Universidade de São Paulo
Instituto de Matemática e Estatística

Operadores Conexos Aplicados à Localização de Texto em Imagens

MAC499 - Trabalho de Formatura Supervisionado

Aluno: *Alexandre Morimitsu*

Orientador: *Prof. Ronaldo Fumio Hashimoto*

- São Paulo, 30 de novembro de 2011-

SUMÁRIO

Parte I

PARTE TÉCNICA

1 INTRODUÇÃO

Extração de informações textuais (*TIE - text information extraction)* de imagens e vídeos é um poderosa ferramenta na área de Visão Computacional. Diversos tipos de aplicações sobre o tema podem ser desenvolvidas, tais como extração de informações sobre placas de carro ou extração de endereço em cartões postais.

Tais aplicações, entretanto, têm fortes suposições sobre o conteúdo de tais vídeos ou imagens, uma vez que endereços de cartões postais ou placas de carro geralmente estão em uma posição específica do objeto em questão. Assim sendo, pode-se assumir que tais mídias visuais avaliadas por essas aplicações seguem algum tipo de padrão, que pode ser usado para obter resultado mais robustos no que se refere ao problema da extração de texto.

Porém, isso não é verdade para imagens ou vídeos em geral. Em fotos, muitas vezes o enfoque dado não é em uma região de texto, mas sim em uma cena. Ainda assim, textos presentes na imagem podem guardar importantes informações relacionadas ao contexto semântico em que foi tirada, um dado que poderia ser útil, por exemplo, para a indexação de mídias digitais.

Alves e Hashimoto[2] demonstraram ser possível desenvolver uma aplicação que detecte regiões de texto com um bom grau de desempenho. Com base em tal aplicação, o objetivo desta monografia é trabalhar em certas partes do código existente, baseado em novas literaturas da área, para verificar os impactos da introdução das novas técnicas ao algoritmo de Localização de Texto.

2 CONCEITOS E TECNOLOGIAS ESTUDADAS

2.1 DEFINIÇÃO DE IMAGEM

Gomes e Velho [5] definem uma *imagem* como um *suporte retangular* e uma *função imagem*.

O suporte retangular é representado por um subconjunto retangular $\mathbb{E} \subset \mathbb{Z}^2$. A função imagem é um função $f : \mathbb{E} \to \mathbb{K}$, sendo $\mathbb{K} \in \{0, 2^n - 1\}$ e n o número de bits da *profundidade* da imagem. Um elemento x de $\mathbb{E}$ é denominado *pixel*, enquanto que o valor $f(x) \in \{0, 2^n - 1\}$ é o *nível de cinza* do *pixel* x.

2.1.1 *Imagem binária*

Uma *imagem binária* é uma imagem com 1 bit de profundidade, isto é, sua função imagem satisfaz $f : \mathbb{E} \to \{0, 1\}$.

2.1.2 *Imagem em níveis de cinza*

Uma *imagem em níveis de cinza* é uma imagem cuja profundidade tem mais de 1 bit. No caso específico desta monografia, as imagens de cinza consideradas terão profundidade de 8 bits, ou seja, cujos níveis de cinza de cada *pixel* variam de 0 a $2^8 - 1 = 255$.

2.1.3 *Imagem multibandas*

Imagens multibandas recebem, para cada *pixel*, mais de um valor. A função imagem satisfaz, portanto, $f : \mathbb{E} \to \mathbb{K}^c$, onde c é o *número de bandas* da imagem. Imagens multibandas geralmente são usadas para representar imagens coloridas, tais como as imagens RGB, em que cada banda representa, respectivamente, a intensidade de vermelho, verde e azul da imagem.

2.2 DEFINIÇÕES SOBRE IMAGENS

2.2.1 *Histograma*

O *histograma* de uma imagem dada é uma representação que mostra, para cada nível de cinza possível, a frequência de *pixels* da imagem com tal tonalidade. Mais formalmente, podemos definí-la como uma função discreta $h(r_k) = n_k$, onde r_k é o k-ésimo valor de intensidade e n_k é o número de *pixels* na imagem com intensidade r_k.

2.2.2 *Foreground e background*

Em imagens binárias, definimos que um *pixel* p pertence ao *foreground* da imagem se ele satisfizer $f(p) = 1$ e que ele pertence ao *background* se satisfizer $f(p) = 0$.

2.2.3 *Vizinhança entre pixels*

Seja $p = (x, y)$ um *pixel* da imagem. Como $p \in \mathbb{E} \subset \mathbb{Z}^2$, existem *pixels* adjacentes a p horizontalmente ($(x-1, y)$ e $(x+1, y)$), verticalmente ($(x, y-1)$ e $(x, y+1)$ e nas diagonais ($(x-1, y-1)$, $(x-1, y+1)$, $(x+1, y-1)$ e $(x+1, y+1)$).

Sob *4-conexidade*, consideramos que dois *pixels* são *vizinhos* se eles são adjacentes entre si, horizontalmente ou verticalmente. Sob *8-conexidade*, consideramos, além dos casos anteriores, que tais *pixels* também são vizinhos se forem diagonalmente adjacentes.

2.2.4 *Conexidade, caminhos e componentes conexas*

Seja V um conjunto de intensidades de niveis de cinza para definir os conceitos relacionados a conexidade.

Para imagens binárias, definimos $V = 1$. Para imagens de cinza, qualquer subconjunto de valores de intensidade é válido.

Dados dois *pixels* p e p' de uma imagem, definimos que eles são *conexos* se eles são vizinhos e suas intensidades pertencem ao conjunto V.

Um *caminho* em uma imagem de um *pixel* $p = (x_0, y_0)$ a um *pixel* $p' = (x_n, y_n)$ é uma sequência de *pixels* distintos e adjacentes de coordenadas $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, onde cada *pixel* (x_i, y_i) é conexo ao *pixel* $(x_{i+1}, y_{i+1}), 0 \leqslant i \leqslant n-1$, onde n é o *tamanho* do caminho.

Um conjunto maximal de *pixels* conexos é definido como uma *componente conexa*(CC). Assim sendo, para todo par (p_i, p_j) de *pixels* de uma componente conexa, existe um caminho de p_i a p_j.

2.2.5 *Atributos*

Para cada componente conexa define-se vários *atributos*. Exemplos de atributos são a *área* (o número de *pixels* de foreground da componente) e a *altura* (a maior diferença vertical dentre dois *pixels* da componente).

2.3 TRANSFORMAÇÃO EM IMAGENS

Uma *transformação* Ψ é uma operação que transforma uma imagem em outra, no qual o contradomínio da imagem de saída é igual ao domínio da imagem de entrada.

2.3.1 *Limiarização*

*Limiarização (*T*)* é uma transformação de *binarização*, isto é, transforma uma dada imagem em uma imagem binária. Para tanto, a limiarização se baseia no nível de cinza de cada *pixel* da imagem de entrada. Se o nível de cinza pertencer a um certo intervalo $[t_1, t_2]$, o *pixel* é mapeado com um *pixel* do *foreground* na imagem da saída e, caso contrário, é mapeado para um *pixel* do *background*. Considerando $f(x)$ como a intensidade do *pixel* x, podemos definir limiarização da seguinte forma:

$$[T_{t_1,t_2}](x) = \begin{cases} 1 & \text{se } t_1 \leqslant f(x) \leqslant t_2 \\ 0 & \text{c. c.} \end{cases}$$

No nosso caso, consideraremos sempre que t_2 é o maior nível de cinza possível, ou seja, a limiarização só depende do valor t_1. Agora, definimos a *limiarização pelo limiar* t_1 como:

$$[T_{t_1}](x) = \begin{cases} 1 & \text{se } t_1 \leqslant f(x) \\ 0 & \text{c. c.} \end{cases}$$

2.3.2 *Método de Otsu*

O *método de Otsu* é um processo de limiarização que visa separar uma imagem de cinza em duas classe: o objeto (*foreground*) e o fundo (*background*). O limiar de binarização é calculado de forma exaustiva e se baseia no histograma da imagem: para cada limiar, o histograma é dividido em duas classes, o histograma dos níveis de cinza menor que o limiar e o histograma dos níveis de cinza restantes. A variância de cada classe é calculada e o limiar escolhido é aquele cuja diferença de variâncias é mínimo.

2.4 TRANSFORMAÇÕES PROJETIVAS

2.4.1 *Espaços de coordenadas*

Um *pixel* pode ser representado por *coordenadas euclideanas* através de um par (x, y), onde x denota a coordenada horizontal e y a coordenada vertical.

Em coordenadas euclideanas, o processo de translação não pode ser calculado através de uma transformação linear, o que não permite que tal transformação seja implementada como uma multiplicação de matrizes.

Coordenadas afins contornam tal inconveniente. Ainda assim, transformações no espaço afim mapeiam, por construção, retas paralelas em retas paralelas.

Para mapear retas paralelas em retas não-paralelas, é necessário utilizar-se do *espaço projetivo*. Tal operação nos será útil para permitir a retificação de regiões de texto, que mapeiam retas que se direcionam a um ponto de fuga para retas paralelas.

Coordenadas do espaço projetivo são chamadas de *coordenadas homogêneas*. Coordenadas homogêneas em um plano são representadas por uma tripla (X, Y, Z).

Um *pixel* em coordenadas euclideanas (x, y) é geralmente escrito como a tripla $(X, Y, 1)$ em coordenadas homogêneas, onde $X = x$, $Y = y$. No mapeamento inverso, um pixel (X, Y, Z) é mapeado para um pixel (x, y) em coordenadas euclideanas calculando $x = \frac{X}{Z}$ e $y = \frac{Y}{Z}$.

2.4.2 *Rotação*

A rotação de um ângulo α em torno da origem de uma imagem pode ser calculada multiplicando-se a *matriz de rotação* por cada pixel da imagem original.

A matriz de rotação para coordenadas euclideanas é definida como:

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

2.4.3 *Cisalhamento*

Cisalhamento é uma transformação que mantém um segmento de reta da imagem fixo enquanto translada os outros pontos da imagem paralelamente a tal segmento.

Dado o coeficiente angular m do cisalhamento, sua matriz é dada por:

$$\begin{pmatrix} 1 & 0 \\ m & 1 \end{pmatrix}$$

2.4.4 *Homografia*

Homografia é uma transformação do espaço projetivo que mapeia linhas retas em linhas retas (não necessariamente paralelas).

Neste trabalho, a homografia será usada na retificação, realizando o mapeamento do plano da região de texto para o plano da imagem.

2.5 OPERADORES MORFOLÓGICOS

2.5.1 *Abertura por atributos*

A abertura por atributos em imagens binárias é um operador morfológico que, para cada componente conexa da imagem, decide se

tal CC deve ser removida da imagem baseado em seu atributo. Para tanto, a abertura por atributo recebe um valor limiar e remove todas as componentes conexas cujo atributo é menor que o limiar.

O conceito de abertura por atributo pode ser estendido a imagens de níveis de cinza. Neste caso, realizamos uma binarização para cada nível de cinza possível na imagem original. Em cada imagem binária criada é aplicada a abertura por atributos. A soma de todas as imagens binárias resultantes gerará a imagem em nível de cinza resultante.

Definiremos a abertura por atributo de limiar i como γ_i.

2.5.2 *Ultimate Opening por atributos*

Ultimate Opening (UO) é um operador morfológico que visa realçar áreas de alto contraste em imagens. Sua computação é feita através de aberturas sucessivas, onde é selecionado, para cada *pixel*, o resíduo máximo de duas aberturas sucessivas. O resultado é armazenado em uma imagem v, a qual chamaremos de *imagem da transformação*. O tamanho da abertura que gerou tal resíduo também é armazenado em outra imagem, q, definida como a *imagem residual*. Usando as definições vistas anteriormente, o *Ultimate Opening* pode ser definido como:

$$v = \sup(r_i) = \sup(\gamma_i - \gamma_{i-1})$$
$$q = \max(i) + 1; r_i = v(\neq 0)$$

Neste trabalho, as aberturas utilizadas se baseiam no atributo de altura. Assim sendo, γ_i representa a abertura pelo atributo altura de limiar i.

2.5.3 *Ultimate Opening por atributos com transições graduais*

Ultimate Opening com transições graduais [6] é uma técnica para aumentar o contraste de regiões de texto borradas de uma imagem. Geralmente, regiões borradas consistem em regiões de transição do fundo para o objeto da imagem, com pouca variação entre os níveis de cinza. Chamaremos tais regiões de *transições graduais*.

O efeito de ter uma transição gradual no texto é a geração de várias componentes conexas de atributos e níveis de cinza muito próximos. A computação da diferença de duas aberturas consecutivas resultará em um valor baixo, devido à proximidade dos valores de níveis de cinza entre as componentes, subestimando o real contraste da região textual.

Porém, o contraste real do texto deveria ser a diferença de nível de cinza do tom do texto para o tom do fundo. Isso pode ser obtido se somarmos o contraste dos *pixels* da região de transição. Assim sendo, o *Ultimate Opening* com transições graduais consiste no acúmulo de contrastes quando a diferença de atributo entre duas componentes conexas for menor que um dado limiar.

Se o limiar passado for o (zero), teremos a definição do *Ultimate Opening* da seção 2.5.2.

2.6 MAXTREE

Maxtree é uma estrutura de dados que permite representar uma imagem através de uma árvore. Nesta estrutura, cada nó está associado a uma componente conexa da imagem. Para fins de identificação, cada nó possui um identificador único, além de estar associado a um nível de cinza, que é definido como o menor nível de cinza pertencente a aquela componente conexa (esse nível é equivalente ao maior limiar cuja limiarização gera aquela componente conexa). Se estivermos interessados nos atributos da imagem original, como é o caso neste trabalho para o atributo de altura, cada nó também armazena o valor do atributo da respectiva componente conexa.

A raiz da árvore é um nó que representa o *background* da imagem, e pode ser visto como a limiarização da imagem com limiar valendo m, onde m é o menor nível de cinza da imagem.

A hierarquia da árvore é definida da seguinte forma: dados dois nós n_1, n_2 da *Maxtree*, n_2 é filho de n_1 se a componente conexa associada a n_2 estiver contida (não igual) na componente conexa associada a n_1.

2.6.1 *Descrição do algoritmo de construção da Maxtree*

Com base nas definições acima, a *Maxtree* de uma imagem pode ser construída aplicando-se limiarizações para cada nível de cinza e comparando-se a imagem binária obtida com a limiarização com o nível anterior, comparando se houve ou não alteração nas componentes conexas da imagem.

Porém, tal abordagem não é tão eficiente: para cada nível de cinza, é necessário gerar sua imagem limiarizada e compará-la com a imagem anterior, ou seja, é preciso percorrer toda a imagem pelo menos uma vez para cada nível de cinza.

Salembier *et al*[9] propopuseram uma abordagem muito mais eficiente, baseada em uma técnica de *inundação(flooding)*, onde cada *pixel* da imagem é processado apenas uma vez.

O processo de inundação começa em um *pixel* da imagem com nível de cinza mínimo e vai se expandindo aos seus vizinhos.

A inundação de um *pixel* p consiste em analisar todos os seus vizinhos p' e criar os nós correspondentes a cada uma das CCs geradas pelos *pixels* p' na árvore. Este processo é repetido para todos os *pixels* da imagem em uma ordem definida pelo nível de cinza dos *pixels* já visitados: sempre que p, que está sendo inundado, encontra um vizinho p' com $f(p') > f(p)$, o algoritmo chama recursivamente o processo de inundação para p'.

Para garantir que um *pixel* não seja inundado mais de uma vez, ele é marcado e colocado em uma fila de prioridades quando visitado pela primeira vez. A prioridade da fila é o nível de cinza do *pixel*. A maior prioridade é dado ao *pixel* de maior nível de cinza. Um *pixel* sai da fila

de prioridades quando todos os seus vizinhos foram marcados e sua prioridade for a maior dentre todos da fila.

Quando a inundação de um *pixel* p′ termina (isto é, quando p′ sai da fila de prioridades), ao invés de retornar para o *pixel* p que o chamou, o algoritmo chama a função de inundação para um *pixel* na fila de maior prioridade. O algoritmo termina quando toda a imagem for inundada.

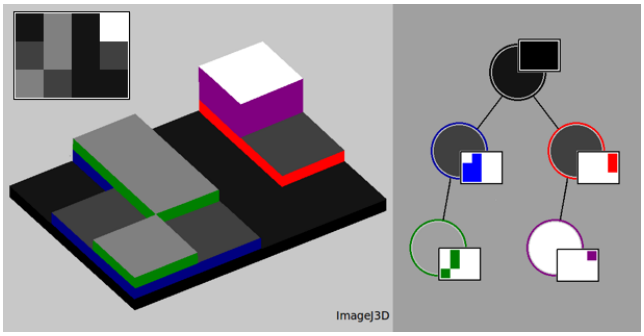


Figura 1: Exemplo de uma imagem, sua visualição em superfície, sua *Maxtree* e suas respectivas componentes conexas, cada uma representada por uma cor diferente

Segue descrição do algoritmo em pseudocódigo.

```
Node -> uma estrutura que representa um nó e composta por dois campos:
        - idFilho: id do primeiro nó filho;
        - idIrmao: indica o próximo nó filho do nó pai.

Maxtree maxtree: uma estrutura que armazena os Nodes.
int idNo: id do próximo nó.
Stack<pixel>[256] s: um vetor de pilhas de pixels,
                     com uma pilha para cada nível de cinza.
int[w][h] imagemL: imagem com labels: indica, para cada pixel
                   p da imagem, o id de seu respectivo nó.
int[256] ultimoNo: ultimoNo[i] indica o id do último nó criado
                   cujo nível de cinza vale i
int[] nivelNo: nivelNo[i] indica o nível de cinza do nó i
int[] atributos: atributos[i] indica o valor do atributo
                 do nó i

void constroiMaxtree(imagem){
   inicializa variaveis acima;

   pixel p = pixel de menor nível de cinza da imagem;
   int nivelAtual = nível de cinza do pixel p;
   s[nivelAtual].push(p);
   ultimoNo[nivelAtual] = 0;
   imagemL[p.x][p.y] = 0;
   nivelNo[0] = nivelAtual;
   atributos[0] = inicializa atributos;
   idNo = 1;
   flood(imagem, 0); //ignora o valor de retorno
}
```

```
int flood(imagem, int idNoAtual){
   nivelAtual = nivelNo[idNoAtual];
   while(s[nivelAtual].notEmpty()){
      pixel p = s[nivelAtual].pop();
      int nivelAtual = imagem[p.x][p.y];
      for_each(pixel q vizinho de p){
         int nivelVizinho = imagem[q.x][q.y];
         if(imagemL[q.x][q.y] == -1){
            if(nivelVizinho > nivelAtual){
               labelMaior(nivelAtual, nivelVizinho);
               ultimoNo[nivelVizinho] = idNo;
               imagemL[q.x][q.y] = idNo;
               atributos[idNo] = inicializa atributos;
               idNo++;
            }
            else if(ultimoNo[nivelVizinho] == -1){
              labelMenor(nivelVizinho);
              ultimoNo[nivelVizinho] = idNo;
              imagemL[q.x][q.y] = idNo;
              atributos[idNo] = inicializa atributos;
              idNo++;
            }
            else{
               int indice = ultimoNo[nivelVizinho];
               imagemL[q.x][q.y] = indice;
               atributos[idNo] = atualiza atributos;
            }
            s[nivelVizinho].push(q);
            if(nivelVizinho > nivelAtual){
               int k = nivelVizinho;
               int noFlood = ultimoNo[k];
               while(k != nivelAtual){
                  k = flood(imagem, noFlood);
                  atributos[noFlood] = atualiza atributos;
                  noFlood = ultimoNo[k];
               }
            }
         }
      }
   }
   k = nivelAtual - 1;
   while(k >= 0 && ultimoNo[k] == -1)
      k--;
   ultimoNo[k] = -1;
   return k;
}
```

```
void labelMaior(int nivelPai, nivelFilho){
   Nó node;
   nivel[idNo] = nivelFilho;
   node.idIrmao = maxtree.getNode(ultimoNo[nivelPai]).idFilho;
   node.idFilho = -1;
   maxtree.setNode(node, idNo);
   maxtree.getNode(ultimoNo[nivelPai]).idFilho = nivelFilho;
}

void labelMenor(int nivelPai, nivelFilho){
   Nó node;
   nivel[idNo] = nivelFilho;
   int l = nivelPai(nivelFilho);
   node.idIrmao = maxtree.getNode(maxtree.getNode(ultimoNo[l]).idFilho).idIrma
   node.idFilho = maxtree.getNode(ultimoNo[l]).idFilho;
   maxtree.setNode(node, idNo);
   maxtree.getNode(maxtree.getNode(ultimoNo[l]).idFilho).idIrmao = -1;
   maxtree.getNode(ultimoNo[l]).idFilho = nivelFilho;
}

int nivelPai(int nivelFilho){
   for(int i = nivelFilho-1; ultimoNo[i] == -1; i--);
   return i;
}
```

2.6.2 *Computação do Ultimate Opening na Maxtree*

Fabrizio e Marcotegui [3] explicitam que a computação do *Ultimate Opening* pode ser feito eficientemente usando *Maxtree*.

O algoritmo começa no nó raiz, onde não há contraste, pois se trata do *background* da imagem. Para cada nó da árvore, o contraste é calculado como a diferença de nível de cinza entre o nó pai e o nó filho, se o valor dos atributos for diferente (se forem iguais, ambas as CCs relativas a cada nó são removidas pela abertura de mesmo tamanho). Se forem iguais, computa-se a diferença entre o nível de cinza do nó filho e o primeiro nó ancestral cujo atributo é diferente.

Os valores das diferenças são repassados aos nós filhos, que computam sua própria diferença e comparam com os valores repassados. O maior valor entre os dois é tomado e o processo é repetido para todos os nós.

O algoritmo citado é eficiente porque o processo descrito acima pode ser feito com uma busca em profundidade (ou em largura) na *Maxtree*.

2.6.3 *Retificação de Texto*

A técnica implementada é baseada na publicação de Myers *et al*[7] e é explicada a seguir.

O método considera que a perspectiva da região de texto é tal que não há ponto de fuga do texto na vertical. Tal hipótese é baseada na dificuldade de estimar tal ponto de fuga por falta de características verticais relevantes em regiões de texto para que este cálculo seja feito de forma confiável.

Assim sendo, para a retificação de texto, considera-se apenas dois parâmetros: as coordenadas euclideanas (x, y) do ponto de fuga horizontal do texto e o ângulo de cisalhamento do texto com relação ao eixo vertical.

Para o cálculo do ponto de fuga, é necessário encontrar duas linhas na região de texto que seguem o alinhamento dos caracteres, as quais chamaremos de *linha de topo* e *linha de base*.

Para o cálculo da linha de topo, a seguinte técnica é aplicada: dada uma região candidata a texto, são aplicadas várias rotações no intervalo de $-20°$ a $20°$ com variações de $0,5°$. Para cada rotação, e para cada coluna da imagem, toma-se as coordenadas y mínimas (se tal coordenada não existir, ignora-se tal coluna) cujo *pixel* pertence ao *foreground* da imagem.

Para cada imagem rotacionada, é projetado um histograma das coordenadas tomadas. Dentre todos os histogramas, procura-se a coordenada que gerou o maior pico. O ângulo que gerou tal pico deve representar o ângulo em que a linha de topo se encontra o mais na horizontal possível e a coordenada deve representar em qual coordenada y da imagem tal linha se encontra.

O cálculo da linha de base é análogo, mas para coordenadas y máximas. Com base nos ângulos e nas coordenadas encontradas, é possível traçar as duas retas na imagem original e encontrar o ponto de fuga.

Para o cálculo do ângulo de cisalhamento, é necessário primeiro rotacionar o texto para que a linha de base fique na horizontal. Feito isso, transformações de cisalhamento são aplicadas, num intervalo variando de $-45°$ a $45°$, com passos de $3°$.

Para cada cisalhamento, é gerado o histograma da imagem. Para cada histograma, é calculada a soma dos quadrados das frequências. O ângulo α que gerar a maior soma é considerado o ângulo de cisalhamento do texto.

Tendo os parâmetros x, y e α, pode-se obter o texto retificado aplicando a seguinte homografia na imagem original:

$$\begin{pmatrix} -(x-y)(x-\alpha y+\alpha-1) & \alpha(x-y)(x-\alpha y+\alpha-1) & 0 \\ y(\alpha-1)(x-\alpha y+\alpha-1) & x(\alpha-1)(x-\alpha y+\alpha-1) & 0 \\ -(\alpha-1)(x-y) & \alpha(\alpha-1)(x-y) & 1 \end{pmatrix}$$

Ao final do processo, teremos os *pixels* em coordenadas homogêneas e será necessário mapeá-los em coordenadas euclideanas. Além disso, como nem todos os *pixels* têm mapeamento na matriz de homografia, uma interpolação pode ser necessária para preencher áreas não mapeadas da imagem de saída.

2.7 DESCRIÇÃO DO ALGORITMO DE LOCALIZAÇÃO

A descrição detalhada dos passos e das definições utilizadas no algoritmo podem ser encontradas na publicação original de Alves e Hashimoto[2].

Explicando resumidamente, o processo de localização de texto original consiste das seguintes etapas:

- Converta a imagem I_0 para uma imagem em níveis de cinza I_1;
- Calcule o UO na imagem I_1 para obter a imagem de transformação ν;
- Binarize a imagem ν usando o método de Otsu, obtendo uma imagem I_b;
- Extraia as CCs da imagem;
- Tente unir as CCs para definir regiões R_c candidatas a texto;
- Para cada R_c obtida, classificar em região de texto ou não-texto.

3

ATIVIDADES REALIZADAS E PRODUTOS DESENVOLVIDOS

3.1 METODOLOGIA

O desenvolvimento do código é feito puramente em Java e foi fortemente baseado no algoritmo já existente desenvolvido em [2]. Sobre o algoritmo existente foram aplicados conceitos mencionados no capítulo anterior.

Como a escrita do código foi uma extensão do que já havia sido feito no trabalho de Alves e Hashimoto[2], reuniões com os autores foram realizadas, para eventuais discussões de implementação ou dúvidas sobre conceitos a serem aplicados.

3.2 PRODUTOS DESENVOLVIDOS

- Binarização de imagens

  Uma das etapas do algoritmo de Localização de Texto trata de binarização. Usando a base já pronta do código, unindo a estrutura de dados *Maxtree* desenvolvida e realizando alguns ajustes nos parâmetros do código, um algoritmo de binarização de imagens foi desenvolvido e submetido à uma competição de binarização de documentos, o DIBCO[1].

  Um executável Binarizer.jar foi gerado a partir do código para testar a binarização em diferentes imagens. Ele pode ser executado no Terminal, através do comando:

  ```
  java -jar Binarizer.jar <diretório da imagem de entrada>
  ```

  Como a aplicação desenvolvida era voltada à binarização de documentos, supõe-se um fundo claro com letras escuras. Ao testar numa imagem com textos claros, o fundo e objeto ficarão invertidos.

- Ultimate Opening com Maxtree

  O algoritmo descrito na seção 2.6.1 foi implementado para análise de desempenho, comparando o tempo de processamento desta técnica com o *Ultimate Opening* sem a *Maxtree*. Os resultados são descritos no próximo capítulo.

- Transições graduais

  Tendo implementado o operador *Ultimate Opening* com *Maxtree*, o próximo passo foi adicionar a técnica de transições graduais ao algoritmo, conceito explicado na seção 2.5.3.

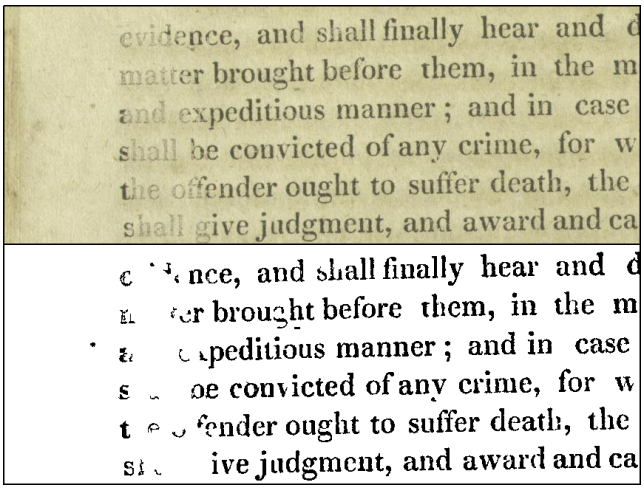


Figura 2: Exemplo de uma imagem e a sua binarização pelo método desenvolvido

Figura 3: Exemplo de uma imagem e a computação do *Ultimate Opening*, realçando áreas com bastante constraste na imagem

Para mostrar as imagens geradas pelo operador UO foi desenvolvida a aplicação UO.jar. Seu uso é dado por:

```
java -jar UO.jar <diretório da imagem de entrada> [modo]
[transição gradual] [fator]
```

Os elementos entre colchetes são opcionais.

`[modo]` se refere ao modo de execução. `[modo] = 0` realça regiões claras da imagem e `[modo] = 1` realça áreas escuras da imagem. Para tanto, é feita uma inversão da imagem antes da aplicação do UO.

`[transição gradual]` deve ser um valor inteiro e se refere ao limiar visto na seção 2.5.3.

Por fim, `[fator]` é um valor real e representa o maior atributo de uma CC que deve ser considerada. CCs com atributo maior que `[fator]` $\times$ altura da imagem são descartadas. Portanto, para imagens com textos grandes, é importante que o fator seja mais alto para que sua região não seja descartada.

Os valores *default* quando os parâmetros são omitidos são `[transição gradual] = 0` (UO sem considerar transições graduais), `[fator] = 0.33` e `[modo] = 0`.

Figura 4: Exemplo de uma imagem, computação do UO sem considerar transições graduais e UO com transições graduais com limiar 5

- Retificação de texto

  O próximo passo do trabalho consistiu em analisar e incorporar mais um algoritmo ao localizador de textos: retificação de textos para o plano da imagem, baseado no método explicado em 2.6.3. Tal etapa é introduzida imediatamente antes da classificação das regiões de texto.

  Para testar as retificações, foi desenvolvido o executável Rectification.jar. Sua execução é feita da seguinte forma:

  ```
  java -jar Rectification.jar <diretório da imagem de entrada>
  ```

  Como este código trata apenas da etapa de retificação, não é realizada nenhuma etapa anterior de detecção de regiões candidatas a texto. Portanto, a imagem a ser passada deve conter predominantemente uma região de texto.

  O método implementado tenta retificar a região de texto supondo que há uma única região de texto na imagem, voltada sempre para o mesmo ponto de fuga e com o mesmo ângulo de cisalhamento. Além disso, textos com ponto de fuga na vertical não são retificados, devido à forma com que o que algoritmo foi desenvolvido.

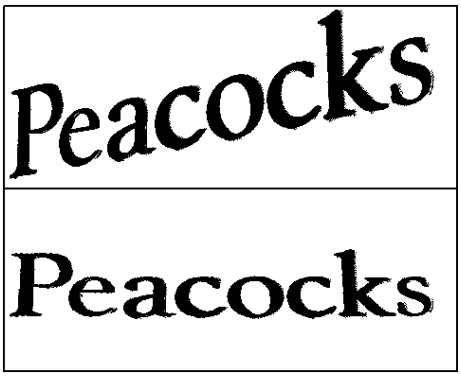


Figura 5: Exemplo de uma imagem de entrada e sua retificação

- Localização de texto

  Por fim, os algoritmos explicados acima foram integrados ao localizador de texto, gerando um último arquivo Text.jar, cuja execução é feita através de:

  `java -jar Text.jar <diretório da imagem de entrada> [Tamanho do elemento estruturante] [fator]`

  O algoritmo pode demorar bastante tempo dependendo do tamanho da imagem. Assim, imagens muito grandes são redimensionadas para tamanhos menores.

  Novamente, `[Tamanho do elemento estruturante]` e `[fator]` são opcionais. Por padrão, o primeiro é calculado internamente e o segundo é inicializado com 0,33.

  `[Tamanho do elemento estruturante]` está fora do escopo desta monografia. Basicamente, ele trata do tamanho do elemento que é usado para combinar os caracteres em uma única região de texto. Estimar um bom valor para `[Tamanho do elemento estruturante]` é algo complicado e que varia de acordo com a região de texto trabalhada. Se o valor passado for zero, considera-se o valor padrão.

  `[fator]` é o mesmo do algoritmo do *Ultimate Opening*.

Figura 6: Exemplo de saída do algoritmo de localização

4

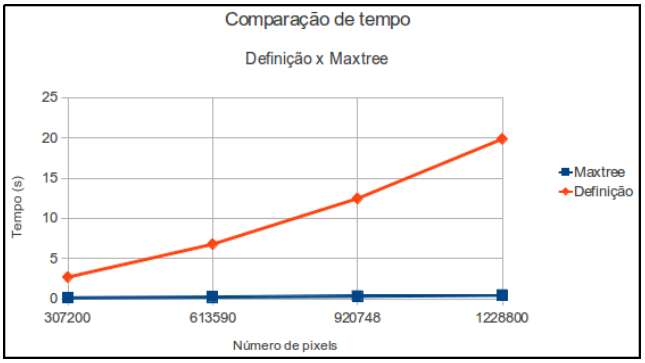
RESULTADOS E ANÁLISES

4.1 USO DA MAXTREE NA COMPUTAÇÃO DO ULTIMATE OPENING

Comparando-se a implementação do cálculo do *Ultimate Opening* baseado na definição da seção 2.5.2 com a implementação baseada em *Maxtree* apresentada em 2.6.1, é possível observar que a segunda é bem mais eficiente no que se refere ao tempo de processamento:

Número total de Pixels (resolução)	Tempo de processamento (s)	
	Definição	*Maxtree*
307200 (640x480)	2,678	0,103
613590 (905x678)	6,775	0,199
920748 (1108x831)	12,456	0,301
1228800 (1280x960)	19,869	0,431

Tabela 1: Comparação de tempo do UO sem e com *Maxtree*



Os testes foram computados em um *notebook* com processador i3 de 2.13GHz e 4GB de RAM.

Para realizar os testes, foram tomadas 10 imagens de cada tamanho especificado. Os tamanhos foram definidos de forma a manter a diferença entre o número de pixels de duas imagens aproximadamente constante. Cada algoritmo foi executado sobre cada imagem 10 vezes, tomando-se o tempo médio das 100 execuções para cada resolução.

4.2 EFEITOS DAS TRANSIÇÃO GRADUAIS

Uma forma de se analisar o efeito da consideração de transições graduais em imagens é verificando o quão boa é a binarização gerada por tal técnica. Para tanto, será utilizado o conjunto de imagens e as

métricas utilizadas na competição de binarização de documentos, o DIBCO.

A métrica utilizada será *F-Measure*, que é definido como:

$$\begin{aligned} \mathrm{FMeasure} &= \frac{2 \mathrm{x} \mathrm{Recall} \mathrm{x} \mathrm{Precision}}{\mathrm{Recall} + \mathrm{Precision}} \\ \mathrm{Recall} &= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \\ \mathrm{Precision} &= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \end{aligned}$$

onde TP denota a porcentagem de *pixels* positivos verdadeiros, FP denota a porcentagem de falsos positivos e FN a porcentagem de falsos negativos. Quanto mais próximos de 100%, melhores os resultados.

Para o teste, será analisado o efeito da alteração do limiar de transição gradual na binarização de imagens do DIBCO de 2009[4]. São 10 imagens, onde cinco são manuscritas (h) e cinco são impressas (p). Será levado em conta o *F-Measure* de cada imagem binarizada.

Imagem	Limiar da transição gradual				
	0	1	2	5	10
h1	**89.178**	78.855	79.861	79.175	79.964
h2	**89.350**	74.365	70.203	63.308	56.691
h3	82.355	82.110	79.429	**82.564**	35.743
h4	83.992	77.292	82.903	**84.263**	85.555
h5	76.673	67.638	73.630	75.283	**79.354**
p1	89.539	88.454	90.611	**90.944**	59.716
p2	94.538	**95.383**	95.044	94.727	94.324
p3	97.354	97.452	**97.509**	97.484	97.461
p4	**89.460**	88.568	88.282	83.800	83.128
p5	88.195	88.259	**88.849**	88.846	88.607
Média	**88.063**	83.838	84.632	84.039	76.054

Tabela 2: Valores de F-Measure (em %) para cada limiar

Os melhores resultados para cada linha da tabela estão em negrito.

Analisando apenas o resultado médio, considerar transições graduais para o processo de Localização de Texto não apresenta melhores resultados. De fato, a publicação original não traz nenhum contexto em que a técnica pode ser utilizada na prática.

Isso não implica que transições graduais não devem ser consideradas: implica apenas que a técnica utilizada, com os parâmetros definidos para este trabalho, não funcionam tão bem quando elas são levadas

em conta. Ainda assim, se analisarmos os resultados individualmente, podemos observar que algumas imagens foram binarizadas com mais sucesso quando levamos em conta as transições graduais.

Uma outra métrica interessante que podemos observar são as médias de *F-Measure*, *Recall* e *Precision* para cada limiar:

Métrica	Limiar da transição gradual				
	0	1	2	5	10
Recall	89.318	85.894	88.655	91.319	93.826
Precision	87.021	83.941	82.503	79.778	68.744
F-Measure	88.063	83.838	84.632	84.039	76.054

Tabela 3: Média das métricas (em %) para cada limiar

Observa-se que, quanto maior o limiar, maior o *Recall*, ou seja, maior a proporção de *pixels* de texto que foram mapeados corretamente. Porém, isso ocorre ao custo de uma queda de *Precision*, o que implica que aumenta também a proporção de falsos positivos.

4.3 EFEITOS DA RETIFICAÇÃO DE TEXTO

Para a análise de desempenho do algoritmo de Localização de Texto após a implementação de texto, foi considerado um dos testes realizados no trabalho original: a análise de métricas obtidas ao executar o algoritmo sobre o banco de dados de 251 imagens de texto do ICDAR. Para mais detalhes do cálculo das métricas, consultar a publicação original[2].

Os resultados encontrados, comparativamente, não foram bons. Com a retificação de texto, o *F-Measure* encontrado foi de 0.33, muito aquém dos 0.51 encontrados originalmente. Algumas razões que poderiam justificar a perda de eficiência são:

- Os parâmetros utilizados funcionavam bem para o algoritmo original, mas precisariam ser re-estimados para o novo algoritmo.
- A retificação nem sempre funciona corretamente. Se o cálculo de ponto de fuga ou do ângulo de cisalhamento não derem os valores corretos, o texto retificado pode ser mais difícil de classificar do que a imagem original.
- O classificador, assim como os parâmetros, precisaria ser reajustado de acordo com o funcionamento do novo algoritmo.

4.4 CONCURSO DE BINARIZAÇÃO DE DOCUMENTOS - DIBCO

O algoritmo enviado ao DIBCO se trata da parte algoritmo de Localização de Texto que se inicia na imagem de entrada e segue até o passo

da binarização. O algoritmo utilizou a *Maxtree* para a computação do *Ultimate Opening*, mas não considerava transições graduais.

Para estimar os parâmetros, foi usado o banco de imagens da competição de 2009, cujos resultados foram apresentados nas tabelas anteriores.

O DIBCO 2011[8] consistiu de 16 imagens, sendo 8 manuscritas e 8 impressas. Executado sobre este banco de dados, o algoritmo proposto teve *F-Measure* de 83,534%, sendo o décimo quarto dentre os dezoito algoritmos enviados. Desta vez, apenas os 3 melhores resultados foram divulgados e algumas métricas mudaram com relação a 2009, de forma que foi mais difícil conseguir comparar o desempenho com os outros algoritmos.

5

CONCLUSÕES

No trabalho desenvolvido foi possível estudar novas literaturas referentes ao problema de Localização de Texto em Imagens de Cena e ver como a implementação destas influencia o resultado final.

No que se refere às técnicas estudadas, em algumas das implementações foi obtido um bom desempenho, enquanto que em outras um estudo mais aprofundado é necessário para obtermos resultados mais robustos. No caso do uso de binarização de imagens com transições graduais, por exemplo, equilibrar o aumento da taxa de acerto com a perda de precisão é um ponto fundamental para que seja possível melhorar os resultados do algoritmo.

Muitos dos conceitos aqui vistos poderiam ser aplicados a outros algoritmos de localização, não necessariamente ficando presos a esta abordagem. Algoritmos de localização baseados em contraste, em geral, podem ter grandes benefícios utilizando a estrutura *Maxtree*. Como vimos, ela permite um rápido processamento, mesmo em imagens não tão pequenas.

Em geral, tratou-se de um trabalho bem abrangente, envolvendo o trabalho de implementar os algoritmos mas também bastante trabalho teórico, principalmente no que se refere a conceitos de Visão Computacional, além de um pouco de Processamento de Imagens e Álgebra Linear. O código vai ficar disponível, de forma que possivelmente alguém interessado nestas áreas possa extendê-lo novamente, tal como foi realizado neste Trabalho de Conclusão.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Document image binarization contest. http://utopia.duth.gr/ ipratika/DIBCO2011/ (Acessado em 5 de junho de 2011). (Citado na página 17.)

[2] W. A. L. Alves and R. F. Hashimoto. Localização de textos em imagens de cenas por meio de operadores morfológicos, 2010. (Citado na página 3, 16, 17, e 23.)

[3] J. Fabrizio and B. Marcotegui. Fast implementation of the ultimate opening. volume 5720, pages 272–281, 2009. (Citado na página 14.)

[4] B. Gatos, K. Ntirogiannis, and I. Pratikakis. Icdar 2009 document image binarization contest (dibco 2009). pages 1375–1382, 2009. (Citado na página 22.)

[5] J. Gomes and L. Velho. *Fundamentos da Computação Gráfica*. IMPA, 2008. (Citado na página 5.)

[6] B. Marcotegui, J. Hernández, and T. Retornaz. Ultimate opening and gradual transitions. 2011. (Citado na página 9.)

[7] G. K. Myers, R. C. Bolles, Q.-T. Luong, J. A. Herson, and H. B. Aradhye. Rectification and recognition of text in 3-d scenes. *IJDAR*, 7:147–158, 2004. (Citado na página 14.)

[8] I. Pratikakis, B. Gatos, and K. Ntirogiannis. Icdar 2011 document image binarization contest (dibco 2011). pages 1506–1510, 2011. (Citado na página 24.)

[9] P. Salembier, A. Oliveras, A. O. Member, and L. Garrido. Antiextensive connected operators for image and sequence processing, 1998. (Citado na página 10.)

Parte II

PARTE SUBJETIVA

6 DIFICULDADES ENCONTRADAS

6.1 DESAFIOS E FRUSTRAÇÕES

O principal desafio encontrado no desenvolvimento deste trabalho foi a leitura das publicações relacionadas a este trabalho. Não apenas por estar em inglês e pela linguagem técnica, mas sim pela falta de detalhes na descrição de como desenvolver as ideias aplicadas e, principalmente, pela provável falta de atenção na transcrição dos códigos (algumas partes dos algoritmos descritos simplesmente não funcionavam corretamente).

Além disso, disciplinas cursadas durante o ano de desenvolvimento do Trabalho de Formatura fizeram com que muitas vezes fosse dificíl conciliar ambas as coisas.

A principal frustração foi descobrir que, ao final da implementação da retificação de texto, não houve melhora de eficiência na classificação das regiões de texto. Como isso foi feito apenas no final do trabalho, não houve tempo para poder estudar como os resultados obtidos poderiam ser melhorados.

6.2 DISCIPLINAS RELEVANTES

6.2.1 *MAC 417 - Visão e Processamento de Imagens*

A disciplina mais importante desta lista, primeiramente porque foi o tema escolhido para o meu Trabalho de Formatura. Durante o ano, foi possível observar na prática a utilidade de várias técnicas aprendidas nessa disciplina.

6.2.2 *MAC 122 - Princípios de Desenvolvimento de Algoritmos*

É a segunda matéria de programação e onde começamos a ver as primeiras estruturas de dados mais complexas (embora hoje a maioria delas nos pareça triviais), sendo que muitas delas foram utilizadas neste Trabalho, tais como pilhas e filas, e onde começa-se a analisar a importância de se usar a estrutura certa para determinada aplicação, além de se verificar que a complexidade do código escrito é importante.

6.2.3 *MAC 323 - Estrutura de Dados*

A continuação natural de MAC 122. A principal importância desta disciplina trata do estudo das estruturas em árvores, fundamental para

a compreensão e o desenvolvimento da *Maxtree*, um dos focos de estudo desta monografia.

6.2.4 *MAT 139 - Álgebra Linear para Computação*

Os principais tópicos estudados em Álgebra Linear que foram importantes para este trabalho são: espaços de coordenadas e mudança de coordenadas. O estudo de espaços de coordenadas é útil pela própria representação de imagens no computador, e o estudo de conceitos sobre mudança de coordendas é útil para a retificação. Esta parece ser uma das disciplinas essenciais do curso, apesar de ser uma das mais complicadas.

6.2.5 *MAC 420 - Introdução a Computação Gráfica*

Em Computação Gráfica foi possível ver, na prática, a utilidade de vários conceitos de Álgebra Linear estudados, que ficavam sempre na teoria. Os mesmo tópicos mencionados para MAT 139 valem aqui, em particular, o espaço projetivo usado para as retificações foi um conceito que só foi visto neste curso.

6.3 TRABALHOS FUTUROS

Neste trabalho não foi possível estudar todas as etapas do algoritmo de Localização de Texto. Em particular, a etapa de classificação foi pouco estudada, e considerando o uso da retificação de texto como uma etapa pré-classificatória, é possível que o classificador pudesse ser melhor treinado para reconhecer regiões de texto com maior grau de sucesso.

Além disso, *Ultimate Opening* com transições graduais é uma técnica nova. Embora ela gere imagens com melhor contraste, ainda não é claro como tal técnica pode ser utilizada para obter resultados mais robustos. Este é um assunto que precisa ser melhor estudado.