{u,v+1}
\end{aligned}
\tag{2.43}
$$

Outra condição de continuidade para que o sistema tenha solução e permita realizar a filtragem nos dados é a equação (2.3) referente ao método de Tikhonov Bidimensional.

$$
\frac{\partial^3 f_{u+1,v}(0,0)}{\partial x^3} - \frac{\partial^3 f_{u,v}(1,0)}{\partial x^3} = \frac{1}{\alpha} * (\tilde{z_{u,v}} - \overbrace{a_{0,0}}^{u+1,v}) \tag{2.44}
$$

para $u = 1, 2, \ldots, n-1$ e $v = 2, 3, \ldots, m$.

Juntando as equações podemos formar uma matriz com incógnitas sendo os coeficientes das SPlines. Essa matriz tem dimensão $16*n*m \times 16*n*m$ no mínimo. Para uma imagem de 128×128 pixels, o computador necessitaria de uma matriz com $68,7 * 10^9$ elementos. Se cada elemento for um inteiro, seriam necessários $549,6$ Gigabytes de memória para armazenar essa matriz.

Uma forma alternativa para a resolução desse sistema é dividir a imagem em partes pequenas e para que as condições de contorno tenham pequena influência no resultado, não se deve considerar o valor obtido nos contornos de cada divisão, como na Figura 2.3. A solução foi usada com $n = m = 5$.

O código desenvolvido para este tratamento pode ser encontrado no APÊNDICE B.

2.3.3 Redução de Ordem

As SPlines Cúbicas Bidimensionais definidas anteriormente possuem ordem 6. Uma solução na redução do tempo de processamento da SPline é a Redução de Ordem do polinômio para 3, seguindo o modelo abaixo.

$$
\begin{aligned}
f(x,y) = a_{3,0}x^3 + a_{2,1}x^2y + a_{2,0}x^2 + a_{1,2}xy^2 + a_{1,1}xy + \\
a_{1,0}x + a_{0,3}y^3 + a_{0,2}y^2 + a_{0,1}y + a_{0,0}
\end{aligned}
$$

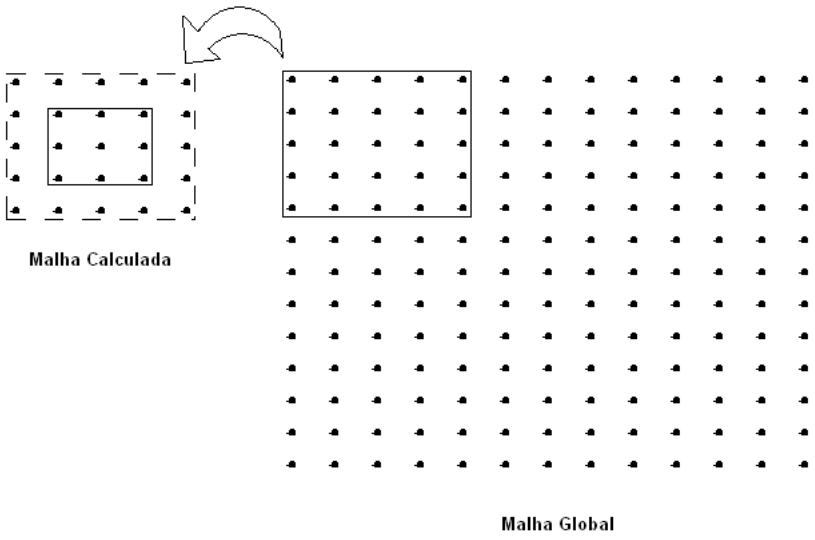

Figura 2.3: Forma de Processamento de Domínio de Dados para SPline Bidimensional

Em testes com o novo polinômio, a imagem apresentou perda de dados e pontos de bordas inesperados para as condições impostas ao sistema. Mas a redução do tempo de processamento, pela redução de 16 variáveis para 10 variáveis por SPline, foi de 25%.

Mas por apresentar bordas incorretas, esse filtro não cumpre os critérios de Canny.

Capítulo 3

Resultados

Os métodos de Laplace e de Sobel desenvolvidos no projeto fazem tratamento de imagens de 256 cores, onde as tonalidades de imagens são tabeladas e apresentam uma forma de função linear, enquanto as fotos anteriormente citadas são apresentadas como RGB, onde cada cor tem uma intensidade de pixel. Por esse fato os 2 métodos não foram aplicados nessas 2 fotos.

As fotos 3.1 e 3.2, mostram uma foto que usa o formato de 256 cores tratada nos dois métodos, este tratamento foi feito em [5]. A foto da LENAG, que aparece no tratamento, foi obtida pelo [6]. Uma observação pode ser feita nos métodos de Laplace e de Sobel que estes além de mostrarem a borda da imagem, são capazes de mostrar as proximidades da existência da borda da imagem (feito através de tons de cinza).

As fotos 3.3 e 3.4 receberam o tratamento por SPlines Cúbicas Unidimensionais e Bidimensionais respectivamente. Nestes casos, foi usado o Métodos de Tikhonov para realizar a filtragem nos dados com as constantes $\alpha = 0,01$ e $\alpha = 0,1$ respectivamente, conforme descrito nas seções referentes aos dois métodos.

Os tratamentos usando o método de Tikhonov apresentaram maior pre-

Figura 3.1: Tratamento com o Método de Sobel

Figura 3.2: Tratamento com o Método de Laplace

Figura 3.3: Tratamento com as SPlines Cúbicas Unidimensionais

Figura 3.4: Tratamento com as SPlines Cúbicas Bidimensionais

cisão no tamanho das bordas e na existência das mesmas, que são dois dos critérios de Canny.

O tratamento de imagem pelo método de Tikhonov foi uma análise [13] de Problemas Inversos e usou como ferramenta [14], descritos na bibliografia, que analisou a melhor aproximação para uma função bem comportada.

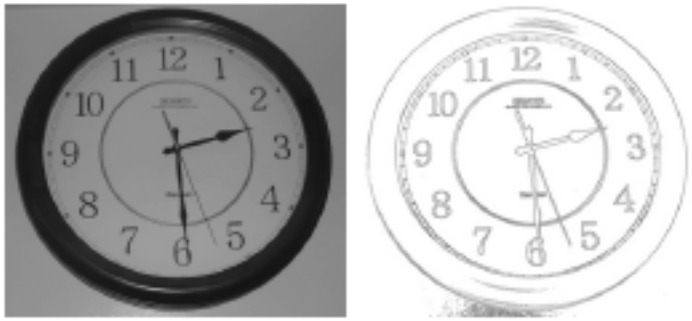

Figura 3.5:

Segundo [13], a demonstração do projeto pode ser feita através da figura 3.5 que mostra a diferença entre os vários tratamentos dados a uma imagem, de acordo com o nível de precisão de cor que a imagem pode possuir.

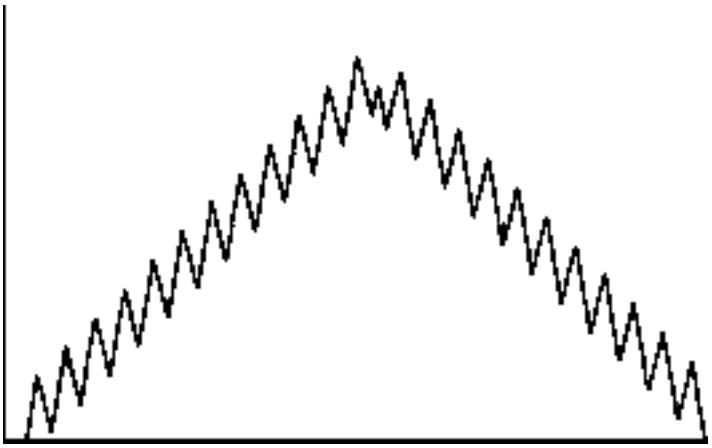

Figura 3.6:

A figura 3.6 mostra os dados de entrada para um software, de implementação própria para a aproximação unidimensional de dados para SPlines cúbicas, e a figura 3.7 mostra o gráfico adaptado para $\alpha = 1$, sendo o mesmo α definido no método de Tikhonov.

As figuras 3.8, 3.9 e 3.10 referem-se à $\alpha = 0.0001, 0.1, 100$.

Em imagens o método se mostrou bem adaptável as condições da imagem, se há grande distúrbio na imagem, ele pode ser removido variando o valor de α.

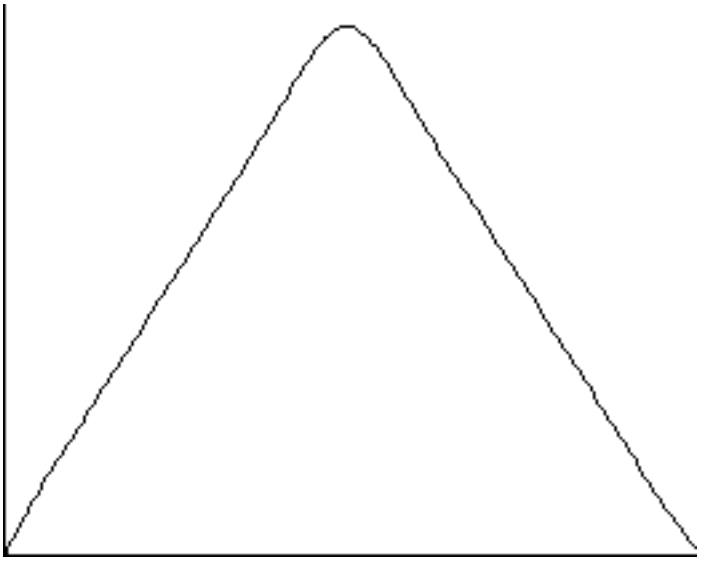

Figura 3.7:

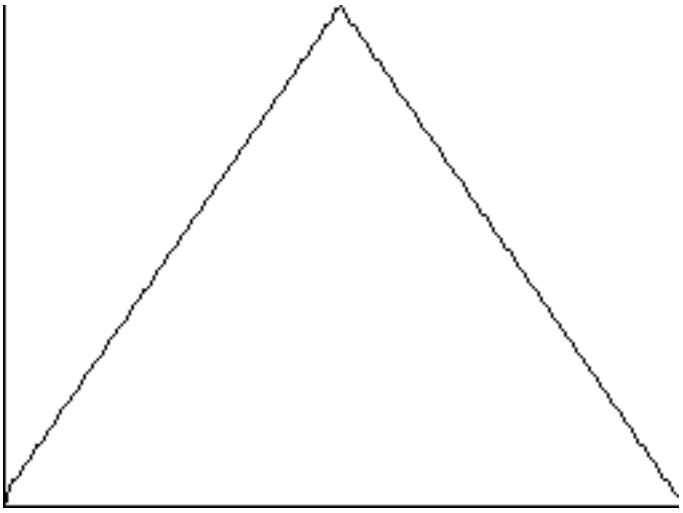

Figura 3.8:

Esta solução pode também ser aplicada para um robô realizando um trabalho de se deslocar em um estacionamento, como pode ser observado em 3.11.

	Unidimensional	Bidimensional
Figura 3.11	$0,000001$	$0,01$
Figura 3.14	$0,000001$	$0,001$
Figura 3.17	$0,000001$	$0,01$

Análise comparativa de α nos Tratamentos.

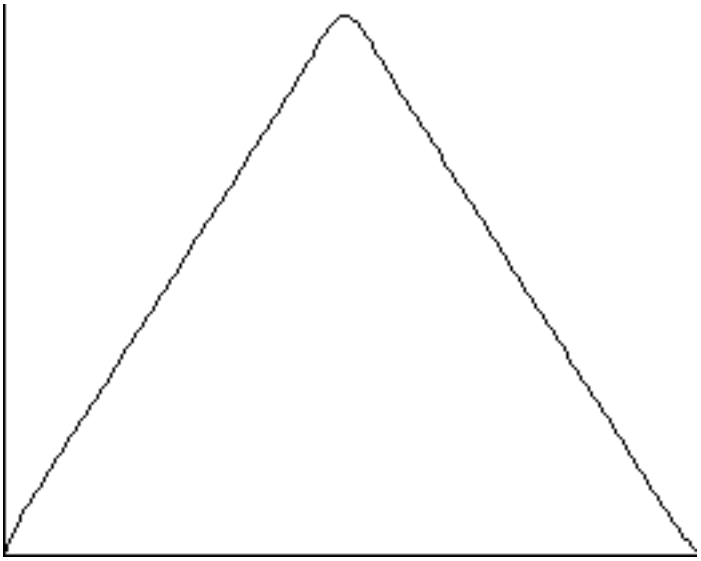

Figura 3.9:

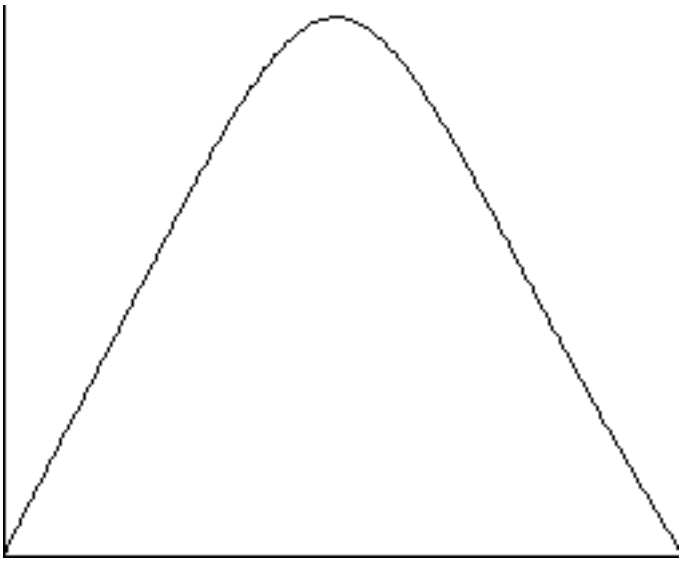

Figura 3.10:

	Unidimensional	Bidimensional
Figura 3.11	$14,3$s	2347s
Figura 3.14	$14,3$s	2277s
Figura 3.17	$13,9$s	2278s

Análise comparativa de tempo nos Tratamentos.
Outros resultado observados:

Figura 3.11: Imagem real de um estacionamento.

Figura 3.12: Tratamento com SPlines Cúbicas Unidimensionais e com o Método de Tikhonov.

Figura 3.13: Tratamento com SPlines Cúbicas Bidimensionais e com o Método de Tikhonov.

Figura 3.14: Imagem real de uma sala.

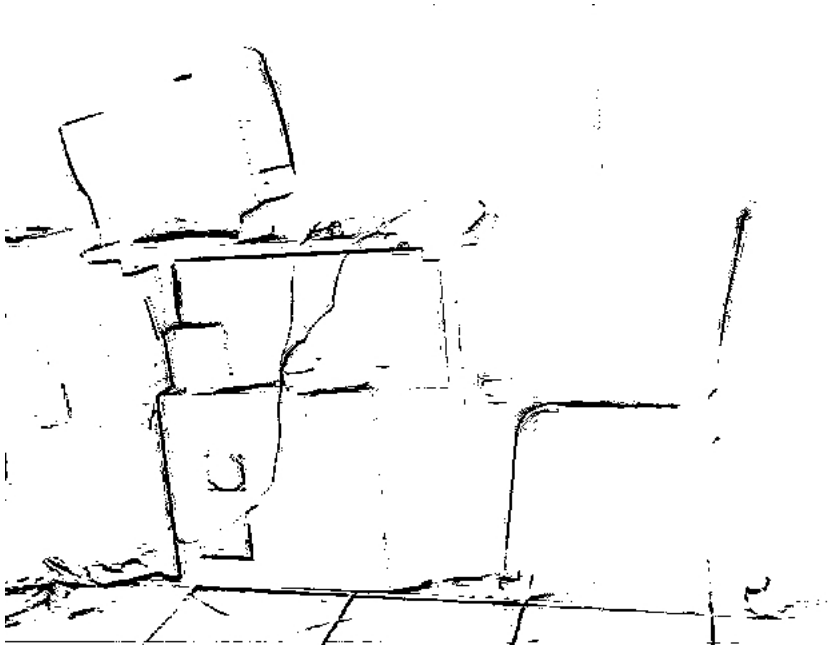

Figura 3.15: Tratamento com SPlines Cúbicas Unidimensionais e com o Método de Tikhonov.

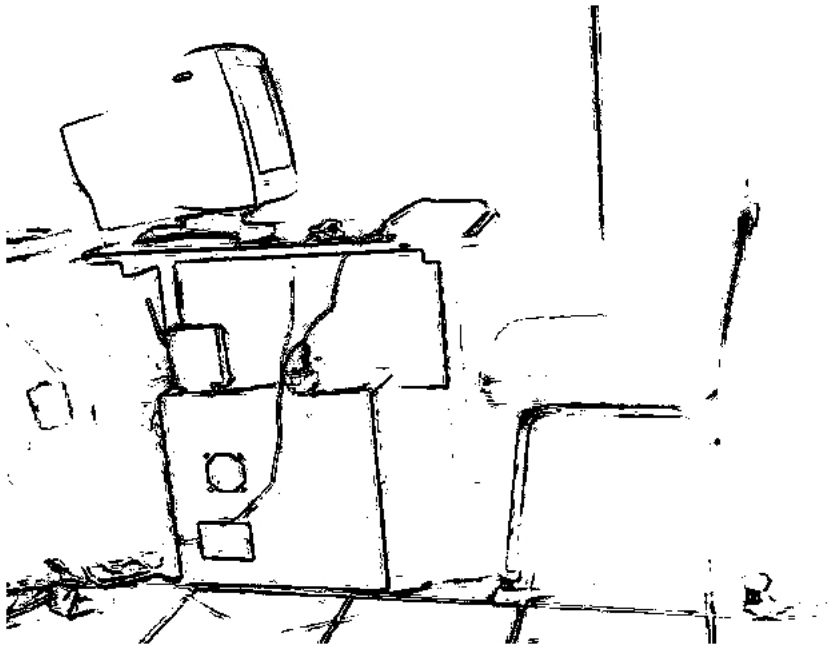

Figura 3.16: Tratamento com SPlines Cúbicas Bidimensionais e com o Método de Tikhonov.

Figura 3.17: Imagem real de uma sala.

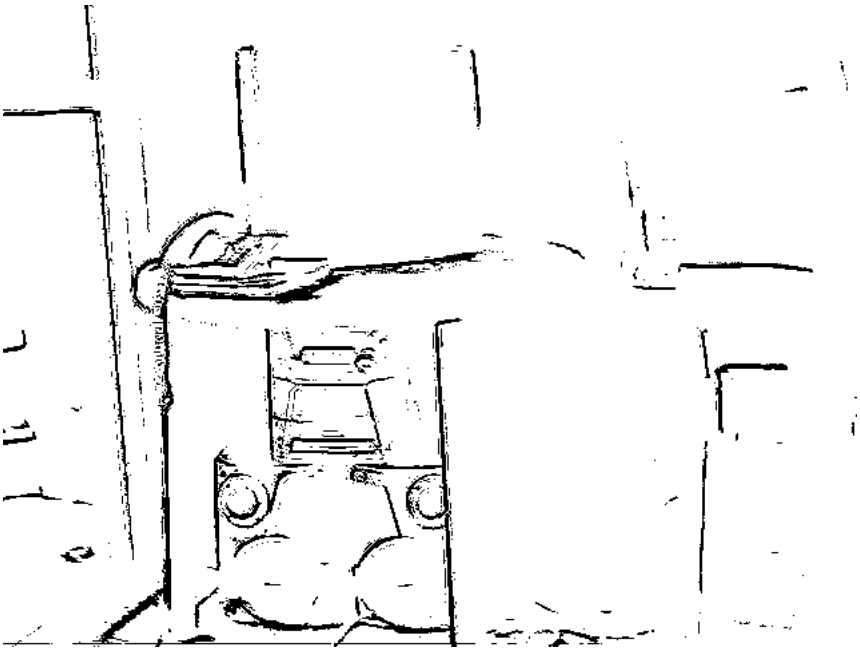

Figura 3.18: Tratamento com SPlines Cúbicas Unidimensionais e com o Método de Tikhonov.

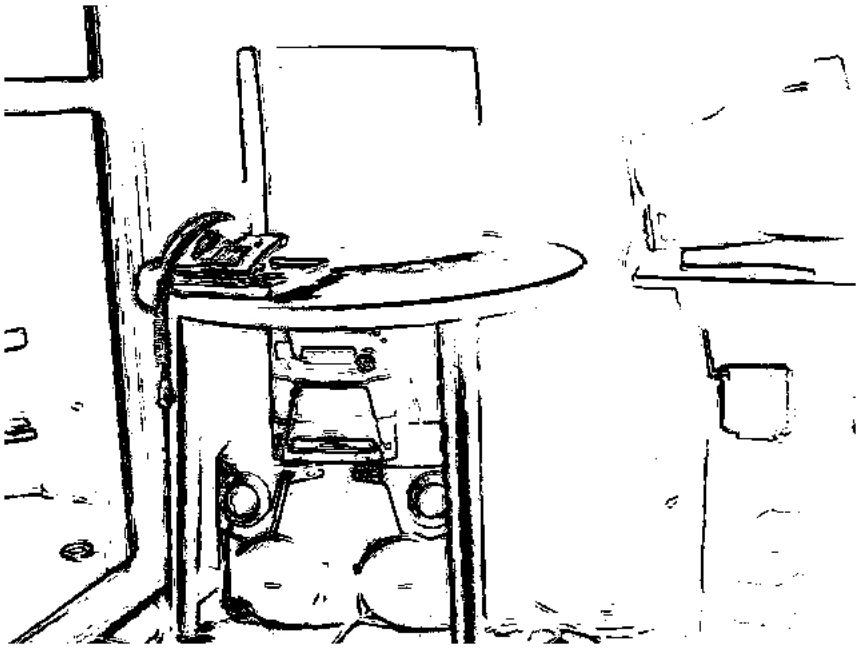

Figura 3.19: Tratamento com SPlines Cúbicas Bidimensionais e com o Método de Tikhonov.

Figura 3.20: Tratamento com SPlines Cúbicas Unidimensionais e com o Método de Tikhonov.

Figura 3.21: Tratamento com SPlines Cúbicas Bidimensionais e com o Método de Tikhonov.

Figura 3.22: Tratamento com SPlines Cúbicas Unidimensionais e com o Método de Tikhonov.

Figura 3.23: Tratamento com SPlines Cúbicas Bidimensionais e com o Método de Tikhonov.

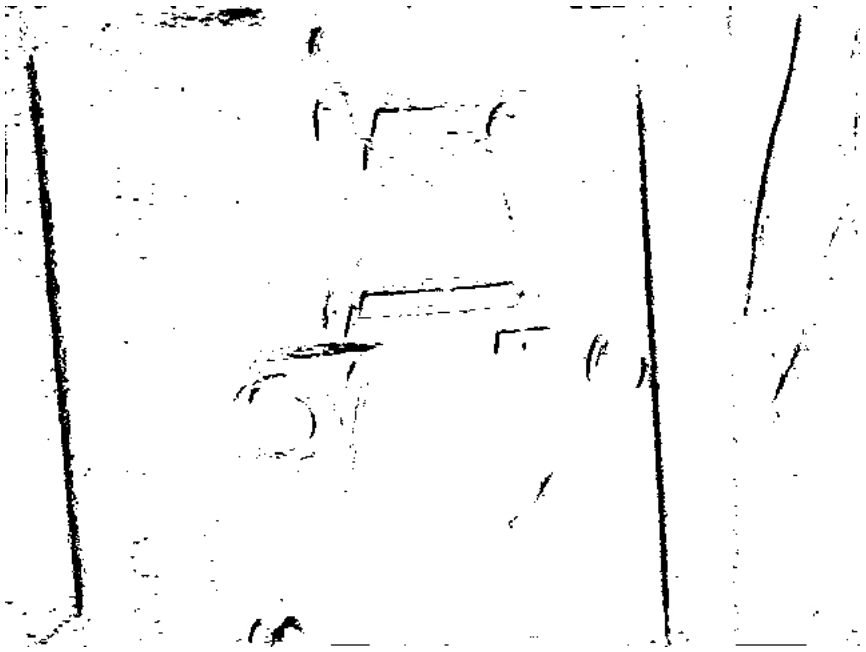

Figura 3.24: Tratamento com SPlines Cúbicas Unidimensionais e com o Método de Tikhonov.

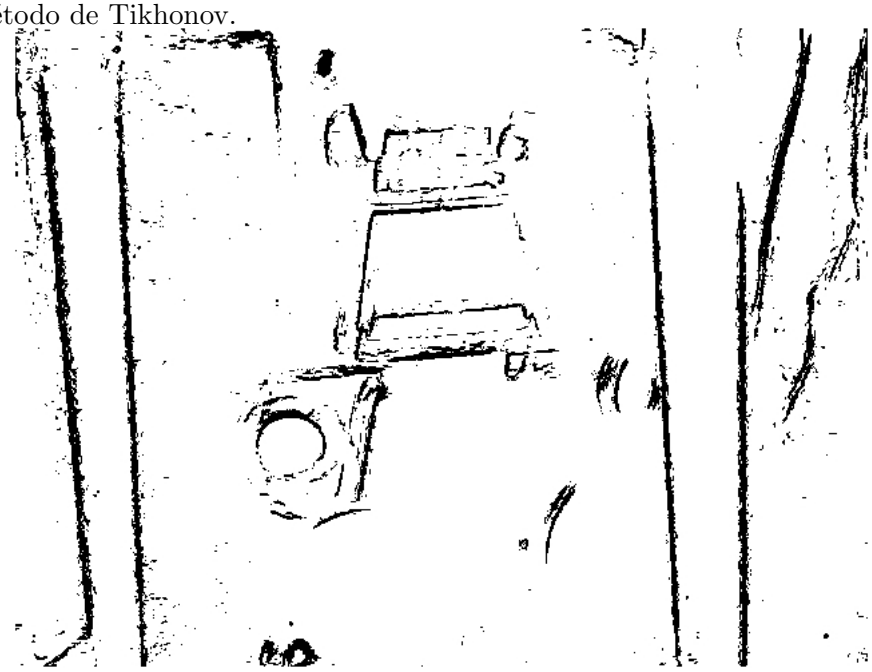

Figura 3.25: Tratamento com SPlines Cúbicas Bidimensionais e com o Método de Tikhonov.

Figura 3.26: Tratamento com SPlines Cúbicas Unidimensionais e com o Método de Tikhonov.

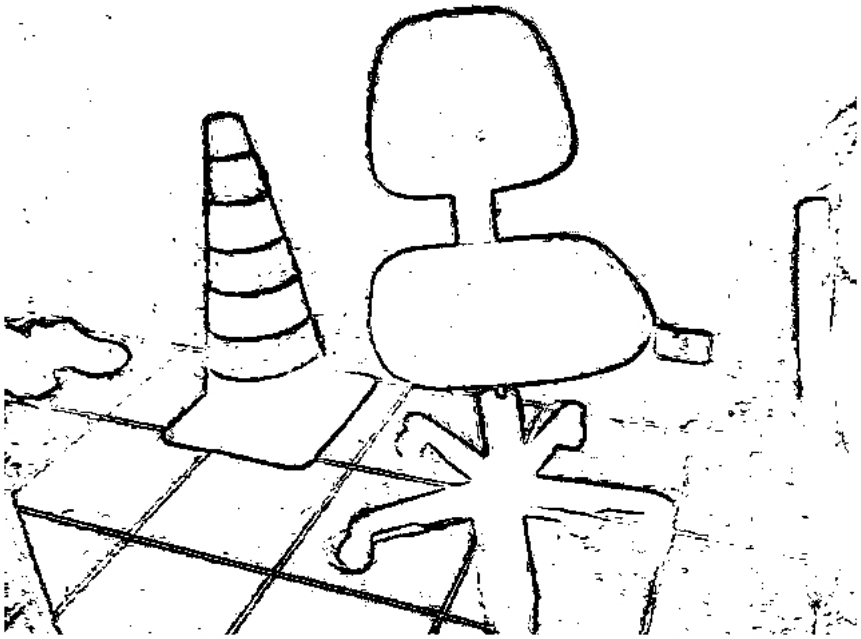

Figura 3.27: Tratamento com SPlines Cúbicas Bidimensionais e com o Método de Tikhonov.

Capítulo 4

Conclusão

Os métodos de Laplace e de Sobel se mostraram rápidos no processamento da imagem, por utilizarem poucas operações.

Mas o método de Laplace, por trabalhar usando a segunda derivada da função imagem, é sensível a ruído, pois cada pequena oscilação provoca o aparecimento de um ponto de pico na primeira derivada.

Já o método de Sobel apresentou um defeito de mostrar bordas muito grossas, tal fato ocorre porque o método usa um comparativo de máximo, mas não permite que ultrapasse o valor máximo da tabela, ou seja, os valores máximos das derivadas não devem ultrapassar 255 (se o valor encontrado pelo operador de Sobel for superior, ele deve ser reduzido ao valor de 255), então o método não encontra um valor de máximo, mas sim inúmeros valores de máximo.

As tabelas de α e de tempo foram montadas para comparar os tratamentos com SPlines Cúbicas Unidimensionais e Bidimensionais.

Os resultados indicam que a precisão do tratamento usando SPlines Cúbicas Bidimensionais é maior por apresentar contornos mais bem definidos para um valor de α maior, ou seja, supondo que a imagem tem menos distúrbios. Mas o tempo de processamento deste método é muito maior.

Caso seja necessária uma precisão muito grande com baixa necessidade de tempo, o tratamento que deve ser escolhido é o de SPlines Cúbicas Bidimensionais. Caso o requisito de uso seja o tempo acima da precisão, a melhor solução é utilizar SPlines Cúbicas Unidimensionais.

Capítulo 5

Trabalhos Futuros

As detecções de bordas feitas com o Método de Tikhonov e SPlines Cúbicas Bidimensionais apresentaram resultados satisfatórios para coeficientes α semelhantes. O tempo de processamento da imagem pode ser considerado alto.

Uma solução para a redução do tempo de processamento sem perda da eficiência é produzir uma máscara semelhante a usada pelos métodos de Sobel e Laplace, para as SPlines Cúbicas para valores definidos de α, ou seja, fixa um valor para o coeficiente α, gera uma matriz com as condições do problema e como resultado um vetor com valores genéricos $\tilde{z}$. Ao escalonar e resolver a matriz, aparecerão coeficientes multiplicados aos valores de $\tilde{z_{i,j}}$, esses coeficientes podem são montados em uma máscara similar às dos métodos de Laplace e Sobel.

Referências Bibliográficas

[1] E. A. S. G. G M do Vale and A. P. D. Poz. O detector de canny-edp: Uma combinação entre as teorias de canny e de difusão anisotrópica não linear. *Revista Brasileira de Cartografia*, 56(2):156–168, Dec. 2004.

[2] B. Green. Canny edge detection tutorial. http://www.pages.drexel.edu/ weg22/can_ tut.html, 2002.

[3] B. Green. Código fonte para implementação do método de laplace. http://www.pages.drexel.edu/ weg22/edgelap.zip, 2002.

[4] B. Green. Código fonte para implementação do método de sobel. http://www.pages.drexel.edu/ weg22/edgesob.zip, 2002.

[5] B. Green. Edge detection tutorial. http://www.pages.drexel.edu/ weg22/edge.html, 2002.

[6] B. Green. Lenag, foto utilizada como base para o tratamento de imagens. http://www.pages.drexel.edu/ weg22/LENAG.bmp, 2002.

[7] B. Green. Mask, imagem que representa a forma de uso das máscaras nos métodos de sobel e laplace. http://www.pages.drexel.edu/ weg22/maskdes.jpg, 2002.

[8] A. M. Jason Nearing, Rob Pickel. Edge detection methods. http://www.owlnet.rice.edu/ elec539/Projects97/segment/edge.html.

[9] M. Johnson. Curso artificial intelligence. http://www.massey.ac.nz/ mjjohnso/notes/59302/l13.html, 1997. Seção 13.

[10] J. G. Kirsten Rieth. *Edge Detection In Noisy Environments, A Fuzzy Reasoning Method*. National Aeronautics and Space Administration, 2003.

[11] M. S. N. P. Mgr. A. A. Frolov. An efficient edge detection approach in image processing. Technical report, Access Server, 2006.

[12] F. B. S. Roberto López Rosas, Adriano de Luca. Simd architecture for image segmentation using sobel operators implemented in fpga technology. *Electrical and Electronics Engineering, 2005 2nd International Conference*, pages 77–80, sep 2005.

[13] Y. B. W. X Q Wan and M. Yamamoto. Detection of irregular points by regularization in numerical differentiation and application to edge detection. *IOP Publishing Ltd*, 22(3):1089–1103, May 2006.

[14] X. Z. J. Y B Wang and J. Cheng. A numerical differentiation method and its application to reconstruction of discontinuity. *IOP Publishing Ltd*, 18(3):1461–1476, Sept. 2002.

[15] T. J. T. Z Y Qian, C Z Cheng and L. L. Yun. Medical images edge detection based on mathematical morphology. In *Proceedings of the 2005 IEEE*, pages 6492–6495, Sept. 2005.

1

[1]De acordo com:
ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 6023: informação e documentação: referências: elaboração. Rio de Janeiro, 2002.

ANEXOS

ANEXO A - Laplace

edgeLap.c - Obtido em [3]

```
/*
  FILE: edgeLap.c - WORKS!
  AUTH: P.Oh
  DESC: 5x5 Laplace mask for edge detection
  DATE: 05/02/02 23:30
  REFS: edgedeta.c
*/

#include (stdio.h)
#include (stdlib.h)
#include (math.h)
#include (alloc.h)

/*-------STRUCTURES---------*/
typedef struct {int rows; int cols;
                unsigned char* data;} sImage;

/*-------PROTOTYPES---------*/
long getImageInfo(FILE*, long, int);
void copyImageInfo(FILE* inputFile, FILE* outputFile);
void copyColorTable(FILE* inputFile, FILE* outputFile,
                    int nColors);

int main(int argc, char* argv[])
{
   FILE         *bmpInput, *bmpOutput;
   sImage      originalImage;
   sImage      edgeImage;
   unsigned int      X, Y;
   int         I, J;
   long         SUM;
   int         nColors;
   unsigned long   vectorSize;
   unsigned long   fileSize;
   int         MASK[5][5];
   unsigned char    *pChar, someChar;
   unsigned int      row, col;

   someChar = '0'; pChar = &someChar;
```

```
/* 5x5 Laplace mask.  Ref: Myler Handbook p. 135 */
MASK[0][0] = -1; MASK[0][1] = -1; MASK[0][2] = -1;
MASK[0][3] = -1; MASK[0][4] = -1;
MASK[1][0] = -1; MASK[1][1] = -1; MASK[1][2] = -1;
MASK[1][3] = -1; MASK[1][4] = -1;
MASK[2][0] = -1; MASK[2][1] = -1; MASK[2][2] = 24;
MASK[2][3] = -1; MASK[2][4] = -1;
MASK[3][0] = -1; MASK[3][1] = -1; MASK[3][2] = -1;
MASK[3][3] = -1; MASK[3][4] = -1;
MASK[4][0] = -1; MASK[4][1] = -1; MASK[4][2] = -1;
MASK[4][3] = -1; MASK[4][4] = -1;

if(argc < 2) {
  printf("Usage: %s bmpInput.bmp\n", argv[0]);
  exit(0);
};
printf("Reading filename %s\n", argv[1]);

/* open files for reading and writing to */
bmpInput = fopen(argv[1], "rb");
bmpOutput = fopen("edgeLap.bmp", "wb");

/* start pointer at beginning of file */
fseek(bmpInput, 0L, SEEK_END);

/* retrieve and print filesize and number
   of cols and rows */
fileSize = getImageInfo(bmpInput, 2, 4);
originalImage.cols = (int)getImageInfo(bmpInput, 18, 4);
originalImage.rows = (int)getImageInfo(bmpInput, 22, 4);
edgeImage.rows = originalImage.rows;
edgeImage.cols = originalImage.cols;

printf("Width: %d\n", originalImage.cols);
printf("Height: %d\n", originalImage.rows);
printf("File size: %lu\n", fileSize);

/* retrieve and print Number of colors */
nColors = (int)getImageInfo(bmpInput, 46, 4);
printf("nColors: %d\n", nColors);

vectorSize = fileSize - (14+40+4*nColors);
printf("vectorSize: %lu\n", vectorSize);
edgeImage.data =
```

```
farmalloc(vectorSize*sizeof(unsigned char));
if(edgeImage.data == NULL) {
  printf("Failed to malloc edgeImage.data\n");
  exit(0);
}
printf("%lu bytes malloc'ed for
         edgeImage.data\n", vectorSize);

originalImage.data =
farmalloc(vectorSize*sizeof(unsigned char));
if(originalImage.data == NULL) {
  printf("Failed to malloc originalImage.data\n");
  exit(0);
}
printf("%lu bytes malloc'ed for
        originalImage.datt\n", vectorSize);

copyImageInfo(bmpInput, bmpOutput);
copyColorTable(bmpInput, bmpOutput, nColors);
fseek(bmpInput, (14+40+4*nColors), SEEK_SET);
fseek(bmpOutput, (14+40+4*nColors), SEEK_SET);

/* Read input.bmp and store it's raster data into
   originalImage.data */
for(row=0; row<=originalImage.rows-1; row++) {
  for(col=0; col<=originalImage.cols-1; col++) {
    fread(pChar, sizeof(char), 1, bmpInput);
    *(originalImage.data + row*originalImage.cols +
      col) = *pChar;
  }
}

for(Y=0; Y<=(originalImage.rows-1); Y++)  {
  for(X=0; X<=(originalImage.cols-1); X++)  {
    SUM = 0;

    /* image boundaries */
    if(Y==0 || Y==1 || Y==originalImage.rows-2 ||
       Y==originalImage.rows-1)
      SUM = 0;
    else if(X==0 || X==1 || X==originalImage.cols-2 ||
            X==originalImage.cols-1)
      SUM = 0;
```

```
      /* Convolution starts here */
      else   {
        for(I=-2; I<=2; I++)  {
          for(J=-2; J<=2; J++)  {
            SUM = SUM + (int)( (*(originalImage.data +
            X + I + (Y + J)*originalImage.cols)) *
            MASK[I+2][J+2]);

          }
        }
      }
      if(SUM>255)  SUM=255;
      if(SUM<0)    SUM=0;

      *(edgeImage.data + X + Y*originalImage.cols) =
      255 - (unsigned char)(SUM);
      fwrite((edgeImage.data+X+Y*originalImage.cols),
               sizeof(char),1,bmpOutput);
    }
  }

  printf("See edgeLap.bmp for results\n");
  fclose(bmpInput);
  fclose(bmpOutput);
  /* Finished with edgeImage.data */
  farfree(edgeImage.data);
  /* Finished with originalImage.data */
  farfree(originalImage.data);
  return 0;
}

/*----------GET IMAGE INFO SUBPROGRAM--------------*/
long getImageInfo(FILE* inputFile, long offset,
                  int numberOfChars)
{
  unsigned char         *ptrC;
  long            value = 0L;
  unsigned char         dummy;
  int            i;

  dummy = '0';
  ptrC = &dummy;

  fseek(inputFile, offset, SEEK_SET);
```

```
  for(i=1; i<=numberOfChars; i++)
  {
    fread(ptrC, sizeof(char), 1, inputFile);
    /* calculate value based on adding bytes */
    value = (long)(value + (*ptrC)*(pow(256, (i-1))));
  }
  return(value);

} /* end of getImageInfo */

/*-------------COPIES HEADER AND INFO HEADER----------------*/
void copyImageInfo(FILE* inputFile, FILE* outputFile)
{
  unsigned char      *ptrC;
  unsigned char      dummy;
  int            i;

  dummy = '0';
  ptrC = &dummy;

  fseek(inputFile, 0L, SEEK_SET);
  fseek(outputFile, 0L, SEEK_SET);

  for(i=0; i<=50; i++)
  {
    fread(ptrC, sizeof(char), 1, inputFile);
    fwrite(ptrC, sizeof(char), 1, outputFile);
  }

}

/*----------------COPIES COLOR TABLE----------------*/
void copyColorTable(FILE* inputFile, FILE* outputFile,
                    int nColors)
{
  unsigned char      *ptrC;
  unsigned char      dummy;
  int            i;

  dummy = '0';
  ptrC = &dummy;

  fseek(inputFile, 54L, SEEK_SET);
```

```
  fseek(outputFile, 54L, SEEK_SET);

  /* there are (4*nColors) bytesin color table */
  for(i=0; i<=(4*nColors); i++)
  {
    fread(ptrC, sizeof(char), 1, inputFile);
    fwrite(ptrC, sizeof(char), 1, outputFile);
  }

}
```

ANEXO B - Sobel

edgeSob.c - Obtido em [4]

```
/*
  FILE: edgeSob.c - WORKS!!
  AUTH: Bill Green
  DESC: 2 3x3 Sobel masks for edge detection
  DATE: 07/23/02
  REFS: edgeLap.c
*/

#include (stdio.h)
#include (stdlib.h)
#include (math.h)
#include (alloc.h)

/*-------STRUCTURES---------*/
typedef struct {int rows; int cols;
                unsigned char* data;} sImage;

/*-------PROTOTYPES---------*/
long getImageInfo(FILE*, long, int);
void copyImageInfo(FILE* inputFile, FILE* outputFile);
void copyColorTable(FILE* inputFile, FILE* outputFile,
                    int nColors);

int main(int argc, char* argv[])
{
   FILE         *bmpInput, *bmpOutput;
   sImage      originalImage;
   sImage      edgeImage;
   unsigned int      X, Y;
   int         I, J;
   long         sumX, sumY;
   int         nColors, SUM;
   unsigned long   vectorSize;
   unsigned long   fileSize;
   int         GX[3][3];
   int         GY[3][3];
   unsigned char    *pChar, someChar;
   unsigned int      row, col;

   someChar = '0'; pChar = &someChar;
```

```
/* 3x3 GX Sobel mask.
   Ref: www.cee.hw.ac.uk/hipr/html/sobel.html */
GX[0][0] = -1; GX[0][1] = 0; GX[0][2] = 1;
GX[1][0] = -2; GX[1][1] = 0; GX[1][2] = 2;
GX[2][0] = -1; GX[2][1] = 0; GX[2][2] = 1;

/* 3x3 GY Sobel mask.
   Ref: www.cee.hw.ac.uk/hipr/html/sobel.html */
GY[0][0] =  1; GY[0][1] =  2; GY[0][2] =  1;
GY[1][0] =  0; GY[1][1] =  0; GY[1][2] =  0;
GY[2][0] = -1; GY[2][1] = -2; GY[2][2] = -1;

if(argc < 2) {
  printf("Usage: %s bmpInput.bmp\n", argv[0]);
  exit(0);
};
printf("Reading filename %s\n", argv[1]);

/*-------DECLARE INPUT & OUTPUT FILES-------*/
bmpInput = fopen(argv[1], "rb");
bmpOutput = fopen("edgeSob.bmp", "wb");

/*---SET POINTER TO BEGINNING OF FILE----*/
fseek(bmpInput, 0L, SEEK_END);

/*-------GET INPUT BMP DATA--------*/
fileSize = getImageInfo(bmpInput, 2, 4);
originalImage.cols = (int)getImageInfo(bmpInput, 18, 4);
originalImage.rows = (int)getImageInfo(bmpInput, 22, 4);
edgeImage.rows = originalImage.rows;
edgeImage.cols = originalImage.cols;

/*--------PRINT DATA TO SCREEN----------*/
printf("Width: %d\n", originalImage.cols);
printf("Height: %d\n", originalImage.rows);
printf("File size: %lu\n", fileSize);

nColors = (int)getImageInfo(bmpInput, 46, 4);
printf("nColors: %d\n", nColors);

/*------ALLOCATE MEMORY FOR FILES--------*/
vectorSize = fileSize - (14+40+4*nColors);
printf("vectorSize: %lu\n", vectorSize);
edgeImage.data =
```

```
farmalloc(vectorSize*sizeof(unsigned char));
if(edgeImage.data == NULL) {
  printf("Failed to malloc edgeImage.data\n");
  exit(0);
}
printf("%lu bytes malloc'ed for
        edgeImage.data\n", vectorSize);

originalImage.data =
farmalloc(vectorSize*sizeof(unsigned char));
if(originalImage.data == NULL) {
  printf("Failed to malloc originalImage.data\n");
  exit(0);
}
printf("%lu bytes malloc'ed for
        originalImage.datt\n", vectorSize);

/*------COPY HEADER AND COLOR TABLE---------*/
copyImageInfo(bmpInput, bmpOutput);
copyColorTable(bmpInput, bmpOutput, nColors);
fseek(bmpInput, (14+40+4*nColors), SEEK_SET);
fseek(bmpOutput, (14+40+4*nColors), SEEK_SET);

/* Read input.bmp and store it's raster data into
   originalImage.data */
for(row=0; row<=originalImage.rows-1; row++) {
  for(col=0; col<=originalImage.cols-1; col++) {
    fread(pChar, sizeof(char), 1, bmpInput);
    *(originalImage.data + row*originalImage.cols + col) =
    *pChar;
  }
}

/*-------------------------------------------------
             SOBEL ALGORITHM STARTS HERE
-------------------------------------------------*/
for(Y=0; Y<=(originalImage.rows-1); Y++)  {
  for(X=0; X<=(originalImage.cols-1); X++)  {
    sumX = 0;
    sumY = 0;

          /* image boundaries */
    if(Y==0 || Y==originalImage.rows-1)
      SUM = 0;
```

```
     else if(X==0 || X==originalImage.cols-1)
       SUM = 0;

            /* Convolution starts here */
     else   {

            /*-------X GRADIENT APPROXIMATION------*/
       for(I=-1; I<=1; I++)  {
         for(J=-1; J<=1; J++)  {
           sumX = sumX + (int)( (*(originalImage.data +
           X + I + (Y + J)*originalImage.cols)) *
           GX[I+1][J+1]);
         }
       }

            /*-------Y GRADIENT APPROXIMATION-------*/
       for(I=-1; I<=1; I++)  {
         for(J=-1; J<=1; J++)  {
           sumY = sumY + (int)( (*(originalImage.data +
           X + I + (Y + J)*originalImage.cols)) *
           GY[I+1][J+1]);
         }
       }

/*---GRADIENT MAGNITUDE APPROXIMATION (Myler p.218)----*/
       SUM = abs(sumX) + abs(sumY);
     }

     if(SUM>255) SUM=255;
     if(SUM<0) SUM=0;

     *(edgeImage.data + X + Y*originalImage.cols) =
     255 - (unsigned char)(SUM);
     fwrite((edgeImage.data+X+Y*originalImage.cols),
                sizeof(char),1,bmpOutput);
   }
 }

 printf("See edgeSob.bmp for results\n");
 fclose(bmpInput);
 fclose(bmpOutput);
 /* Finished with edgeImage.data */
 farfree(edgeImage.data);
 /* Finished with originalImage.data */
```

```
   farfree(originalImage.data);
   return 0;
}

/*----------GET IMAGE INFO SUBPROGRAM--------------*/
long getImageInfo(FILE* inputFile, long offset,
                  int numberOfChars)
{
  unsigned char         *ptrC;
  long            value = 0L;
  unsigned char         dummy;
  int            i;

  dummy = '0';
  ptrC = &dummy;

  fseek(inputFile, offset, SEEK_SET);

  for(i=1; i<=numberOfChars; i++)
  {
    fread(ptrC, sizeof(char), 1, inputFile);
    /* calculate value based on adding bytes */
    value = (long)(value + (*ptrC)*(pow(256, (i-1))));
  }
  return(value);

} /* end of getImageInfo */

/*-------------COPIES HEADER AND INFO HEADER----------------*/
void copyImageInfo(FILE* inputFile, FILE* outputFile)
{
  unsigned char      *ptrC;
  unsigned char      dummy;
  int            i;

  dummy = '0';
  ptrC = &dummy;

  fseek(inputFile, 0L, SEEK_SET);
  fseek(outputFile, 0L, SEEK_SET);

  for(i=0; i<=50; i++)
  {
    fread(ptrC, sizeof(char), 1, inputFile);
```

```
    fwrite(ptrC, sizeof(char), 1, outputFile);
  }

}

/*----------------COPIES COLOR TABLE----------------*/
void copyColorTable(FILE* inputFile, FILE* outputFile,
                    int nColors){
  unsigned char      *ptrC;
  unsigned char      dummy;
  int            i;

  dummy = '0';
  ptrC = &dummy;

  fseek(inputFile, 54L, SEEK_SET);
  fseek(outputFile, 54L, SEEK_SET);

  /* there are (4*nColors) bytesin color table */
  for(i=0; i<=(4*nColors); i++)
  {
    fread(ptrC, sizeof(char), 1, inputFile);
    fwrite(ptrC, sizeof(char), 1, outputFile);
  }

}
```

APÊNDICE A - Unidimensional

unidim.c - Este software necessita do pacote ImageMagick

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX 2800
#define MAX2 700
#define EPS 1e-4

double A[MAX][8], x[MAX], b[MAX], ai[MAX], bi[MAX],
       ci[MAX], di[MAX], yi[MAX], alfa, beta;
int ni, nj, l, R[MAX2][MAX2], G[MAX2][MAX2],
       B[MAX2][MAX2], RESP[MAX2][MAX2], v[MAX];
FILE *arq;

void zeraA(){
   int i, j;
   for(i=0; i < 4*nj; i++)
     for(j=0; j < 8; j++)
       A[i][j] = 0;
}

void zerab(){
   int j;
   for(j=0; j < 4*nj; j++)
     b[j] = 0;
}

void calculaMatrizes(){
   int i, k;
   zeraA();
   zerab();
   A[0][1] = 1;
   v[0] = 0;
   A[1][3] = 1;
   v[1] = 0;
   b[1] = yi[0];
   for(i=1; i < nj; i++){
     k = 4*i-2;
     A[k][0] = -6;
```

```
      A[k][4] = 6;
      A[k][7] = 1.0/alfa;
      v[k] = 4*(i-1);
      b[k] = 1.0/alfa*yi[i+1];
      A[k+1][0] = -6;
      A[k+1][1] = -2;
      A[k+1][5] = 2;
      v[k+1] = 4*(i-1);
      A[k+2][0] = -3;
      A[k+2][1] = -2;
      A[k+2][2] = -1;
      A[k+2][6] = 1;
      v[k+2] = 4*(i-1);
      A[k+3][0] = -1;
      A[k+3][1] = -1;
      A[k+3][2] = -1;
      A[k+3][3] = -1;
      A[k+3][7] = 1;
      v[k+3] = 4*(i-1);
   }
   A[4*nj-2][4] = 6;
   A[4*nj-2][5] = 2;
   v[4*nj-2] = 4*nj - 8;
   A[4*nj-1][4] = 1;
   A[4*nj-1][5] = 1;
   A[4*nj-1][6] = 1;
   A[4*nj-1][7] = 1;
   v[4*nj-1] = 4*nj - 8;
   b[4*nj-1] = yi[nj];
}

void eliminacaoGauss(){
   int i, j, k, aux2;
   double aux;
   for(i=0; i< 4*nj-1; i++){
     if(A[i][i-v[i]] == 0){
       for(k=i+1; k < 4*nj && v[k] - v[i] > -7 &&
                  v[k]-v[i] < 7 && A[k][i-v[k]] == 0; k++);
       for(j=0; j < 8; j++){
         aux = A[i][j];
         A[i][j] = A[k][j];
         A[k][j] = aux;
       }
       aux = b[i];
```

```
        b[i] = b[k];
        b[k] = aux;
        aux2 = v[i];
        v[i] = v[k];
        v[k] = aux2;
      }
      for(k=i+1; k < 4*nj && v[k] - v[i] > -7 &&
                     v[k]-v[i] < 7; k++){
        if(i-v[k] >= 0){
          aux = A[k][i-v[k]]/A[i][i-v[i]];
          if(aux > EPS || aux < -EPS){
            if(v[k] == v[i])
              for(j=0; j < 8; j++)
                A[k][j] = A[k][j] - aux*A[i][j];
            if(v[k] > v[i])
              for(j=0; j < 4; j++)
                A[k][j] = A[k][j] - aux*A[i][j+4];
            if(v[k] < v[i])
              for(j=4; j < 8; j++)
                A[k][j] = A[k][j] - aux*A[i][j-4];
            b[k] = b[k] - aux*b[i];
          }
        }
      }
    }
    for(i=4*nj-1; i>= 0; i--){
      x[i] = b[i];
      for(j=i+1-v[i]; j < 8; j++)
        x[i] = x[i] - A[i][j]*x[j+v[i]];
      x[i] = x[i]/A[i][i-v[i]];
    }
}

void leate(char c){
    char aux;
    aux = -1;
    while(aux != c)
      fscanf(arq, "%c", &aux);
}

void recebeDados(){
    int i, j;
    char c;
    arq = fopen("fig.txt", "r");
```

```
   if(arq == NULL){
     printf("O arquivo especificado não existe,
              favor tente novamente!\n");
     exit(-1);
   }
   leate('\n');
   while(!feof(arq) && !ferror(arq)){
     fscanf(arq, "%d", &j);
     c = fgetc(arq);
     if(c < 0)
       break;
     fscanf(arq, "%d", &i);
     leate('(');
     fscanf(arq, "%d", &R[i][j]);
     leate(',');
     fscanf(arq, "%d", &G[i][j]);
     leate(',');
     fscanf(arq, "%d", &B[i][j]);
     leate('\n');
     ni = i;
     nj = j;
   }
   fclose(arq);
}

void verificaBorda(char tipo){
   int i;
   if(tipo == 'h'){
     for(i=1; i <= nj; i++)
       if(ci[i] >= beta)
         RESP[l][i-1] = 1;
     if(3*ai[nj+1]+2*bi[nj+1]+ci[nj+1] >= beta)
       RESP[l][nj] = 1;
   }
   if(tipo == 'v'){
     for(i=1; i <= ni; i++)
       if(ci[i] >= beta)
         RESP[i-1][l] = 1;
     if(3*ai[nj+1]+2*bi[nj+1]+ci[nj+1] >= beta)
       RESP[ni][l] = 1;
   }
}

void imprimeBorda(){
```

```
   int i, j;

   arq = fopen("sai.txt", "w");
   fprintf(arq, "# ImageMagick pixel enumeration:
                 %d,%d,255,RGB\n",nj+1,ni+1);

   for(i=0; i <= ni; i++)
     for(j=0; j <= nj; j++){
       fprintf(arq, "%d,%d: (", j, i);
       if(RESP[i][j] == 1)
         fprintf(arq, "0,0,0) black\n");
       else
         fprintf(arq, "255,255,255) white\n");
     }
}

void transformaImagemTexto(char *nome){
   int i;
   char comando[100];
   comando[0] = 'c';
   comando[1] = 'o';
   comando[2] = 'n';
   comando[3] = 'v';
   comando[4] = 'e';
   comando[5] = 'r';
   comando[6] = 't';
   comando[7] = ' ';
   for(i=0; nome[i] != 0; i++)
     comando[i+8] = nome[i];
   comando[i+8] = ' ';
   comando[i+9] = 'f';
   comando[i+10] = 'i';
   comando[i+11] = 'g';
   comando[i+12] = '.';
   comando[i+13] = 't';
   comando[i+14] = 'x';
   comando[i+15] = 't';
   comando[i+16] = 0;
   system(comando);
}

int main(int argc, char *argv[]){
   int i, j;
   char *nome;
```

```
nome = argv[1];
transformaImagemTexto(nome);
recebeDados();
alfa = atof(argv[2]);
beta = atof(argv[3]);
for(i=0; i <= ni; i++)
  for(j=0; j <= nj; j++)
    RESP[i][j] = 0;
/*Verifica nas linhas horizontais os contornos de imagem*/
for(l=0; l <= ni; l++){
  for(i=0; i <= nj; i++)
    yi[i] = R[l][i];
  calculaMatrizes();
  eliminacaoGauss();
  for(i=1; i <= nj; i++){
    ai[i] = x[4*i-4];
    bi[i] = x[4*i-3];
    ci[i] = x[4*i-2];
    di[i] = x[4*i-1];
  }
  verificaBorda('h');
}
for(l=0; l <= ni; l++){
  for(i=0; i <= nj; i++)
    yi[i] = G[l][i];
  calculaMatrizes();
  eliminacaoGauss();
  for(i=1; i <= nj; i++){
    ai[i] = x[4*i-4];
    bi[i] = x[4*i-3];
    ci[i] = x[4*i-2];
    di[i] = x[4*i-1];
  }
  verificaBorda('h');
}
for(l=0; l <= ni; l++){
  for(i=0; i <= nj; i++)
    yi[i] = B[l][i];
  calculaMatrizes();
  eliminacaoGauss();
  for(i=1; i <= nj; i++){
    ai[i] = x[4*i-4];
    bi[i] = x[4*i-3];
    ci[i] = x[4*i-2];
```

```
    di[i] = x[4*i-1];
  }
  verificaBorda('h');
}
/*Verifica nas linhas horizontais os contornos de imagem*/
for(l=0; l <= nj; l++){
  for(i=0; i <= ni; i++)
    yi[i] = R[i][l];
  calculaMatrizes();
  eliminacaoGauss();
  for(i=1; i <= ni; i++){
    ai[i] = x[4*i-4];
    bi[i] = x[4*i-3];
    ci[i] = x[4*i-2];
    di[i] = x[4*i-1];
  }
  verificaBorda('v');
}
for(l=0; l <= nj; l++){
  for(i=0; i <= ni; i++)
    yi[i] = G[i][l];
  calculaMatrizes();
  eliminacaoGauss();
  for(i=1; i <= ni; i++){
    ai[i] = x[4*i-4];
    bi[i] = x[4*i-3];
    ci[i] = x[4*i-2];
    di[i] = x[4*i-1];
  }
  verificaBorda('v');
}
for(l=0; l <= nj; l++){
  for(i=0; i <= ni; i++)
    yi[i] = B[i][l];
  calculaMatrizes();
  eliminacaoGauss();
  for(i=1; i <= ni; i++){
    ai[i] = x[4*i-4];
    bi[i] = x[4*i-3];
    ci[i] = x[4*i-2];
    di[i] = x[4*i-1];
  }
  verificaBorda('v');
}
```

```
    imprimeBorda();
    fclose(arq);
    system("convert sai.txt sai.jpg");
    system("rm fig.txt sai.txt");
    return 0;
}
```

APÊNDICE B - Bidimensional

bidim.c - Este software necessita do pacote ImageMagick

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX     576
#define MAX2    700
#define EPS    1e-4

double A[MAX][MAX], x[MAX], b[MAX], a[4][4][3][3],
 y[MAX2][MAX2], alfa, beta;
int ni, ni2, nj, nj2, R[MAX2][MAX2], G[MAX2][MAX2],
    B[MAX2][MAX2], RESP[MAX2][MAX2], tam, tam2, v, w;
FILE *arq;

void anulaA(){
   int i, j;
   for(i=0; i < MAX; i++)
     for(j=0; j < MAX; j++)
       A[i][j] = 0;
}

void zerab(){
   int i;
   for(i=0; i < MAX; i++)
     b[i] = 0;
}

void calculaMatrizes(){
   int i, j, k, n1, n2, n3, n4;
   anulaA();
   zerab();
   k = 0;
   for(j=0; j < nj; j++){
     n1 = 16*j*ni;
     A[k][n1+0] = 1;
     b[k] = y[0][j];
     A[k+1][n1+0] = A[k+1][n1+1] = A[k+1][n1+2] =
     A[k+1][n1+3] = 1;
     b[k+1] = y[0][j+1];
```

```
A[k+2][n1+8] = 2;
A[k+3][n1+9] = 2;
A[k+4][n1+10] = 2;
A[k+5][n1+11] = 2;
k += 6;
for(i=1; i < ni; i++){
  n1 = 16*j*ni+16*(i-1);
  n2 = n1 + 16;
  A[k][n1+3] = A[k][n1+7] = A[k][n1+11] =
  A[k][n1+15] = 1;
  A[k][n2+3] = -1;
  A[k+1][n1+2] = A[k+1][n1+6] = A[k+1][n1+10] =
  A[k+1][n1+14] = 1;
  A[k+1][n2+2] = -1;
  A[k+2][n1+1] = A[k+2][n1+5] = A[k+2][n1+9] =
  A[k+2][n1+13] = 1;
  A[k+2][n2+1] = -1;
  A[k+3][n1+0] = A[k+3][n1+4] = A[k+3][n1+8] =
  A[k+3][n1+12] = 1;
  A[k+3][n2+0] = -1;
  A[k+4][n1+7] = 1;
  A[k+4][n1+11] = 2;
  A[k+4][n1+15] = 3;
  A[k+4][n2+7] = -1;
  A[k+5][n1+6] = 1;
  A[k+5][n1+10] = 2;
  A[k+5][n1+14] = 3;
  A[k+5][n2+6] = -1;
  A[k+6][n1+5] = 1;
  A[k+6][n1+9] = 2;
  A[k+6][n1+13] = 3;
  A[k+6][n2+5] = -1;
  A[k+7][n1+4] = 1;
  A[k+7][n1+8] = 2;
  A[k+7][n1+12] = 3;
  A[k+7][n2+4] = -1;
  A[k+8][n1+11] = 2;
  A[k+8][n1+15] = 6;
  A[k+8][n2+11] = -2;
  A[k+9][n1+10] = 2;
  A[k+9][n1+14] = 6;
  A[k+9][n2+10] = -2;
  A[k+10][n1+9] = 2;
  A[k+10][n1+13] = 6;
```

```
  A[k+10][n2+9] = -2;
  A[k+11][n1+8] = 2;
  A[k+11][n1+12] = 6;
  A[k+11][n2+8] = -2;
  k+=12;
  if(j > 0){
    A[k][n1+12] = -6;
    A[k][n2+12] = 6;
    A[k][n2] = 1.0/alfa;
    b[k] = 1.0/alfa*y[i][j];
    k++;
  }
  else{
    A[k][n2] = 1;
    b[k] = y[i][j];
    k++;
  }
  if(j == nj-1){
    A[k][n2] = A[k][n2+1] = A[k][n2+2] =
    A[k][n2+3] = 1;
    b[k] = y[i][j+1];
    k++;
  }
}
n1 = 16*j*ni+16*(ni-1);
A[k][n1+0] = A[k][n1+4] = A[k][n1+8] =
A[k][n1+12] = 1;
b[k] = y[ni][j];
A[k+1][n1+0] = A[k+1][n1+1] = A[k+1][n1+2] =
A[k+1][n1+3] = A[k+1][n1+4] = A[k+1][n1+5] =
A[k+1][n1+6] = A[k+1][n1+7] = A[k+1][n1+8] =
A[k+1][n1+9] = A[k+1][n1+10] = A[k+1][n1+11] =
A[k+1][n1+12] = A[k+1][n1+13] = A[k+1][n1+14] =
A[k+1][n1+15] = 1;
b[k+1] = y[ni][j+1];
A[k+2][n1+8] = 2;
A[k+2][n1+12] = 6;
A[k+3][n1+9] = 2;
A[k+3][n1+13] = 6;
A[k+4][n1+10] = 2;
A[k+4][n1+14] = 6;
A[k+5][n1+11] = 2;
A[k+5][n1+15] = 6;
k += 6;
```

```
}
for(i=0; i < ni; i++){
  n1 = 16*i;
  A[k][n1+0] = 1;
  b[k] = y[i][0];
  A[k+1][n1+0] = A[k+1][n1+4] = A[k+1][n1+8] =
  A[k+1][n1+12] = 1;
  b[k+1] = y[i+1][0];
  A[k+2][n1+2] = 2;
  A[k+3][n1+6] = 2;
  A[k+4][n1+10] = 2;
  A[k+5][n1+14] = 2;
  k += 6;
  for(j=1; j < nj; j++){
    n1 = 16*(j-1)*ni+16*i;
    n2 = 16*j*ni + 16*i;
    A[k][n1+12] = A[k][n1+13] = A[k][n1+14] =
    A[k][n1+15] = 1;
    A[k][n2+12] = -1;
    A[k+1][n1+8] = A[k+1][n1+9] = A[k+1][n1+10] =
    A[k+1][n1+11] = 1;
    A[k+1][n2+8] = -1;
    A[k+2][n1+4] = A[k+2][n1+5] = A[k+2][n1+6] =
    A[k+2][n1+7] = 1;
    A[k+2][n2+4] = -1;
    A[k+3][n1+0] = A[k+3][n1+1] = A[k+3][n1+2] =
    A[k+3][n1+3] = 1;
    A[k+3][n2+0] = -1;
    A[k+4][n1+13] = 1;
    A[k+4][n1+14] = 2;
    A[k+4][n1+15] = 3;
    A[k+4][n2+13] = -1;
    A[k+5][n1+9] = 1;
    A[k+5][n1+10] = 2;
    A[k+5][n1+11] = 3;
    A[k+5][n2+9] = -1;
    A[k+6][n1+5] = 1;
    A[k+6][n1+6] = 2;
    A[k+6][n1+7] = 3;
    A[k+6][n2+5] = -1;
    A[k+7][n1+1] = 1;
    A[k+7][n1+2] = 2;
    A[k+7][n1+3] = 3;
    A[k+7][n2+1] = -1;
```

```
  A[k+8][n1+14] = 2;
  A[k+8][n1+15] = 6;
  A[k+8][n2+14] = -2;
  A[k+9][n1+10] = 2;
  A[k+9][n1+11] = 6;
  A[k+9][n2+10] = -2;
  A[k+10][n1+6] = 2;
  A[k+10][n1+7] = 6;
  A[k+10][n2+6] = -2;
  A[k+11][n1+2] = 2;
  A[k+11][n1+3] = 6;
  A[k+11][n2+2] = -2;
  k += 12;
  if(i == 0){
    A[k][n2] = 1;
    b[k] = y[i][j];
    k++;
  }
  if(i == ni-1){
    A[k][n2] = A[k][n2+4] = A[k][n2+8] = A[k][n2+12] = 1;
    b[k] = y[i+1][j];
    k++;
  }
}
n1 = 16*(nj-1)*ni + 16*i;
A[k][n1+0] = A[k][n1+1] = A[k][n1+2] =
A[k][n1+3] = 1;
b[k] = y[i][nj];
A[k+1][n1+0] = A[k+1][n1+1] = A[k+1][n1+2] =
A[k+1][n1+3] = A[k+1][n1+4] = A[k+1][n1+5] =
A[k+1][n1+6] = A[k+1][n1+7] = A[k+1][n1+8] =
A[k+1][n1+9] = A[k+1][n1+10] = A[k+1][n1+11] =
A[k+1][n1+12] = A[k+1][n1+13] = A[k+1][n1+14] =
A[k+1][n1+15] = 1;
b[k+1] = y[i+1][nj];
A[k+2][n1+2] = 2;
A[k+2][n1+3] = 6;
A[k+3][n1+6] = 2;
A[k+3][n1+7] = 6;
A[k+4][n1+10] = 2;
A[k+4][n1+11] = 6;
A[k+5][n1+14] = 2;
A[k+5][n1+15] = 6;
k += 6;
```

```
   }
   A[k][5] = 1;
   A[k+1][16*(ni-1)+5] = 1;
   A[k+1][16*(ni-1)+9] = 2;
   A[k+1][16*(ni-1)+13] = 3;
   A[k+2][16*ni*(nj-1)+5] = 1;
   A[k+2][16*ni*(nj-1)+6] = 2;
   A[k+2][16*ni*(nj-1)+7] = 3;
   A[k+3][16*(ni*nj-1)+5] = 1;
   A[k+3][16*(ni*nj-1)+6] = A[k+3][16*(ni*nj-1)+9] = 2;
   A[k+3][16*(ni*nj-1)+7] = A[k+3][16*(ni*nj-1)+13] = 3;
   A[k+3][16*(ni*nj-1)+10] = 4;
   A[k+3][16*(ni*nj-1)+11] = A[k+3][16*(ni*nj-1)+14] = 6;
   A[k+3][16*(ni*nj-1)+15] = 9;
   tam = k+4;
   tam2 = 16*nj*ni;
}

void eliminacaoGauss(){
   int i, j, k;
   double aux;

   for(i=0; i < tam2; i++){
     for(k=i; A[k][i] < EPS &&
                A[k][i] > -EPS && k < tam; k++);
     if(k > i){
       for(j=0; j < tam2; j++){
         aux = A[k][j];
         A[k][j] = A[i][j];
         A[i][j] = aux;
       }
       aux = b[i];
       b[i] = b[k];
       b[k] = aux;
     }
     for(j=i+1; j < tam; j++){
       aux = A[j][i]/A[i][i];
       if(aux > EPS || aux < -EPS){
         for(k=i; k < tam2; k++)
           A[j][k] = A[j][k] - aux*A[i][k];
         b[j] = b[j] - aux*b[i];
       }
     }
   }
```

```
   for(i=tam2-1; i>= 0; i--){
     x[i] = b[i];
     for(j=i+1; j < tam2; j++)
       x[i] = x[i] - A[i][j]*x[j];
     x[i] = x[i]/A[i][i];
   }
}


void leate(char c){
   char aux;
   aux = -1;
   while(aux != c)
     fscanf(arq, "%c", &aux);
}

void recebeDados(){
   int i, j;
   char c;
   arq = fopen("fig.txt", "r");
   if(arq == NULL){
     printf("O arquivo especificado não existe,
                    favor tente novamente!\n");
     exit(-1);
   }
   leate('\n');
   while(!feof(arq) && !ferror(arq)){
     fscanf(arq, "%d", &j);
     c = fgetc(arq);
     if(c < 0)
       break;
     fscanf(arq, "%d", &i);
     leate('(');
     fscanf(arq, "%d", &R[i][j]);
     leate(',');
     fscanf(arq, "%d", &G[i][j]);
     leate(',');
     fscanf(arq, "%d", &B[i][j]);
     leate('\n');
     ni2 = i;
     nj2 = j;
   }
   fclose(arq);
}
```

```
void verificaBorda(){
   int i, j;
   for(i=0; i < ni-1; i++)
     for(j=0; j < nj-1; j++)
       if(sqrt(pow(a[1][0][i][j],2.0)+
                 pow(a[0][1][i][j],2.0)) >= beta)
         RESP[v+i+1][w+j+1] = 1;
}

void imprimeBorda(){
   int i, j;

   arq = fopen("sai.txt", "w");
   fprintf(arq, "# ImageMagick pixel enumeration:
                        %d,%d,255,RGB\n",nj2+1,ni2+1);

   for(i=0; i <= ni2; i++)
     for(j=0; j <= nj2; j++){
       fprintf(arq, "%d,%d: (", j, i);
       if(RESP[i][j] == 1)
         fprintf(arq, "0,0,0) black\n");
       else
         fprintf(arq, "255,255,255) white\n");
     }
}

void transformaImagemTexto(char *nome){
   int i;
   char comando[100];
   comando[0] = 'c';
   comando[1] = 'o';
   comando[2] = 'n';
   comando[3] = 'v';
   comando[4] = 'e';
   comando[5] = 'r';
   comando[6] = 't';
   comando[7] = ' ';
   for(i=0; nome[i] != 0; i++)
     comando[i+8] = nome[i];
   comando[i+8] = ' ';
   comando[i+9] = 'f';
   comando[i+10] = 'i';
   comando[i+11] = 'g';
```

```
   comando[i+12] = '.';
   comando[i+13] = 't';
   comando[i+14] = 'x';
   comando[i+15] = 't';
   comando[i+16] = 0;
   system(comando);
}

int main(int argc, char *argv[]){
   int i, j, var1, var2;
   char *nome;
   nome = argv[1];
   transformaImagemTexto(nome);
   recebeDados();
   alfa = atof(argv[2]);
   beta = atof(argv[3]);
   for(i=0; i <= ni2; i++)
     for(j=0; j <= nj2; j++)
       RESP[i][j] = 0;
   ni = nj = 4;
   for(v=0; v < ni2; v+=3){
     if(v+4 > ni2)
       ni = ni2-v;
     for(w=0; w < nj2; w+=3){
       if(w+4 > nj2)
         nj = nj2-w;
       for(i=0; i <= ni; i++)
         for(j=0; j <= nj; j++)
           y[i][j] = R[v+i][w+j];
       calculaMatrizes();
       eliminacaoGauss();
       for(var1=0; var1 < ni-1; var1++)
         for(var2=0; var2 < nj-1; var2++)
           for(i=0; i < 4 ; i++)
             for(j=0; j < 4; j++)
               a[i][j][var1][var2] = x[16*ni*(var2+1)+
                                       16*(var1+1)+4*i+j];
       verificaBorda();
       for(i=0; i <= ni; i++)
         for(j=0; j <= nj; j++)
           y[i][j] = G[v+i][w+j];
       calculaMatrizes();
       eliminacaoGauss();
       for(var1=0; var1 < ni-1; var1++)
```

```
          for(var2=0; var2 < nj-1; var2++)
            for(i=0; i < 4 ; i++)
              for(j=0; j < 4; j++)
                a[i][j][var1][var2] = x[16*ni*(var2+1)+
                                        16*(var1+1)+4*i+j];
        verificaBorda();
        for(i=0; i <= ni; i++)
          for(j=0; j <= nj; j++)
            y[i][j] = B[v+i][w+j];
        calculaMatrizes();
        eliminacaoGauss();
        for(var1=0; var1 < ni-1; var1++)
          for(var2=0; var2 < nj-1; var2++)
            for(i=0; i < 4 ; i++)
              for(j=0; j < 4; j++)
                a[i][j][var1][var2] = x[16*ni*(var2+1)+
                                        16*(var1+1)+4*i+j];
        verificaBorda();
        nj = 4;
      }
      ni = 4;
    }
    imprimeBorda();
    fclose(arq);
    system("convert sai.txt sai.jpg");
    system("rm fig.txt sai.txt");
    return 0;
}
```