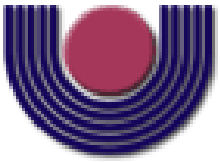


UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Estudo da viabilidade do uso de um OCR na placa Beagleboard e sua integração no xLupa embarcado

Ilson Akito Shigeharu Junior

CASCAVEL

2014

ILSON AKITO SHIGEHARU JUNIOR

ESTUDO DA VIABILIDADE DO USO DE UM OCR NA PLACA BEAGLEBOARD E SUA INTEGRAÇÃO NO XLUPA EMBARCADO

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Marcio Seiji Oyamada

CASCAVEL

2014

ILSON AKITO SHIGEHARU JUNIOR

ESTUDO DA VIABILIDADE DO USO DE UM OCR NA PLACA BEAGLEBOARD E SUA INTEGRAÇÃO NO XLUPA EMBARCADO

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Marcio Seiji Oyamada (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Adair Santa Catarina
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Alexandre Augusto Giron
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 14 de Outubro de 2014

DEDICATÓRIA

Dedico este trabalho aos meus pais, Ilson Akito Shigeharu e Cleufani Aparecida Macedo Shigeharu, que sempre estiveram ao meu lado, apoiando-me em todos momentos de minha vida.

AGRADECIMENTOS

Aos meus pais, por me darem todo o suporte e apoio incondicional ao longo de toda a minha vida.

Ao meu orientador, Prof. Marcio Seiji Oyamada, pela atenção e por sua disposição ao me dar toda a assistência possível sempre que requisitado durante o desenvolvimento deste trabalho, assim como todo conhecimento passado como professor.

A todos os professores que me passaram conhecimento durante toda a minha vida.

Aos meus amigos e companheiros de curso, em especial aos colegas Alexandre Santetti Scortegagna, Fernando Fernandes e Deivide Possamai por contribuirem e me auxiliarem no desenvolvimento deste trabalho e à amiga Débora Moreira Amorim pelo apoio e ajuda com o desenvolvimento da monografia. Além disso, agradeço a todos os demais colegas, amigos e parentes por fazerem parte da minha vida, pois cada um tem um espaço importante nos capítulos da minha vida.

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

O Reconhecimento Ótico de Caracteres ou OCR, abreviado do inglês *Optical Character Recognition* é a conversão mecânica ou eletrônica de imagens digitalizadas de escrita à mão ou digitada. É amplamente utilizado como forma de entrada de dados cuja fonte original está em papel, como documentos, recibos de vendas, cartas ou qualquer registro impresso. Este trabalho apresenta um estudo de viabilidade de execução de um sistema de OCR de código aberto na placa BeagleBoard, uma placa de baixo custo *open source*, e sua integração ao xLupa embarcado, um ampliador digital móvel desenvolvido em código aberto para auxiliar pessoas com baixa visão.

Palavras-chave: Reconhecimento Ótico de Caracteres, Processamento de Imagens Digitais, BeagleBoard, código aberto, viabilidade.

Capítulo 1

Introdução

Reconhecimento Ótico de Caracteres (ou OCR, abreviado do inglês *Optical Character Recognition*) é o processo de reconhecimento de texto em documentos impressos ou escritos à mão. OCR é um campo de pesquisa multidisciplinar que envolve áreas como o reconhecimento de padrões, inteligência artificial e visão computacional. A dificuldade enfrentada pelos OCRs está em superar as diversidades encontradas na interpretação dos caracteres durante o reconhecimento, tais como as distorções geométricas geradas pela aquisição da imagem, qualidade do material analisado, diversidades de fontes e escrita despadronizada. Outro problema é a quantidade de processamento e armazenamento exigidos no processo. A solução para contornar essas adversidades consiste em aplicar técnicas de processamentos de imagens digitais como codificação e transformações de imagem.

O objetivo deste trabalho é desenvolver uma tecnologia assistiva para auxiliar pessoas com baixa visão por meio de um ampliador de tela embarcado capaz de fazer a leitura em voz durante atividades cotidianas que envolvem leitura.

Neste trabalho é apresentada a placa BeagleBoard [1], o xLupa embarcado [2], métodos de processamento de imagens para análise de imagens, exemplos de OCR de código aberto, análise e testes de OCRs de código aberto, e explicado como foi realizada a integração do OCR e do sintetizador de voz com o xLupa embarcado.

1.1 Justificativas

De acordo com o Instituto Brasileiro de Geográfia e Estatística IBGE (2010) [3], 6.585.308 de pessoas no Brasil possuem deficiência visual, das quais 582.624 possuem cegueira e o restante possui baixa visão. No caso de usuários baixa visão, recursos de acessibilidade são necessários para que o mesmo tenha acesso à informação. A acessibilidade em computadores é normalmente fornecida pelo uso de ampliadores de tela. Para documentos impressos, a

ampliação é realizada utilizando lupas óticas e digitais. No entanto, devido à fadiga, o uso de tal recurso normalmente não pode ser utilizado por longos períodos de tempo, sendo necessário o apoio de professores leitores ou algum método de síntese de voz.

1.2 Objetivos

Este trabalho tem como objetivo a integração de um sistema OCR e um sintetizador de voz com o xLupa embarcado. Com essa integração é possível que durante a ampliação de um documento impresso, o usuário baixa visão possa solicitar que o mesmo faça a leitura do documento através de um sintetizador de voz, auxiliando em atividades do seu cotidiano, como ler um livro, jornal, revista, história em quadrinhos, dentre outras.

Os objetivos específicos são:

- Estudar os métodos computacionais envolvidos no processo de OCR e fazer um levantamento das ferramentas de código aberto disponíveis;
- Estudar a placa BeagleBoard;
- Escolher e portar uma ferramenta OCR de código aberto para a placa BeagleBoard e analisar a viabilidade de uso;
- Integrar o OCR escolhido ao xLupa embarcado.

1.3 Organização do Texto

Neste capítulo foi feito uma introdução sobre os tópicos abordados, bem como os objetivos e as motivações para o desenvolvimento deste trabalho.

No Capítulo 2 serão apresentados métodos de processamento de imagens para análise de documentos, que abrangem as etapas de aquisição, transformação, segmentação e extração de características de imagem.

O Capítulo 3 aborda os sistemas OCR, contando um pouco de sua origem, formas de uso, e de funcionamento. Neste capítulo também são apresentados exemplos de sistemas OCR de código aberto.

No Capítulo 4 é apresentado a placa BeagleBoard e suas especificações, além de uma introdução sobre a arquitetura ARM e suas características.

No Capítulo 5 é feito uma avaliação da taxa de acertos dos softwares de OCR assim como a descrição de como foi realizada a integração do OCR escolhido no xLupa embarcado.

Por fim, o Capítulo 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Métodos de Processamento de Imagens para Análise de Documentos

Análise de imagem de documentos, como o nome sugere, é um subcampo da análise de imagens [4]. Isso implica que, por um lado, herda as técnicas mais gerais de análise de imagem, e por outro, pode servir como uma plataforma para testar várias metodologias de análises de imagem. Além disso, com o tempo, a análise de imagem de documentos também adquiriu suas próprias técnicas, especificamente projetadas para suas necessidades.

Este capítulo abrange quatro seções principais: aquisição de imagem, transformação de imagem, segmentação de imagem e uma introdução à extração de características. Na aquisição de imagem é descrito o processo de converter um documento na sua representação numérica, que é adequada para subsequente processamento em um computador. A imagem resultante da etapa de aquisição pode apresentar diversas imperfeições, tais como a presença de ruídos, sombreamento e enviesamento causado por dobras. A etapa da transformação de imagem visa aprimorar a qualidade da imagem para as etapas subsequentes. A extração de características visa simplificar o conjunto de dados requeridos para descrever um grande conjunto com mais precisão, obtendo-se informações relevantes dos dados de entrada em vez de usá-los na íntegra.

2.1 Aquisição

Para poder processar um documento físico na forma digital é necessário convertê-lo em uma representação numérica. Essa conversão é feita através de um digitalizador (*scanner*, câmera, etc.). A imagem é considerada como um sinal analógico que precisa ser amostrada espacialmente convertendo de sinal analógico para digital, isto é, que pode ser representada através de *bits* 0s e 1s. Cada amostra é quantizada em um número finito de níveis de cinza [4], correspondente a um pixel, tendo como única informação sua intensidade. A quantidade de

informação a ser armazenada ou processada pode variar bastante dependendo da qualidade esperada para a aplicação, e da utilização de técnicas de codificação de imagens.

2.1.1 Amostragem

A imagem é considerada uma função $i(x,y)$ de duas variáveis contínuas espacialmente independentes x e y no espaço D, em que $i(x,y)$ representa o nível de cinza da imagem na posição (x,y) e D representa a extensão espacial da imagem [4].

Existe um problema de determinar a resolução espacial no momento de digitalizar uma imagem: quanto maior a resolução, mais informação é preservada. Entretanto, uma resolução alta demais pode revelar texturas do papel e detalhes que podem representar ruídos, dificultando a captação das formas. Sendo assim, a escolha da resolução espacial é geralmente determinada por dois fatores, que são o conteúdo do documento e o objetivo das operações subsequentes. Por exemplo, uma imagem com fontes grandes pode ser escaneada em resolução baixa para continuar sendo legível, mas um texto com fontes pequenas requer uma resolução maior para tal.

O processo de amostragem é um passo necessário na conversão de um documento em uma forma eletrônica, mas ele pode causar perda de informação, como serrilhamento nas bordas. Isto pode ser controlado através do uso de *anti-aliasing* [4], que minimiza esse efeito embaçando as bordas, dando a impressão visual de um contorno mais suave.

2.1.2 Quantização

As amostras produzidas pelo processo de amostragem assumem valores reais contínuos e precisam ser quantizados em números finitos de níveis de cinza para que possam ser processados digitalmente. Se o número de níveis de cinza (L) é igual a 2, a quantização é chamada de binarização ou limiarização (*Thresholding*) [4]. Binarização consiste em separar as regiões de uma imagem quando esta apresenta duas classes, o fundo e o objeto. A forma mais simples de binarização consiste na bipartição do histograma, convertendo os *pixels* cujo tom de cinza é maior ou igual a certo valor de limiar (T) em brancos e os demais em pretos (Figura 2.5). A binarização provê resultados que simplificam drasticamente as operações subsequentes como segmentação e reconhecimento.

2.1.3 Codificação da Imagem

Uma imagem amostrada e quantizada pode conter uma grande quantidade de informação que pode gerar problemas no armazenamento, transmissão e processamento; a codificação de imagem reduz estes problemas. Duas técnicas conhecidas são a Modelagem do Vizinho mais Próximo e a Codificação Entrópica [4]. O primeiro modelo consiste na premissa de que *pixels* adjacentes de uma imagem tendem a ter o mesmo valor, isto é, preto ou branco. Essa propriedade é chamada de "redundância espacial". Uma maneira particularmente simples, mas eficiente de tirar vantagem da redundância espacial é percorrer de forma contígua os *pixels* pretos ou brancos em vez dos próprios *pixels*. O segundo modelo consiste em explorar o fato de que algumas configurações espaciais podem aparecer mais que outras. Essa ideia consiste em usar símbolos curtos para expressões frequentes e símbolos mais longos para expressões raras, resultando em menos informação a ser guardada, em média.

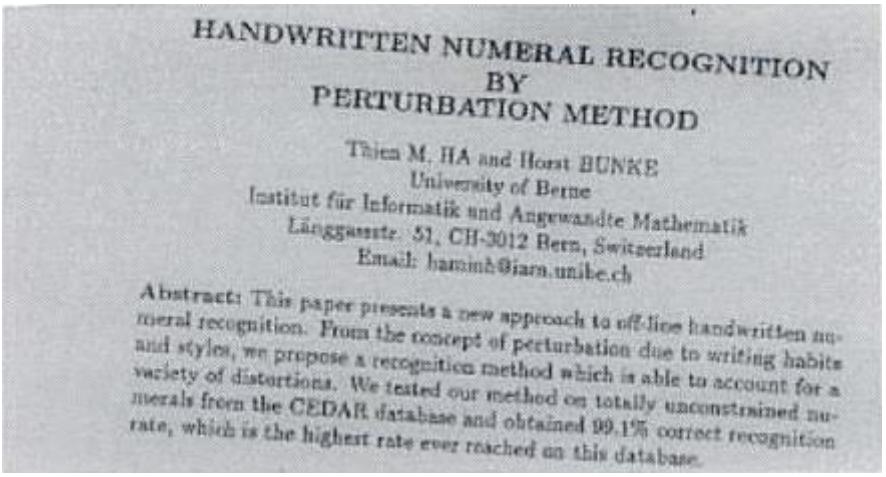
2.2 Transformação de Imagem

Transformação de imagem é uma operação imagem-para-imagem, isto é, a imagem de entrada é transformada em uma imagem de saída, e abrange uma grande quantidade de técnicas que variam de correções geométricas, filtragem e separação da imagem do fundo, à detecção de bordas e afinamento. Transformações geométricas podem servir para corrigir as distorções causadas durante a aquisição da imagem e a normalização de letra de mão despadronizada [4]. A filtragem é uma operação fundamental no processamento de imagem e pode ajudar a aprimorar a separação do objeto (caracteres) e do fundo (*background*) [4][5]. A detecção de borda reduz significativamente a quantidade de dados a serem processados, determinando os contornos dos objetos contidos em uma imagem e descartando informação que é considerada menos relevante, ainda que preservando importantes propriedades estruturais de uma imagem. Afinamento é a transformação de uma imagem digital em uma forma simplificada, mas topologicamente equivalente.

2.2.1 Transformações Geométricas

Um documento adquirido através de um digitalizador pode exibir vários tipos de distorções geométricas que podem ser corrigidas por transformações geométricas apropriadas. Por exemplo, um documento pode não estar bem alinhado em um *scanner* (figura 2.1), gerando

assim uma imagem digital inclinada. Uma aquisição por câmera pode exibir, além da inclinação, uma distorção óptica devido ao sistema imperfeito das lentes (figura 2.2).



HANDWRITTEN NUMERAL RECOGNITION
BY
PERTURBATION METHOD

Thien M. HA and Horst BUNKE
University of Berne
Institut für Informatik und Angewandte Mathematik
Länggassstr. 51, CH-3012 Bern, Switzerland
Email: hamm@iam.unibe.ch

Abstract: This paper presents a new approach to off-line handwritten numeral recognition. From the concept of perturbation due to writing habits and styles, we propose a recognition method which is able to account for a variety of distortions. We tested our method on totally unconstrained numerals from the CEDAR database and obtained 99.1% correct recognition rate, which is the highest rate ever reached on this database.

Figura 2.1 – Parte de um documento escaneado em viés [4]

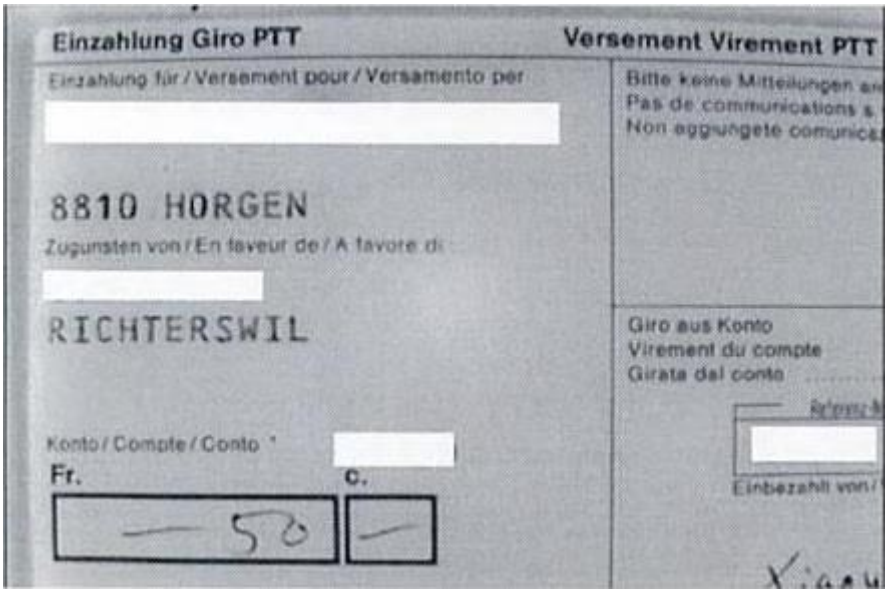


Figura 2.2 – Distorção óptica gerada pela aquisição por câmeras [4]

Transformações geométricas podem também ser usadas para normalizar roteiros escritos à mão de forma despadronizada (figura 2.3). Em desenhos técnicos, caracteres e numerais podem ser escritos em diferentes orientações (figura 2.4) e assim para realizar o reconhecimento, requerem uma prévia rotação.

Figura 2.3 – Exemplo de escrita à mão [4]

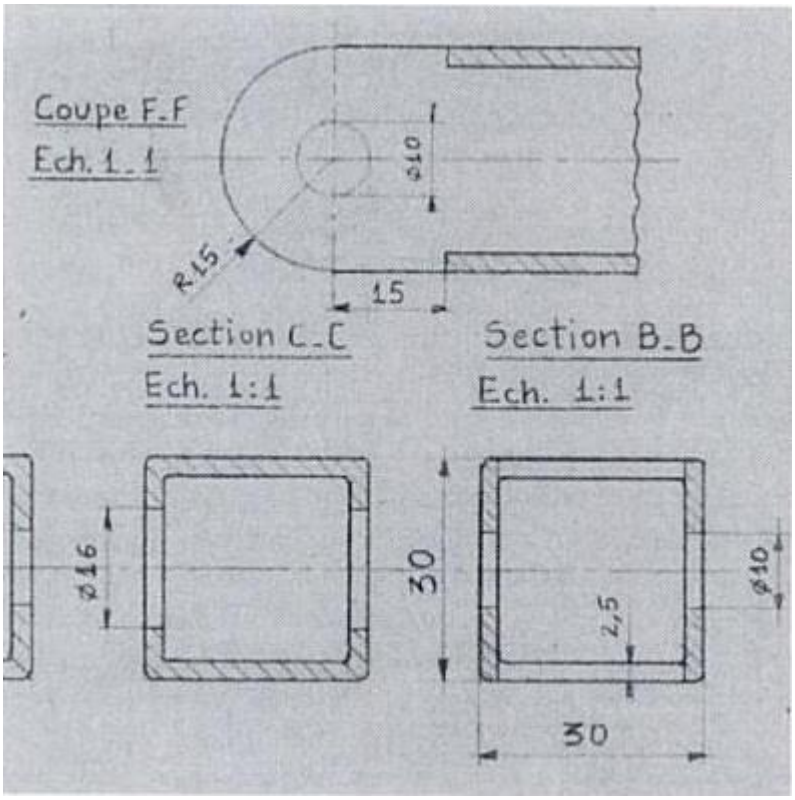


Figura 2.4 – Um desenho técnico [4]

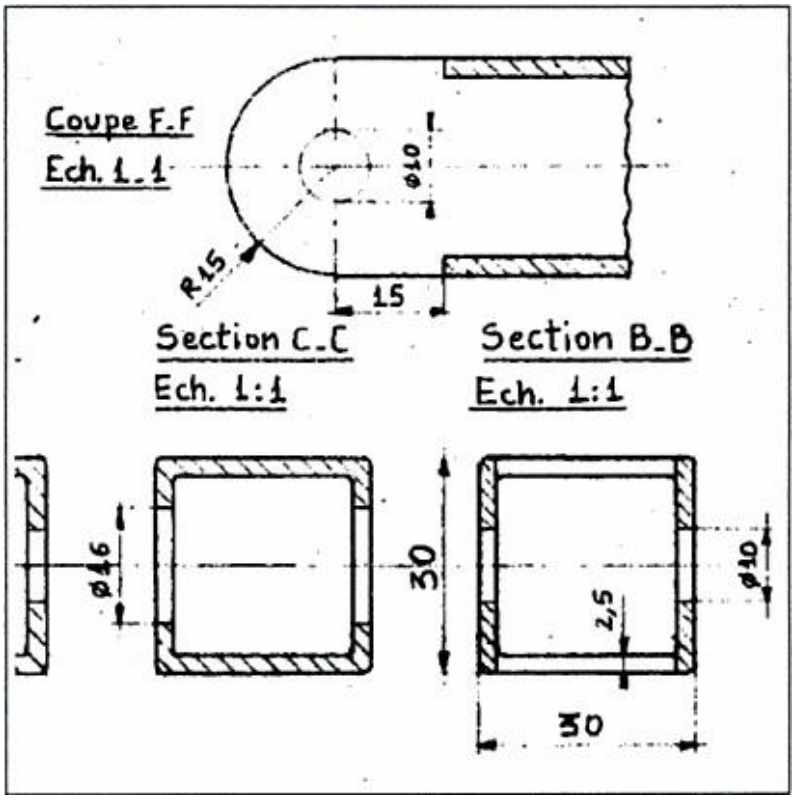


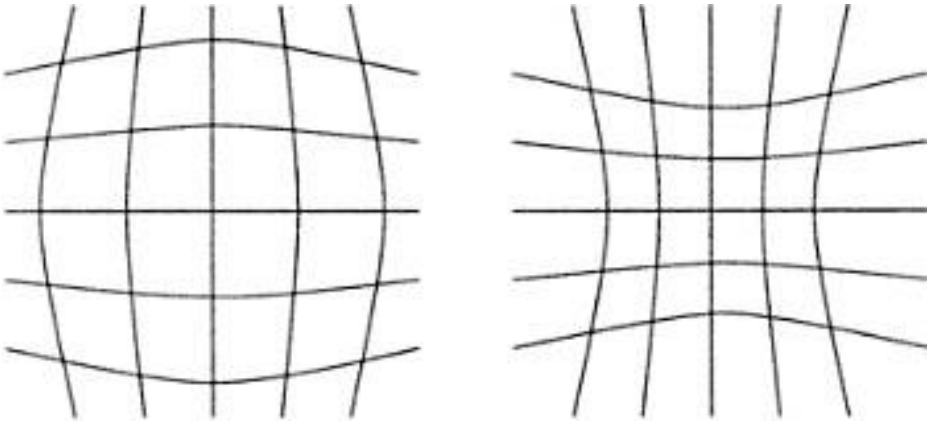
Figura 2.5 – Versão binarizada da figura 2.4 [4]

Para obter a correção apropriada, vários passos são necessários. Primeiro é preciso determinar o tipo de distorção e elaborar um modelo matemático. No caso do problema da imagem inclinada, existem modelos matemáticos precisos para efetuar a rotação necessária e corrigir o problema, embora a determinação dos valores dos parâmetros da transformação (ângulo de rotação) sejam dependentes da aplicação, seja para rotacionar o documento todo ou apenas algumas áreas específicas. No caso do problema da distorção óptica gerado pela aquisição por câmeras, a distorção é não linear e, portanto, exige transformações não lineares como a projetiva e a polinomial. No campo da análise de documentos, o eixo óptico do sistema de câmera geralmente é disposto de modo a ser perpendicular ao plano do documento. Assim, transformações projetivas não são necessárias. Em contraste, transformações polinomiais continuam bastante úteis para modelar distorções geradas por sistemas de lentes. Dois tipos de distorções ópticas comuns são a barril e o efeito almofada (figura 2.6), ambas distorções radiais que podem ser aproximadas pelas equações 1.1 e 1.2:

$$r' = C_m . r + C_d . r^3 \qquad (1.1)$$

$$r' = \sqrt{x'^2 + y'^2}\ ,\ r = \sqrt{x^2 + y^2} \qquad (1.2)$$

Nestas equações, C_m é a ampliação e C_d é o coeficiente de distorção. O coeficiente de distorção é negativo para a distorção barril e positivo para o efeito almofada. No geral, transformações polinomiais podem ser usadas para aproximar uma variedade de distorções geométricas.



Distorção Barril Efeito Almofada

Figura 2.6 – Distorções ópticas devido à imperfeição da lente [4]

2.3 Segmentação

Segmentação é o processo de dividir uma imagem em regiões, cada uma suscetível a conter um único objeto ou um grupo de objetos do mesmo tipo. Por exemplo, um objeto pode ser um caractere em uma página de texto ou um segmento de linha em um desenho técnico; um grupo de objetos pode representar uma palavra ou dois segmentos de linha que se cruzam. Na análise de imagem de documento, quatro algoritmos de segmentação comumente utilizados são rotulagem de componentes conectados, *X-Y-tree decomposition*, *run-length smearing* e transformada de Hough [4].

2.3.1 Rotulagem de Componentes Conectados

Essa técnica padrão atribui a cada componente conectado da imagem binária um rótulo distinto. Os rótulos geralmente são números naturais começando do um até o número total de componentes conectados na imagem de entrada. O algoritmo processa a imagem da esquerda para direita e de cima para baixo. Na primeira linha contendo *pixels* pretos, um único rótulo é atribuído para os *pixels* pretos contíguos. Para cada *pixel* preto das próximas linhas seguintes, os *pixels* vizinhos na linha anterior e o *pixel* da esquerda são examinados. Se algum desses *pixels* vizinhos estiver rotulado, o mesmo rótulo é atribuído ao atual *pixel* preto; caso contrário o próximo rótulo não utilizado é usado. Esse procedimento continua até a última linha da imagem.

Após completar esse processo, um componente conectado pode conter *pixels* tendo diferentes rótulos porque quando é examinado a vizinhança de um *pixel* preto (o *pixel* "?" na Figura 2.7), o *pixel* a sua esquerda e aqueles na linha anterior podem ter sido rotulados diferentemente. Esta situação precisa ser detectada e corrigida. Depois de rotulados os *pixels*, o processo é completado unificando os rótulos conflitantes e reatribuindo rótulos não utilizados.

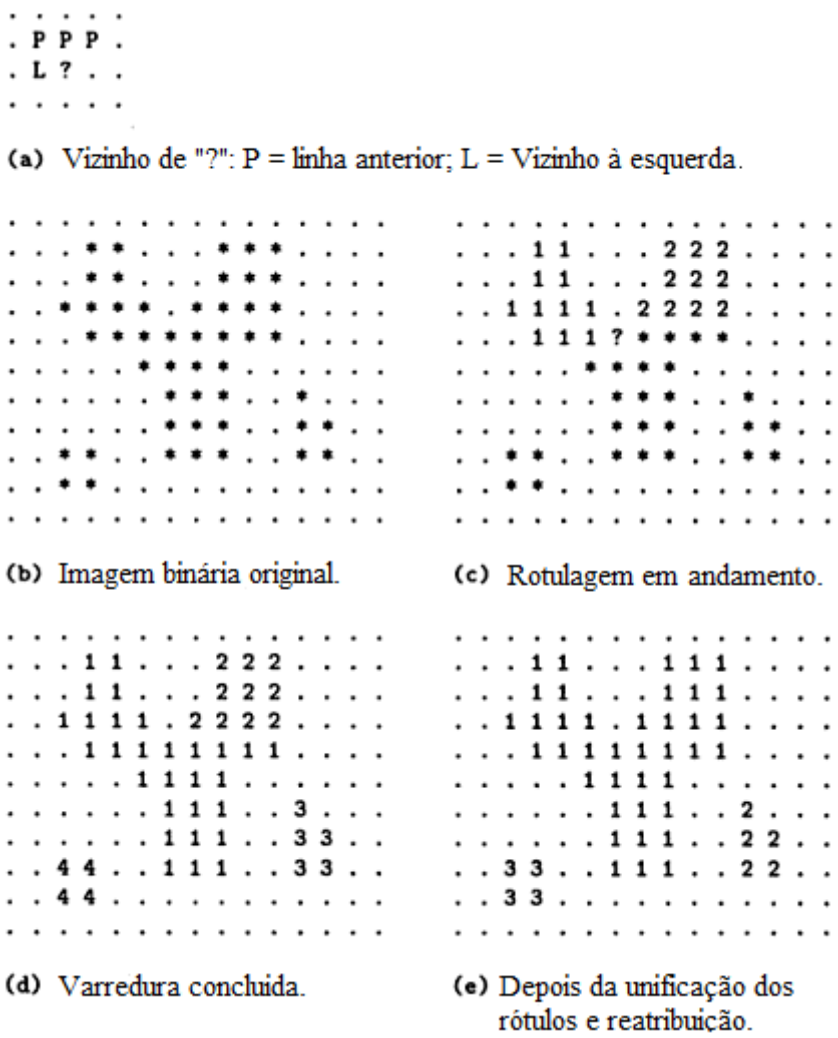


Figura 2.7 – Rotulagem de Componentes Conectados [4]

2.3.2 X-Y-Tree Decomposition

O *X-Y-tree decomposition*, também chamada de método *Iterative Projection Profile Cuttings*, é um algoritmo popular de segmentação no contexto de análise de documentos [4]. A ideia básica deste algoritmo é a exploração do fato de que a maioria das imagens de documentos tem uma estrutura vertical e/ou horizontal. De fato, uma página normal de texto geralmente é composta de diferentes linhas horizontais de texto. Essa observação imediatamente leva a ideia de projetar horizontalmente os *pixels* pretos em um eixo vertical (Figura 2.8). O perfil resultante claramente indica a estrutura de linha da página. Uma simples análise do resultado, como a comparação do perfil de projeção com um limiar, segmenta a página em faixas de brancos e faixas de texto. Uma faixa de branco corresponde a um conjunto de linhas contíguas tendo menos que N *pixels* pretos, e uma faixa de texto

corresponde a um conjunto de linhas contíguas que têm pelo menos N *pixels*, se o limiar do comparador está configurado para N. Subsequentemente, cada faixa de texto pode ser projetada verticalmente em um eixo horizontal contendo as posições dos caracteres com essa faixa. O seguinte procedimento pode ser usado para extração de caracteres:

1. Computar o perfil de projeção horizontal por toda a página.
2. Analisar o perfil de projeção para extrair as linhas.
3. Para cada linha, computar o perfil de projeção vertical.
4. Analisar cada perfil de projeção obtido no passo 3 para extrair os caracteres.

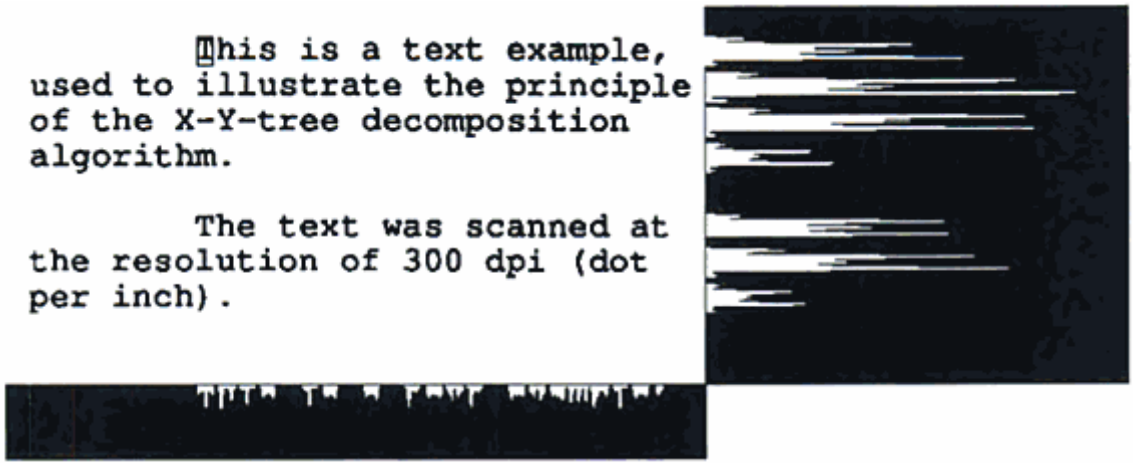


Figura 2.8 – Projeção horizontal da página e projeção vertical da primeira linha [4]

A estrutura de dados para armazenar o resultado do algoritmo é uma árvore cujos nodos representam zonas retangulares (Figura 2.9). Cada nodo pode ter vários filhos, cada um representando uma sub-região da região do pai. As folhas são aquelas que não podem mais ser decompostas. Elas representam as regiões indivisíveis e são chamadas entidades atômicas.

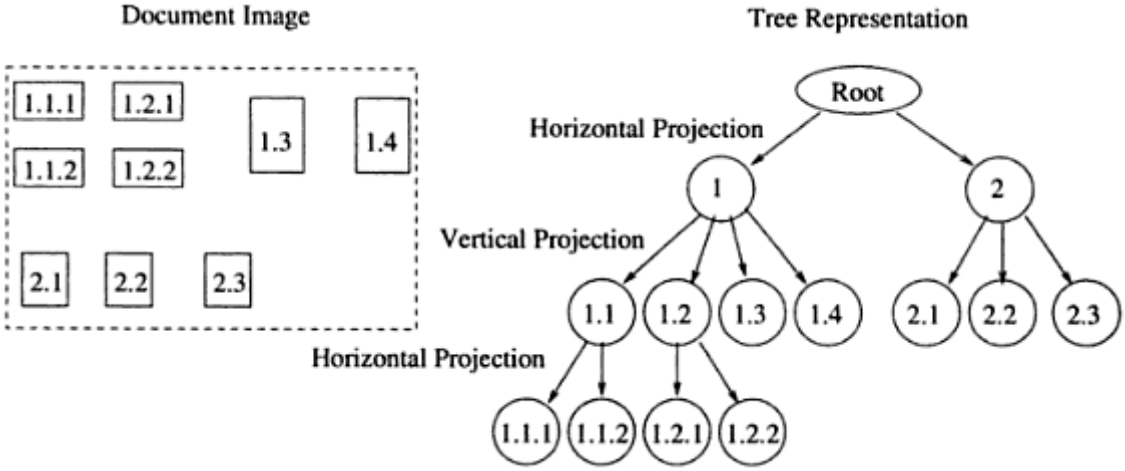


Figura 2.9 – Um documento e sua representação em árvore [4]

Esta técnica é utilizada para detecção e avaliação do enviesamento do texto na imagem digitalizada, para posterior tratamento. O grau de enviesamento é o valor de inclinação da linha de base do texto escaneado com relação à horizontal. Isso ocorre durante o processo de escaneamento do documento aonde a folha que contém o texto acaba inclinando em relação ao sensor e, consequentemente, todo o texto aparece com um grau fixo de enviesamento em relação ao eixo x.

2.3.3 Run-Length Smearing (RLS)

Considerando uma linha de 0s (*pixels* brancos) e 1s (*pixels* pretos), o RLS primeiro detecta todos brancos contíguos da linha e então converte aqueles cujo comprimento é menor que um limiar T pré-definido, para contíguos pretos. Os pretos contíguos são inalteráveis. Por exemplo, com T = 3, o RLS converte a linha

00011010111001000111100

em

00011111111111000111111

Para obter a segmentação, o algoritmo RLS é primeiro aplicado linha por linha e depois coluna por coluna, gerando dois *bitmaps* distintos, que são eventualmente combinados pela operação lógica *AND*.

A imagem 2.10 ilustra os passos descritos acima.

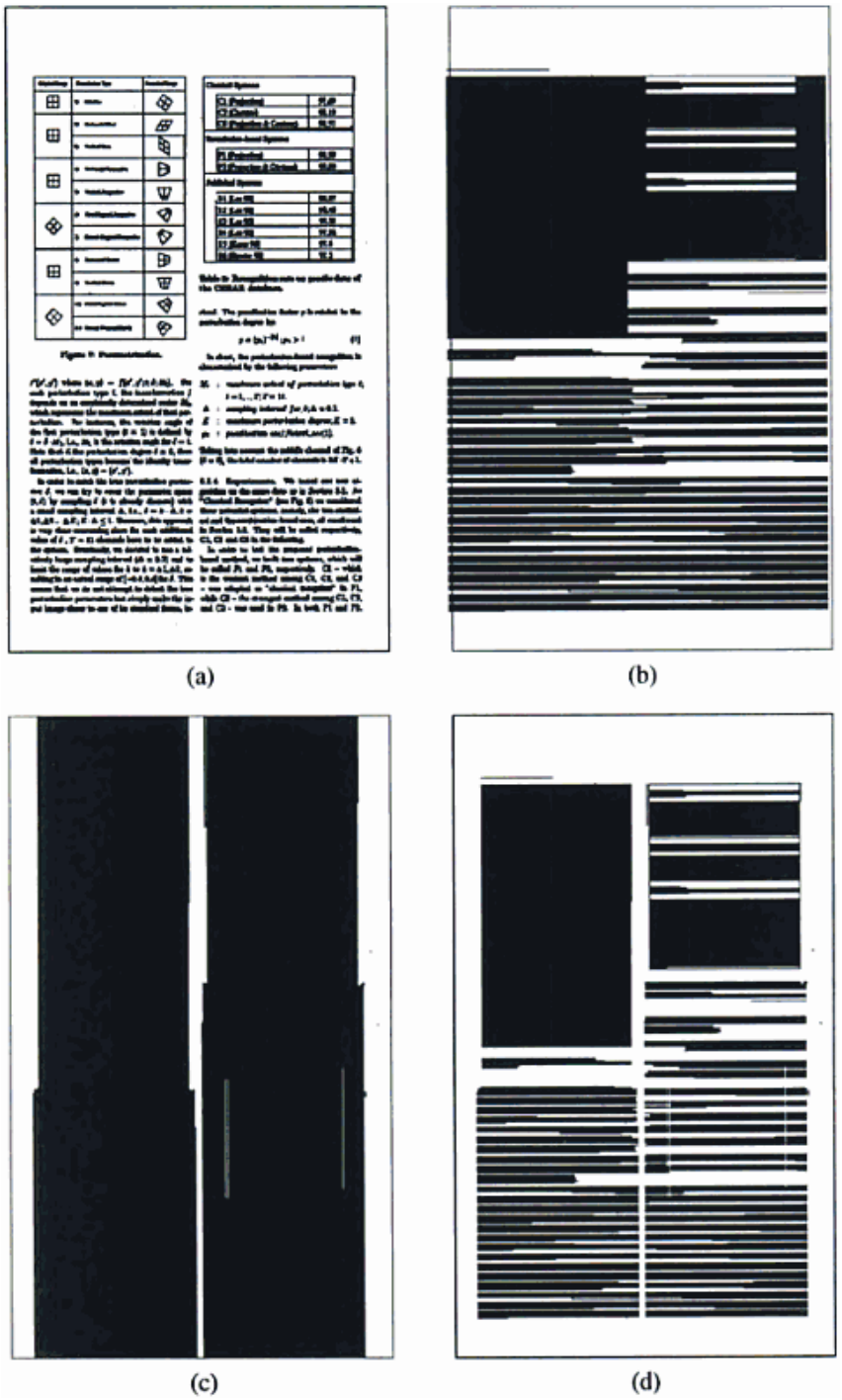


Figura 2.10 – Segmentação de página pelo algoritmo RLS. (a) Uma página contendo ilustração e texto. (b) RLS aplicado em linha. (c) RLS aplicado em coluna. (d) Versão segmentada de (a) pelo algoritmo RLS [4]

O resultado do algoritmo RLS pode ser usado como entrada para a rotulagem de componentes conectados, gerando um conjunto de regiões. Aqueles que possuem tamanhos muito grandes em ambas as coordenadas x e y geralmente correspondem aos objetos gráficos onde as linhas de texto geralmente produzem regiões alongadas horizontalmente.

2.3.4 Transformada de Hough

O objetivo dessa técnica é encontrar em imagens digitais formas geométricas que possam ser parametrizadas, tais como círculos e elipses, para auxiliar a detecção de caracteres [4]. As formas dos objetos são determinadas através de um processo de votação. A votação é levada até o espaço de parametrização, da qual objetos candidatos são obtidos como máximos locais no espaço acumulador que é construído explicitamente pelo algoritmo para computar a transformada de Hough [4].

Considerando a equação de uma reta $y = ax + b$, da qual os parâmetros (a,b) devem ser determinados, se o ponto (x_1,y_1) pertence à reta, então qualquer par (a,b) que satisfaça $y = ax + b$, é uma solução potencial. Em outras palavras, para um dado ponto (x_1,y_1), a reta $b = -x_1a + y_1$ no espaço de parametrização (a,b) descreve todas soluções possíveis. O mesmo é verdade para os pontos (x_2,y_2), e assim por diante. Se os n pontos (x_1,y_1), (x_2,y_2), ..., (x_n,y_n) estiverem na mesma reta que o espaço da imagem, suas curvas correspondentes precisam intersectar umas as outras no mesmo ponto (a^*,b^*) no espaço de parametrização. Baseado nessa observação, o algoritmo de Hough funciona da seguinte forma:

1. Quantizar o espaço de parametrização (a,b) em células.

2. Formar uma matriz acumuladora $Acc(a.b)$ e inicializar seus componentes com zero.

3. Para cada pixel preto (x,y) no espaço da imagem, todas as células de $Acc(a,b)$ que satisfazem $b = -x_1a + y_1$, são incrementados em um.

4. Detectar a máxima local no espaço de parametrização (a,b). Cada máxima corresponde a um conjunto de pontos colineares no espaço da imagem.

Na prática, o valor de a é infinito para retas verticais, o que causa problemas no Passo 1. Uma parametrização de reta mais apropriada é $r = x \cdot cos\ \theta + y \cdot sen\ \theta$ (Figura 11), onde r é a distância da origem do espaço (x,y) até a reta e θ é o ângulo da normal até a reta. Isso produz uma senóide no espaço de parametrização (θ,r) para cada ponto (x,y).

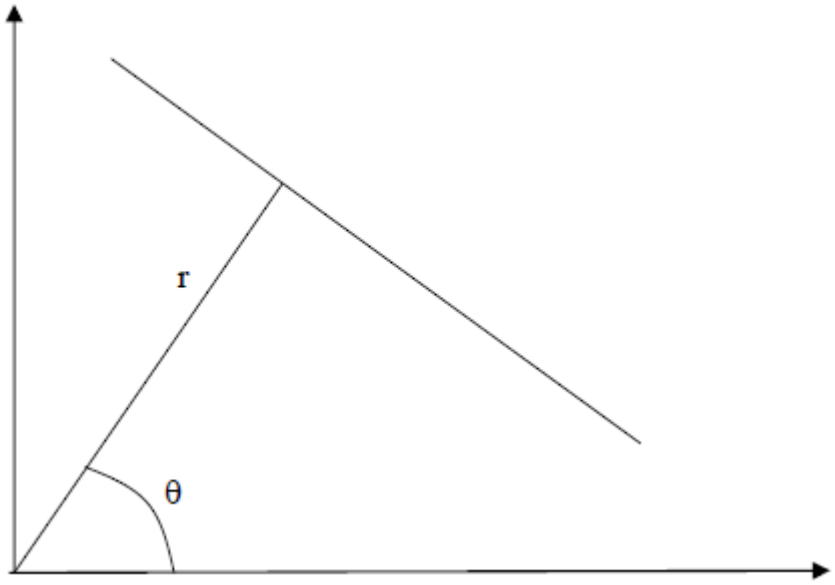


Figura 2.11 – Representação da parametrização de reta

Assim, para todo pixel preto (*x*,*y*) no espaço da imagem, a matriz acumuladora será incrementada com a tupla (θ,*r*), variando θ de 0 até o valor definido pelo usuário, de forma que cada ponto (*x*,*y*) terá uma senóide no acumulador. Se uma reta é desenhada na imagem, ocorrerá um acúmulo de pontos no acumulador, pois cada ponto da reta vota no mesmo ponto do acumulador. Em seguida, é detectado o máximo local no espaço de parametrização (*a*,*b*) e encontrado as retas na imagem original.

2.4 Extração de Características

Esta etapa procura extrair características das imagens resultantes da segmentação através de descritores que permitam caracterizar com precisão cada caractere. Por exemplo, uma dificuldade normalmente encontrada pelos OCR é a discriminação entre dígitos parecidos, como o “5” e o “6”. Estes descritores devem ser representados por uma estrutura de dados adequada ao algoritmo de reconhecimento. É importante observar que nesta etapa a entrada ainda é uma imagem, mas a saída é um conjunto de dados correspondentes àquela imagem [5].

Para maior clareza, supondo-se que os descritores utilizados para descrever um caractere sejam as coordenadas normalizadas x e y de seu centro de gravidade e a razão entre sua altura e largura. Neste caso, um vetor de três elementos é uma estrutura de dados adequada para armazenar estas informações sobre cada dígito processado por essa etapa.

A extração de características pode ser dividida em duas abordagens: estatística e estrutural [4]. A abordagem estatística consiste em determinar o caractere identificado comparando-o com os encontrados em um banco de dados contendo diversos tipos de caracteres. A abordagem estrutural classifica padrões com base nos componentes dos caracteres e a relação entre estes componentes. Primeiro as primitivas (formas geométricas que possam ser parametrizadas) do caractere são identificadas e em seguida, sequências das primitivas são verificadas com base em regras pré-determinadas. Geralmente um caractere é representado em uma árvore e sua classificação é feita encontrado um caminho até uma folha. O percurso do caminho na estrutura é traçado e as informações estruturais do percurso são guardadas [6].

Capítulo 3

OCR

OCR é uma tecnologia utilizada para reconhecer caracteres a partir de um arquivo de imagem, sejam eles escritos à mão ou digitada. É geralmente um processo "*offline*", que analisa um documento estático e obtém um arquivo de texto. Os primeiros OCRs podem ser destacados por atuarem em torno de dois tópicos: telegrafia expandida e criação de dispositivos de leituras para cegos [7]. Por volta de 1914, Emanuel Goldberg desenvolveu uma máquina que lia caracteres e os convertia em código telegráfico padrão [8]. Por volta dessa época, Edmund Fournier d'Albe desenvolveu o Otofone, um *scanner* portátil que quando movido sobre uma página impressa, produzia tons que correspondem a letras específicas ou caracteres [8].

Sistemas OCR podem ser usados para entrada de dados para documentos de negócios, reconhecimento automático de número de placas, agilizar a criação de versões textuais editáveis de documentos impressos, tornar pesquisável as imagens eletrônicas de documentos impressos, entre outros.

Vários sistemas OCR comerciais e *open source* estão disponíveis para várias línguas, tais como: latim, cirílico, árabe, hebraico, sânscrito, caracteres chineses, japoneses e coreanos.

Softwares OCR geralmente fazem pré-processamento da imagem para aprimorar as chances de reconhecimento bem sucedido através de binarização e segmentação.

Existem dois tipos básicos de algoritmos de OCR que podem produzir uma lista ordenada de caracteres candidatos [9]:

- Matriz de correspondência envolve a comparação de uma imagem para um glifo[1] armazenado em uma base de *pixel*-a-*pixel*, também conhecida como "correspondência de padrão" ou "reconhecimento de padrão" [10]. Isto depende da entrada de glifo estar corretamente isolada do resto da imagem e do glifo armazenado estar em uma fonte semelhante e na mesma escala. Esta técnica funciona melhor com texto datilografado e

não funciona bem quando novas fontes são encontradas. Esta é a técnica implementada nos primeiros OCRs.

- Extração de características decompõe glifos em "características" como linhas, ciclos fechados, direção de linha e os cruzamentos de linha. Estes são comparados com uma representação vetorial abstrata de um caractere, o que pode reduzir a um ou mais protótipos de glifo. Técnicas gerais de detecção de características em visão computacional são aplicáveis a este tipo de OCR, que é comumente visto em reconhecimento "inteligente" de escrita e nos mais modernos OCRs [10]. Algoritmos do Vizinho Mais Próximo são usados para comparar as características de imagem com características de glifos armazenados e escolher a correspondência mais próxima [11].

Softwares como o Tesseract [12] e o Cuneiform [13] usam uma abordagem de duas passagens de reconhecimento de caracteres. A segunda passagem é conhecida como "reconhecimento adaptativo" e usa as formas reconhecidas de letras com alto grau de confiança na primeira passagem para reconhecer melhor as letras restantes na segunda passagem. Isso é vantajoso para fontes incomuns ou digitalizações de baixa qualidade, onde a fonte é distorcida (por exemplo, borrada ou desbotada) [14].

Após o reconhecimento dos caracteres, é realizado um pós-processamento para otimizar o resultado. A precisão do OCR pode ser aumentada se a saída é limitada por um léxico – uma lista de palavras que têm permissão para ocorrer em um documento [15]. Isso pode ser, por exemplo, todas as palavras em um idioma específico, ou um léxico mais técnico para um campo específico. Esta técnica pode ser problemática se o documento contiver palavras não pertencentes ao léxico, como nomes próprios. O *software* Tesseract usa seu dicionário para influenciar a etapa de segmentação de caracteres, para maior precisão [14].

O fluxo de saída pode ser um fluxo de texto sem formatação ou um arquivo de caracteres, mas os sistemas de OCR mais sofisticados podem preservar o *layout* original da página e produzir, por exemplo, um PDF com anotações que inclui tanto a imagem original da página quanto uma representação textual pesquisável.

"Análises de vizinho mais próximo" podem fazer uso de frequência de coocorrências para corrigir erros, notando que certas palavras são frequentemente encontradas próximas [10]. Por exemplo, "Washington, DC" geralmente é muito mais comum em Inglês do que "Washington DOC".

Conhecimento da gramática da língua que está sendo digitalizada também pode ajudar a determinar se uma palavra é provavelmente um verbo ou um substantivo, por exemplo, permitindo maior precisão.

Recentemente, os principais fornecedores de tecnologia OCR passaram a ajustar os sistemas de OCR para melhor lidar com tipos específicos de entrada. Além de um léxico específico do aplicativo, maior eficiência no reconhecimento de caracteres pode ser obtida levando-se em conta as regras de negócio, expressão padrão, ou informações contidas em imagens coloridas. Esta estratégia é chamada "OCR Orientada a Aplicação" ou "OCR Personalizado", e tem sido aplicada para cartões de negócio, notas fiscais, captura de telas, cartões de identificação, carteira de habilitação e etiquetas.

3.1 OCRs de código aberto

Esta seção apresenta alguns exemplos de OCR de código aberto.

Tesseract: foi originalmente desenvolvido pela Hewlett-Packard e é atualmente mantido pelo Google [12]. Até a versão 2 ele aceitava como entrada apenas imagens em formato tiff com o texto digitalizado em uma única coluna, convertendo-a na saída em formato txt. Essas versões não possuíam mecanismos para reconhecimento de *layout*; se fossem utilizados textos com mais de uma coluna, imagens ou equações produziam uma saída ilegível. O Tesseract pode detectar se o texto é não-proporcional ou proporcional, onde texto não-proporcional é quando cada caractere ocupa o mesmo espaço horizontal independentemente de sua largura – por exemplo, um *i* ocupa o mesmo espaço que um *m* – e texto proporcional é o contrário, onde cada caractere ocupa um espaço específico.

CuneiForm: suporta 20 linguagens e cada linguagem é provida de um dicionário que permite uma checagem de contexto de caracteres reconhecidos e aprimora o resultado dos reconhecimentos. Trabalha facilmente com imagens, tabelas de diferentes estruturas, colunas e vários tipos de fontes. Melhorou a procura automática e semiautomática de texto, tabelas e imagens, que torna o trabalho com documentos de complexa estrutura altamente flexível [13].

GOCR (ou JOCR): converte imagens digitalizadas de texto para arquivos de texto. Joerg Schulenburg iniciou o programa e agora lidera uma equipe de desenvolvedores. GOCR pode ser usado com diferentes *front-ends*, o que torna muito fácil para portá-lo para diferentes sistemas operacionais e arquiteturas. Ele pode abrir muitos formatos de imagem diferentes, e sua qualidade tem melhorado constantemente [16].

Ocrad: baseado em um método de extração de características, ele lê imagens em formatos *pixmap* portáteis conhecidos como *Portable anymap*, e produz texto em formatos *byte* (8 *bits*) ou UTF-8. Também é incluído um analisador de *layout*, capaz de separar as colunas ou blocos de texto normalmente encontrados nas páginas impressas [17].

Capítulo 4

BeagleBoard

BeagleBoard é um *kit* de desenvolvimento integrado em uma única placa de baixo custo. Sua primeira versão foi lançada em 28 de julho de 2008 pela parceria entre a Texas Instruments e a Digi-Key [1]. A placa foi desenvolvida por uma pequena equipe de engenheiros como uma placa educacional que poderia ser usada em faculdades de todo o mundo para ensinar hardware *open source* e recursos de software de código aberto. Também é vendido ao público sob a licença Creative Commons Share-Alike.

A BeagleBoard mede aproximadamente 75 por 75 mm e tem todas as funcionalidades de um computador básico [18]. O OMAP3530 [19] inclui um processador ARM Cortex-A8, um DSP TMS320C64x+ para aceleração de vídeo e decodificação de áudio, e uma GPU PowerVR SGX530 da Imagination Technologies para proporcionar renderização 2D e 3D em aplicações OpenGL ES 2.0. A saída de vídeo é fornecida por conectores S-Video e HDMI separados. Um *slot* de cartão único SD/MMC com suporte a SDIO, uma porta USB, uma conexão serial RS-232, uma conexão JTAG, e dois conectores de 3,5 mm estéreo para áudio de entrada e saída são fornecidos. A placa pode ser alimentada a partir do conector USB, ou uma fonte de alimentação independente de 5 V. Devido ao baixo consumo de energia, refrigeração adicional ou dissipadores de calor não são necessários.

A versão utilizada para este trabalho foi a BeagleBoard-XM, que contém o sistema-em-um-chip DM3730. Esta plataforma possui interfaces e drivers de vídeo, USB e som, dimensões reduzidas (8.5x8.7 cm) e baixo consumo de potência, podendo ser alimentada a partir do conector USB, ou uma fonte de alimentação independente de 5 V. Além disso, ela possui uma interface nativa com suporte a câmeras de resolução VGA até 5 MP (Mega Pixel). Um resumo dos principais componentes desta placa pode ser visto na tabela 4.1.

Tabela 4.1: Plataforma de Desenvolvimento BeagleBoard-xM

Processador	Processador ARM Cortex-A8 de 1 GHz
Memória	512 MB LPDDR RAM
Periféricos e conexões	Conexões S-Video, HDMI/DVI, JTAG, entrada e saída Estéreo, 4 portas USB, porta RS-232 (serial) e porta Ethernet
Armazenamento	Slot cartão SD

4.1 Arquitetura ARM

A arquitetura ARM é classificada como *Reduced Instruction Set Computer* (RISC) ou Computador com um Conjunto Reduzido de Instruções. São usadas principalmente em sistemas embarcados e seu desenvolvimento se deu visando obter o melhor desempenho possível, com a limitação de ser simples, ocupar pouca área e ter baixo consumo de energia [20].

A arquitetura ARM é desenvolvida pela companhia Britânica ARM Holdings, que projeta e licencia uma família de processadores baseado no conjunto de instruções ARM. A arquitetura de 32 *bits* foi desenvolvida na década de 1980 pela Acorn Computers Ltd para alimentar suas máquinas *desktop* e posteriormente desmembrada como uma empresa separada, agora ARM Holdings. Atualmente é a arquitetura de conjunto de instruções de 32 *bits* mais utilizada globalmente em termos de quantidade produzida [21][22]. De acordo com a ARM Holdings, só em 2010, fabricantes de *chips* baseados em arquiteturas ARM informaram 6.1 bilhões de remessas de processadores ARM, representando 95% dos smartphones, 35% dos televisores digitais e 10% dos computadores móveis [23]. O nome originalmente era um acrônimo para Acorn RISC Machine [24] e, posteriormente, depois que o nome Acorn foi abandonado, Advanced RISC Machine.

Usando uma abordagem baseada em RISC para o projeto do computador, processadores ARM possuem uma série de características, tais como, o conjunto de instruções simples e reduzido, formato de instrução fixo, uma instrução por ciclo de máquina, entre outras [25]. Os benefícios desta abordagem são os custos, calor e uso de energia reduzidos em comparação com designs de *chips* mais complexos, características que são desejáveis para dispositivos

leves, portáteis e alimentados por bateria, como smartphones e *tablets*. A complexidade reduzida e design mais simples permite que as empresas construam um Sistema-em-um-*Chip* de baixo consumo de energia para um sistema embarcado incorporando memória, interfaces, etc.

A arquitetura ARM de 32 *bits* inclui as seguintes características RISC [26]:

- Arquitetura load/store, onde o processamento de dados opera apenas nos conteúdos de registro, e não diretamente sobre os conteúdos de memória.
- Modos de endereçamento simples, com todos os endereços de load/store determinados somente a partir do conteúdo do registro e campos de instrução.
- Sem suporte para acessos à memória desalinhada (embora agora suportada desde os núcleos ARMv6 e posteriores.
- Execução condicional da maioria das instruções, reduzindo a sobrecarga causada por desvios nas operações de tomada de decisão e compensando pela falta de um preditor de desvio (*Branch Predictor*).
- Instruções aritméticas alteram códigos de condição somente quando desejado.
- Formato de instruções de 3 endereços (isto é, os dois registradores operandos e o registrador de resultado são independentemente especificados).

4.1.1 Cortex-A8

O processador usado pela BeagleBoard é Cortex-A8, projetado pela *ARM Holdings* e baseado na arquitetura de conjunto de instruções ARMv7 [27]. Este processador possui um total de 40 registradores de 32 *bits*, sendo 33 registradores para propósito geral e 7 registradores de *status*. Estes registradores não são acessíveis ao mesmo tempo. O estado do processador e o modo de operação determinam os registradores que estão disponíveis para o programador. Suas características podem ser vistas na tabela 4.2.

Tabela 4.2: Características do processador Cortex-A8

Cache L1 otimizado	Cache L1 de 64 KB, dividido em dois blocos de 32 KB (dados e instruções). O cache de nível 1 é integrado no processador com um único ciclo de tempo de acesso. Os caches combinam mínima latência de acesso com determinação dividida para maximizar o desempenho e minimizar o consumo de energia [27].
Cache L2 integrado	Cache L2 de 256 KB, integrado no núcleo, que pode ser expandido para até 1 MB de acordo com o nível de desempenho desejado pelo fabricante [27].
Previsão de desvio dinâmica	Para minimizar as penalidades de previsão de desvio erradas, o preditor de desvio dinâmico atinge 95% de precisão em uma ampla gama de benchmarks da indústria. O preditor é ativado por meta de desvio e buffers de histórico global. O mecanismo de repetição minimiza a penalidade por falha de previsão [27].
Unidade de Gerenciamento de Memória	A Unidade de Gerenciamento de Memória (ou MMU, abreviado do inglês *Memory Management Unit*) permite que o Cortex-A8 execute uma rica variedade de aplicativos e sistemas operacionais [27].
Sistema de Memória	Otimizado para eficiência no consumo de energia e alto desempenho. O cache L1dividido em 2 blocos limita a ativação das memórias a serem usadas somente quando necessárias [27].

Capítulo 5

Avaliação dos softwares de OCR e integração ao xLupa embarcado

5.1 Metodologia

A integração com o xLupa embarcado foi realizada para que o usuário de baixa visão, durante a ampliação de um documento impresso, possa solicitar que o mesmo faça a leitura do documento através de um sintetizador de voz. Como o xLupa já está portado para a BeagleBoard, o estudo envolveu a integração do sistema OCR e do sintetizador de voz ao código do xLupa embarcado.

Os materiais referentes aos métodos computacionais envolvidos na implementação de um OCR foram obtidos em livros e artigos. As ferramentas de OCR livres foram buscadas principalmente na Internet. Para auxiliar nesta etapa, os resultados obtidos previamente com testes realizados na arquitetura x86, foram utilizados para escolha da solução adequada. As informações sobre a placa BeagleBoard foram obtidas principalmente do site oficial da placa [1].

Para determinar qual OCR seria portado/compilado para a BeagleBoard, foram escolhidos OCRs desenvolvidos em C/C++, para garantir a compatibilidade com o S.O. de distribuição Linux utilizado na BeagleBoard, o Ubuntu 12.04.4 LTS 32 bits de codinome Precise Pangolin.

Os testes foram realizados na arquitetura x86 utilizando o S.O. Windows 7 Professional 64 bits SP1, em um desktop com processador Intel i5 2.67GHz, 4 GB de memória RAM DDR3 e placa de vídeo ATI Radeon HD5850.

O reconhecimento ótico de caracteres em imagens digitalizadas pode ser bastante desafiador devido a fontes muito pequenas e suavizadas. Para avaliar o desempenho de sistemas OCR no reconhecimento de palavras, existem base de dados padronizadas, como as propostas por Wachenfeld, Klein e Jiang [28], a *Screen-Char database* que contém imagens de caracteres individuais e isolados, e a *Screen-Word database*, que consiste de duas partes,

sendo uma composta imagens extraídas de documentos publicados com por palavras incorporadas em frases, e outra composta por palavras isoladas e geradas artificialmente em diversos tamanhos e situações. Ambas bases de dados contêm metadados com informações acerca da altura dos caracteres, tipo de fonte, estilo e condições de renderização. Como o foco principal deste trabalho não é fazer uma avaliação aprofundada sobre os sistemas OCR disponíveis em código aberto, para mostrar qual o OCR mais eficiente no reconhecimento de palavras foi adotada uma abordagem mais sucinta e direta para apresentar os resultados avaliados. A metodologia utilizada para os testes consiste em contabilizar as palavras que os programas OCR conseguiram reconhecer dentre as palavras legíveis nas imagens – incluindo sinais de pontuação – levando em consideração acertos parciais. Sendo estes as palavras que, embora tenham sofrido a troca de algum caractere indevido, ainda poderiam ser reconhecidas por leitores, aos quais foram apresentados os textos produzidos pelos OCRs.

Os testes foram realizados em 13 imagens diferentes, capturadas de diferentes materiais presentes no cotidiano como rótulo de remédio, documentos e jornais. A obtenção das imagens digitalizadas foi realizada por meio da captura de tela do monitor e da captura de imagem utilizando uma *webcam*. Em todas as imagens foram feitos testes no seu tamanho original, tamanho ampliado e em diferentes ângulos, embora em alguns casos os conversores não tenham apresentado dados de saída para todas as situações, como no caso em que a imagem está enviesada demais.

No presente trabalho serão mostrados os resultados em um subconjunto com 6 imagens executados com cada um dos 4 *softwares* citados na seção 3.1. O subconjunto de 6 imagens, foi escolhido para abranger o maior número de tipos de material impresso, tanto em termos de organização do texto como tamanho de fonte. Na seção a seguir são apresentados os resultados dos acertos parciais e as tabelas com a contagem das palavras reconhecidas corretamente.

5.2 Resultados da avaliação

O primeiro teste foi realizado com uma imagem do trecho de um documento capturada da tela (Figura 5.1) e o resultado de acertos obtidos constam na Tabela 5.1. Sobre este deve-se acrescentar:

- Com o Tesseract a letra “l” foi quase totalmente substituída pela letra “i” e não há quebra de linha;

- Com o Cuneiform um “ç” foi trocado por um “q” e uma crase foi omitida;
- Com o GORC a letra “l” foi completamente substituída pelo caractere “_”, e as palavras ficaram com espaçamentos indevidos.
- Com o OCRAD a letra “l” foi substituída pelo caractere “_” e não houve reconhecimento do “ç” e de caracteres com acentuação gráfica.

Tabela 5.1: Resultado do teste dos softwares OCR relativo à figura 5.1

OCR	Acertos (de 51)
Tesseract	51
Cuneiform	51
GOCR	50
OCRAD	43

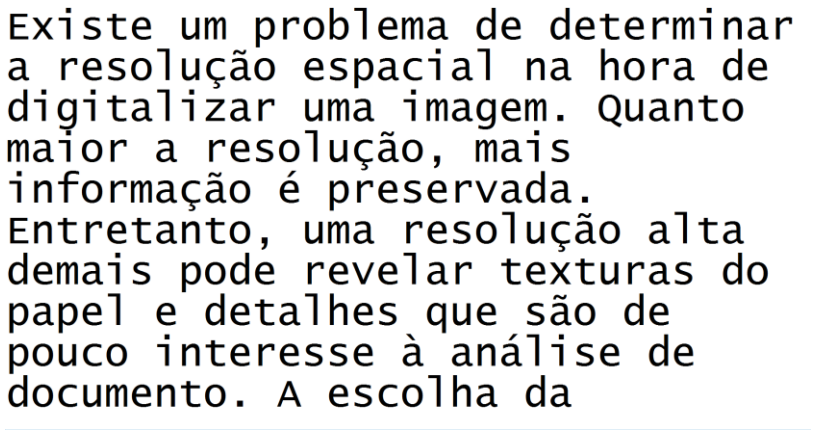


Figura 5.1: Captura de tela de um pedaço de documento em 1897x995

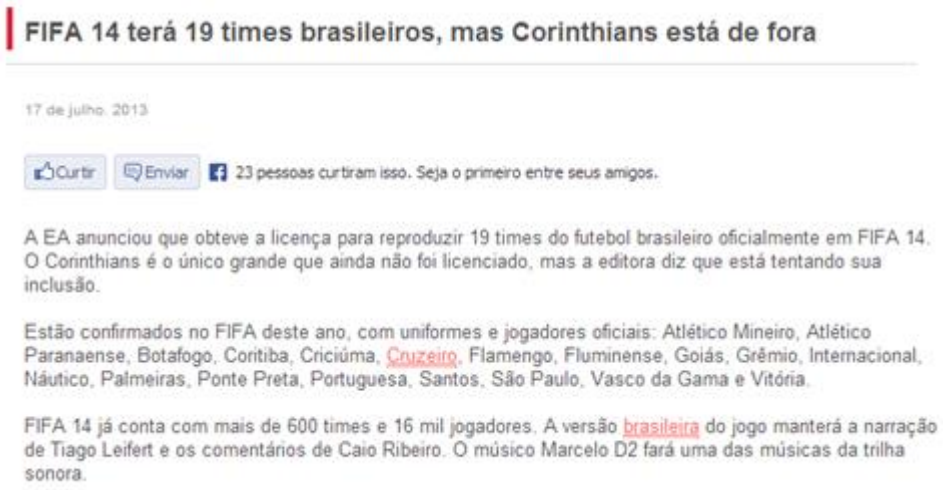
O segundo teste foi feito a partir da captura de tela de uma imagem (Figura 5.2) que apresenta texto em diferentes cores e fontes. Sobre os resultados da Tabela 5.2 observa-se:

- O Tesseract fez substituições de vírgulas pelos caracteres "_" e ".", não detectou duas palavras com cor diferente e confundiu "os caracteres "fi" com "n" e "Fl" "com "N";
- O Cuneiform trocou ou não detectou as pontuações, confundiu “ri” com “n” em 3 palavras, e não detectou duas palavras com cor diferente;
- O GOCR trocou algumas pontuações, inseriu espaçamentos e caracteres indevidos e fez trocas de caracteres como "N" por "rJ" e "ç" por "c_", entre outras;
- O OCRAD fez a troca das vírgulas por outros caracteres e teve dificuldade em acertar palavras de fonte menor, porém foi o único capaz de detectar as palavras

"Cruzeiro" e "brasileira", que não se encontram na cor preta no texto.

Tabela 5.2: Resultado do teste dos softwares OCR relativo à figura 5.2

OCR	Acertos (de 168)
Tesseract	154
Cuneiform	146
GOCR	146
OCRAD	124



FIFA 14 terá 19 times brasileiros, mas Corinthians está de fora

17 de julho, 2013

Curtir Enviar 23 pessoas curtiram isso. Seja o primeiro entre seus amigos.

A EA anunciou que obteve a licença para reproduzir 19 times do futebol brasileiro oficialmente em FIFA 14. O Corinthians é o único grande que ainda não foi licenciado, mas a editora diz que está tentando sua inclusão.

Estão confirmados no FIFA deste ano, com uniformes e jogadores oficiais: Atlético Mineiro, Atlético Paranaense, Botafogo, Coritiba, Criciúma, Cruzeiro, Flamengo, Fluminense, Goiás, Grêmio, Internacional, Náutico, Palmeiras, Ponte Preta, Portuguesa, Santos, São Paulo, Vasco da Gama e Vitória.

FIFA 14 já conta com mais de 600 times e 16 mil jogadores. A versão brasileira do jogo manterá a narração de Tiago Leifert e os comentários de Caio Ribeiro. O músico Marcelo D2 fará uma das músicas da trilha sonora.

Figura 5.2: Captura de tela em um texto com diferentes cores, fontes e imagem em 659x345

O terceiro teste é um texto de um rótulo de remédio (Figura 5.3) capturado com *webcam*, e apresenta distorção devido à curvatura da superfície onde se encontra. Sobre os resultados da Tabela 5.3 que se encontra mais abaixo, observa-se que o Tesseract, apesar de cometer muitos erros, foi o programa que apresentou mais acertos dentre os quatro aqui testados. Os demais programas apresentaram uma quantidade de acertos muito baixa e, consequentemente, um texto incompreensível.

Tabela 5.3: Resultado do teste dos softwares OCR relativo à figura 5.3

OCR	Acertos (de 53)
Tesseract	19
Cuneiform	4
GOCR	4
OCRAD	2

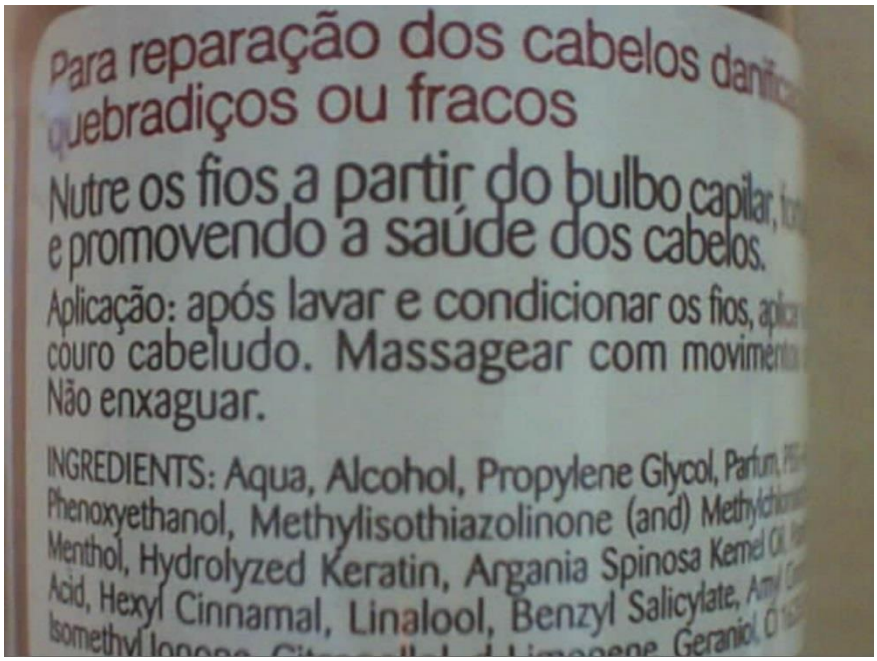


Figura 5.3: Rótulo de remédio capturado com *webcam* em 1920x1440

O quarto teste é uma página de mangá (revista em quadrinho japonesa) capturada com *webcam* (Figura 5.4). Sobre os resultados apresentados na Tabela 5.4, acrescenta-se que:

- O Tesseract confundiu um “S” por um “5” e um “O” por um “0”;
- O Cuneiform confundiu oito vezes o caractere “S” com o “5”, duas vezes o “U” com “LI”, uma vez o “G” com “C” e uma vez o “O” com “D”;
- O GOCR confundiu duas vezes o caractere “S” com “$” e duas vezes o “I” com “L”;
- O OCRAD confundiu uma vez a letra “T” com um a letra “r”, uma o “C” com o “G”, duas vezes o “D” com o “O”, duas vezes o “H” com “N” e uma “S” com o “5”.

Tabela 5.4: Resultado do teste dos softwares OCR relativo à figura 5.4

OCR	Acertos (de 47)
Tesseract	30
Cuneiform	27
GOCR	14
OCRAD	21

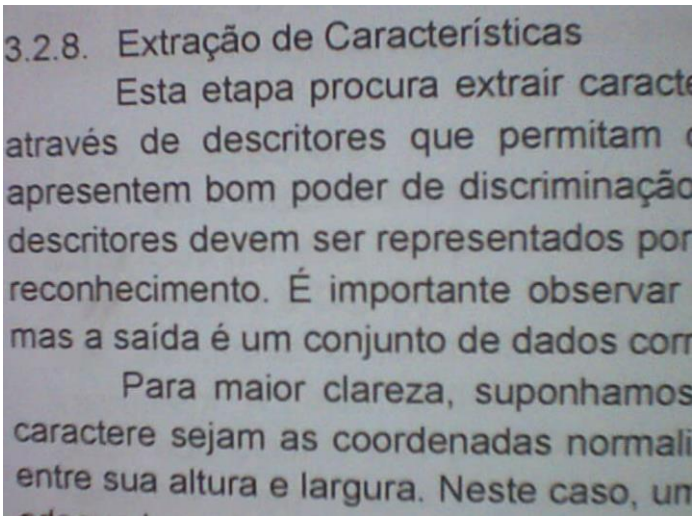
Figura 5.4: Página de mangá capturado com *webcam* em 1024x768

O quinto teste foi feito com a captura de um documento por *webcam* (Figura 5.5). Os acertos de cada programa constam na Tabela 5.5, acrescentando-se as seguintes observações:

- O Tesseract trocou o caractere "," por "." em dois momentos;
- O Cuneiform trocou o que devereia ser "3.2.8." por "p.g.g." ;
- O GOCR teve muitos problemas no processo, tornando o texto bastante confuso;
- O OCRAD teve problemas para identificar caracteres com acentuação gráfica e também o "ç".

Tabela 5.5: Resultado do teste dos softwares OCR relativo à figura 5.5

OCR	Acertos (de 57)
Tesseract	55
Cuneiform	56
GOCR	31
OCRAD	45


3.2.8. Extração de Características

Esta etapa procura extrair caract
através de descritores que permitam
apresentem bom poder de discriminaçã
descritores devem ser representados por
reconhecimento. É importante observar
mas a saída é um conjunto de dados cor

Para maior clareza, suponhamos
caractere sejam as coordenadas normali
entre sua altura e largura. Neste caso, un

Figura 5.5: Página de documento capturado com *webcam* em 1024x768

O sexto teste é uma página de mangá (revista em quadrinho japonesa) didático capturada com *webcam* (Figura 5.6) e apresenta quatro blocos de texto, em vez de um, como até então vinha sendo apresentado. Os resultados podem ser vistos na Tabela 5.6.

Tabela 5.6: Resultado do teste dos softwares OCR relativo à figura 5.6

OCR	Acertos (de 46)
Tesseract	20
Cuneiform	8
GOCR	2
OCRAD	2

Figura 5.6: Página de mangá didático capturado com *webcam* em 1024x768

Para melhor visualização dos resultados, a tabela 5.7 contém a porcentagem dos acertos de cada OCR para cada imagem aqui apresentada:

Tabela 5.7: Síntese dos resultados dos testes dos softwares OCR

Software OCR	Porcentagem de acertos de palavras					
	Figura 5.1	Figura 5.2	Figura 5.3	Figura 5.4	Figura 5.5	Figura 5.6
Tesseract	100%	91,7%	35,8%	63,8%	96,5%	43,5%
Cuneiform	100%	86,9%	7,5%	57,4%	98,2%	17,4%
GOCR	98%	86,9%	7,5%	29,8%	54,4%	4,3%
OCRAD	84,3%	73,8%	3,8%	44,7%	78,9%	4,3%

Com base nos resultados reunidos na tabela acima, percebe-se que Tesseract foi o que obteve os melhores resultados mesmo nas situações mais adversas como as encontradas nas figuras 5.3 e 5.6. O Cuneiform obteve resultados consideravelmente satisfatórios, conseguindo separar os textos de saída em coluna conforme captado durante a análise, mas a captação dos caracteres, de modo geral, não foi tão boa quanto à do Tesseract. O GOCR ficou em terceiro lugar em termos de resultados, mas foi o que mais conseguiu detectar caracteres durante as análises. O OCRAD foi o que se saiu pior, tendo os piores resultados e falhando em captar caracteres na maioria dos casos. Sendo assim o Tesseract foi o *software* de reconhecimento de caracteres utilizado nos testes com a placa BeagleBoard.

5.3 Testes na BeagleBoard

Para realização dos testes na BeagleBoard com a câmera do xLupa embarcado, foram utilizados materiais que seriam usados no cotidiano pelo público alvo deste projeto como jornais, livros, apostilas e revistas em quadrinhos. Os testes na BeagleBoard foram realizados utilizando imagens capturadas pelo xLupa embarcado com uso de uma *webcam*, todos na resolução 1280x720.

O desempenho do OCR executando na BeagleBoard foi analisado em termos de tempo de execução. Para medir o tempo, foi utilizado o comando no *shell* do Linux "time" antes do comando de execução do OCR.

Como o Linux é um sistema de tempo compartilhado, diversos processos compartilham o acesso a CPU, podendo interferir no desempenho do OCR avaliado, em termos de tempo de execução. Para medir o tempo de execução de forma precisa, deve-se, portanto, realizar os testes com o mínimo de tarefas desnecessárias à aplicação abertas, quanto possível.

O Tesseract é um programa executado através de linha de comando, então tudo é feito através de um terminal ou janela de *prompt* de comando. O comando é basicamente este:

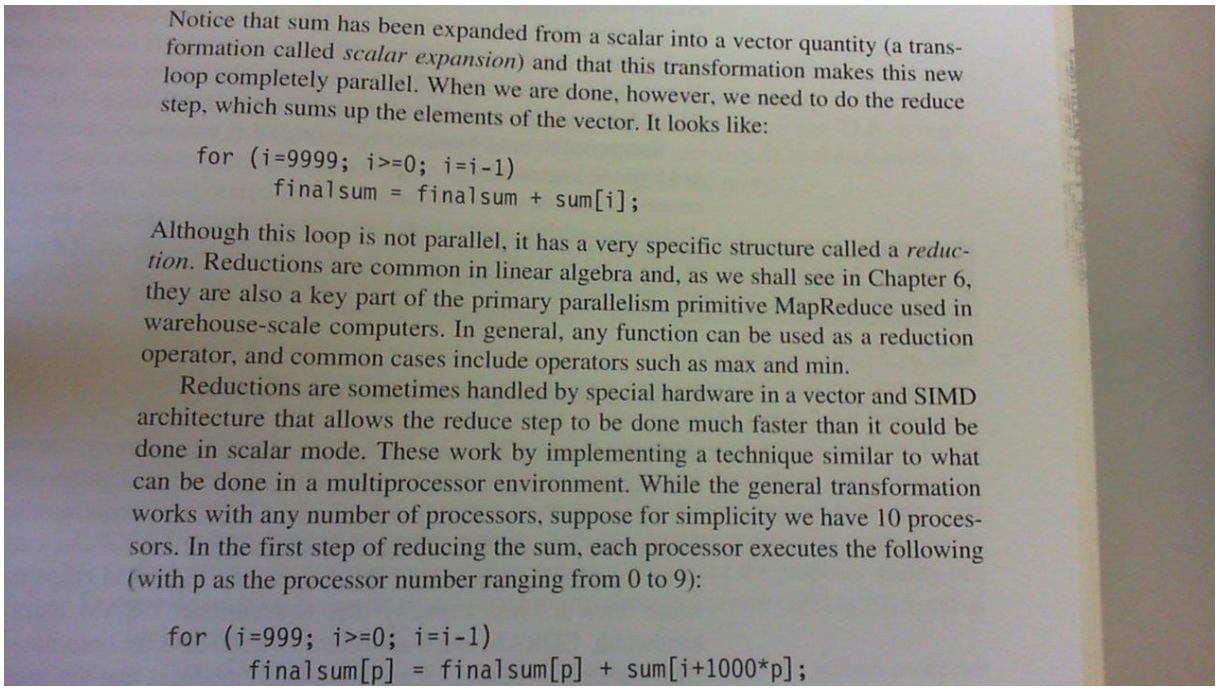
tesseract imagename outputbase [-l lang]

O parâmetro -l representa o dicionário da linguagem determinada pelo usuário, a fim de aprimorar o resultado do reconhecimento das palavras no texto analisado. Por padrão, se nenhum dicionário for especificado, o inglês é assumido. Múltiplas linguagens podem ser especificadas, separadas pelo caractere "+". Assim, para ler uma imagem chamada "teste.ppm" com texto em português e salvar o resultado em "teste.txt", o comando seria:

tesseract teste.ppm teste -l por

A seguir, são apresentados os resultados de algumas das imagens avaliadas, junto com o tempo que o Tesseract levou para ler as imagens e comentários referentes aos resultados obtidos.

A Figura 5.7 apresenta um texto de um livro de Organização e Arquitetura de Computadores, escrito em inglês. O objetivo dessa captura foi tentar capturar uma página inteira de texto para analisar o tempo de processamento na placa BeagleBoard-XM, entretanto isto acarretou o problema de que os caracteres ficaram pequenos (com altura de 10 *pixels*), dificultando o reconhecimento de palavras do OCR.



Notice that sum has been expanded from a scalar into a vector quantity (a transformation called *scalar expansion*) and that this transformation makes this new loop completely parallel. When we are done, however, we need to do the reduce step, which sums up the elements of the vector. It looks like:

```
for (i=9999; i>=0; i=i-1)
      finalsum = finalsum + sum[i];
```

Although this loop is not parallel, it has a very specific structure called a *reduction*. Reductions are common in linear algebra and, as we shall see in Chapter 6, they are also a key part of the primary parallelism primitive MapReduce used in warehouse-scale computers. In general, any function can be used as a reduction operator, and common cases include operators such as max and min.

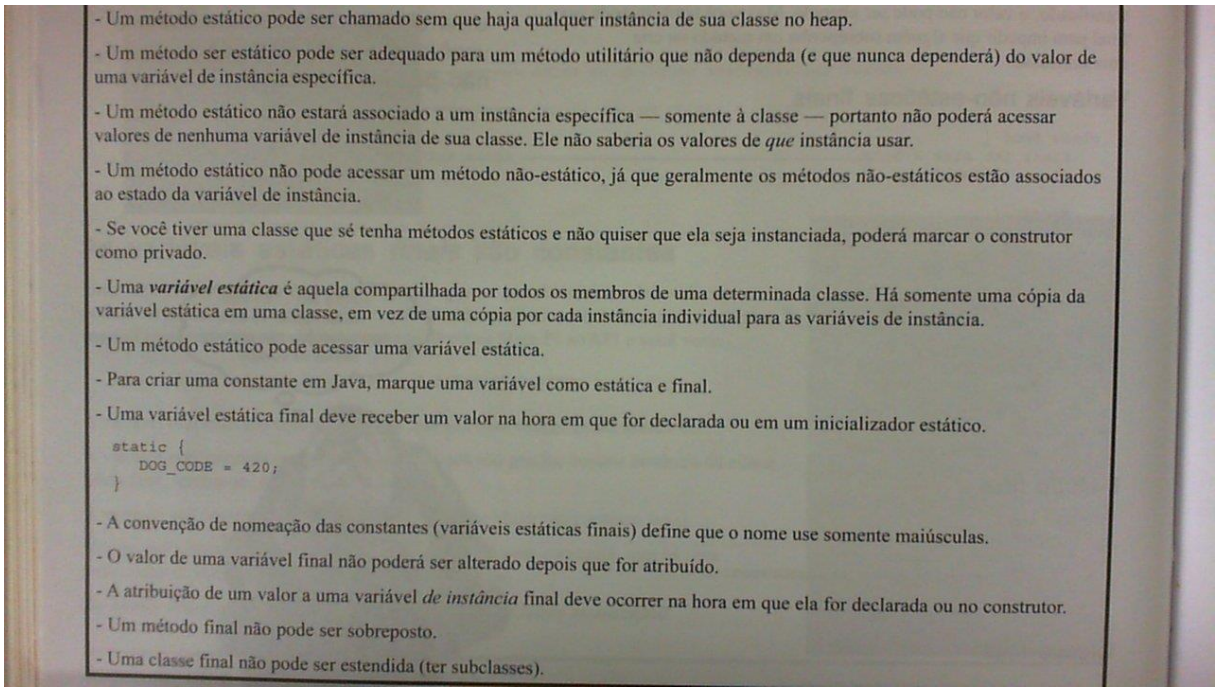
Reductions are sometimes handled by special hardware in a vector and SIMD architecture that allows the reduce step to be done much faster than it could be done in scalar mode. These work by implementing a technique similar to what can be done in a multiprocessor environment. While the general transformation works with any number of processors, suppose for simplicity we have 10 processors. In the first step of reducing the sum, each processor executes the following (with p as the processor number ranging from 0 to 9):

```
for (i=999; i>=0; i=i-1)
      finalsum[p] = finalsum[p] + sum[i+1000*p];
```

Figura 5.7: Página de um livro de Organização e Arquitetura de Computadores

- Tempo de processamento: 54 segundos.
- Número total de palavras: 299
- Número total de acertos: 226
- Observações: Ele ignorou completamente as duas primeiras linhas e por duas vezes gerou quebras de página no meio de uma linha de texto por culpa do enviesamento, não alocando o texto pós-quebra de página na linha seguinte, prejudicando a compreensão do texto. Além disso, ele confundiu o caractere "i" com "1`" nas ocasiões em que ele aparecia nos dois trechos de código.

A Figura 5.8 apresenta um texto de um livro de Java escrito em português com o mesmo objetivo da Figura 5.7, o de analisar o tempo de processamento na placa BeagleBoard-XM para processar uma página com bastante texto. Embora o texto dessa captura de tela tenha ficado bem alinhado, o OCR falhou no reconhecimento de algumas palavras por causa do tamanho dos caracteres, que assim como na imagem anterior, acabaram ficando pequenos.



- Um método estático pode ser chamado sem que haja qualquer instância de sua classe no heap.
- Um método ser estático pode ser adequado para um método utilitário que não dependa (e que nunca dependerá) do valor de uma variável de instância específica.
- Um método estático não estará associado a um instância específica — somente à classe — portanto não poderá acessar valores de nenhuma variável de instância de sua classe. Ele não saberia os valores de *que* instância usar.
- Um método estático não pode acessar um método não-estático, já que geralmente os métodos não-estáticos estão associados ao estado da variável de instância.
- Se você tiver uma classe que sé tenha métodos estáticos e não quiser que ela seja instanciada, poderá marcar o construtor como privado.
- Uma ***variável estática*** é aquela compartilhada por todos os membros de uma determinada classe. Há somente uma cópia da variável estática em uma classe, em vez de uma cópia por cada instância individual para as variáveis de instância.
- Um método estático pode acessar uma variável estática.
- Para criar uma constante em Java, marque uma variável como estática e final.
- Uma variável estática final deve receber um valor na hora em que for declarada ou em um inicializador estático.

```
static {
   DOG_CODE = 420;
}
```

- A convenção de nomeação das constantes (variáveis estáticas finais) define que o nome use somente maiúsculas.
- O valor de uma variável final não poderá ser alterado depois que for atribuído.
- A atribuição de um valor a uma variável *de instância* final deve ocorrer na hora em que ela for declarada ou no construtor.
- Um método final não pode ser sobreposto.
- Uma classe final não pode ser estendida (ter subclasses).

Figura 5.8: Página de um livro de Java

- Tempo de processamento: 1 minuto e 35 segundos.
- Número total de palavras: 321

- Número total de acertos: 238
- Observações: O número de acertos até a metade da página foi quase total, depois disso alguns caracteres, que até então estavam sendo corretamente reconhecidos, passaram a ser confundidos. O código contido entre as linhas do texto foi completamente perdido, uma vez que o Tesseract não acertou nenhum caractere.

A Figura 5.9 apresenta uma captura de tela a partir de uma página de Jornal.

Aumento de leitos sem planejamento em cidades como M

Expectativa é que redes substituam estabelecimentos menores em meio à concorrência maior

INGRID FAGUNDEZ
DE SÃO PAULO

Os dias depois da Copa do Mundo preocupam os setores hoteleiros de Manaus, Cuiabá e Belo Horizonte, que já trabalham com a ideia de superoferta de leitos depois de a bola parar de rolar.

Para empresários e associações, na concorrência que deve se instalar após o torneio, os mais prejudicados serão os pequenos hotéis.

Os empreendimentos menores seriam substituídos pelos de rede, em movimento parecido com o que ocorreu em São Paulo há dez anos.

As redes começaram a ir para Belo Horizonte, Manaus e Cuiabá após 2009, quando as sedes foram reveladas.

Desde então, o número de hotéis se multiplicou. Na capital de Mato Grosso, a quantidade de leitos deve pular de 12 mil, em 2009, para 17 mil durante a Copa —alta de 41%.

"Cuiabá tem a situação mais alarmante. Não tem uma economia pungente ou grandes atrativos turísticos. Nem fizemos estudos sobre ela porque não valia a pena", diz o presidente da consultoria especializada BSH, José Marino.

Ele vê Belo Horizonte e Manaus em um patamar melhor, pois têm um ambiente de negócios mais desenvolvido.

Nas cidades-sede, as construções em série foram consequência da baixa oferta de leitos quando os locais foram

Figura 5.9: Página de um jornal

- Tempo de processamento: 38 segundos.
- Número total de palavras: 283
- Número total de acertos: 279
- Observações: Nesta imagem ele apenas ignorou dois hifens e errou duas palavras (trocou "multiplicou" por "mulüplicou" e "pecializada" por "pecialimda"). Em textos apresentados em colunas, como o texto apresentado nesta imagem, o Tesseract detecta que o texto está apresentado neste formato e dispõe o texto gerado separado por colunas, em vez de converter linha por linha sem se importar com a coluna que o texto pertence. Graças a isso, a compreensão do texto gerado não é

prejudicada e o texto pode ser lido sem problemas por uma pessoa ou por um sintetizador de voz.

A Figura 5.10 apresenta um teste realizado a partir de uma página de um mangá didático de Banco de Dados.

Figura 5.10: Página de mangá didático

- Tempo de processamento: 12 segundos.
- Número total de palavras: 36
- Número total de acertos: 23
- Comentário: O programa acertou a maior parte das palavras, porém teve dificuldades para reconhecer a letra "S" em quase todas as vezes que esta apareceu e não reconheceu por completo duas palavras "então" e "está" substituindo-as por "ig emo" e "sem".

Muitos das falhas nas detecções de palavras nas figuras 5.7 e 5.8 foram devido ao tamanho da altura em pixels dos caracteres nas imagens, que em ambas possuem altura em torno de 10 *pixels*. Segundo as especificações [11], a recomendação para que o Tesseract reconheça caracteres com maestria é que eles tenham pelo menos 20 *pixels* de altura, sendo que abaixo

de 10 *pixels* as chances reconhecimento são baixíssimas, e abaixo de 8 *pixels* o Tesseract irá considerar os caracteres como ruídos e os ignorará.

5.4 Integração do OCR no xLupa Embarcado

A integração do OCR com o xLupa embarcado foi realizada por meio de modificações na função read_frame do código do xLupa embarcado, que realiza a captura através da *webcam*. Caso o usuário pressione o botão existente na placa, a imagem é capturada e é feita uma chamada do Tesseract OCR que gera um texto editável a partir desta imagem e depois uma chamada do eSpeak [29], um sintetizador de voz de código aberto que irá ler em voz o texto gerado pelo OCR. O eSpeak foi escolhido porque ele já é utilizado pelo xLupa desktop, por possuir suporte ao português-brasileiro. Além disso, o código binário está disponível nos repositórios do Ubuntu para a placa BeagleBoard. É importante frisar que o evento que ativa o OCR e o sintetizador de voz pode ser alterado para mouse ou teclado caso necessário.

A captura de imagens da *webcam* no Linux é realizada pelo V4L2 (Video For Linux 2), que é uma interface de programação para captura de vídeo no Linux e que impõe um padrão de comunicação para esta classe de dispositivos [30]. Além de uma biblioteca comum do espaço do usuário, V4L2 é também um conjunto de *device drivers* padronizados para o *kernel* do Linux, que fazem todo o trabalho de comunicação com o dispositivo.

A comunicação do aplicativo no lado do usuário com os drivers no lado do *kernel* é feita com chamadas *ioctl*. Estas chamadas são compostas por um identificador do dispositivo, um inteiro que indica a operação desejada e um ponteiro para os dados que serão transmitidos entre o usuário e o *kernel*. Mediante uma chamada *ioctl*, é enviado para o driver o formato de imagem desejada, as dimensões dela e outras informações. Neste trabalho foram usadas imagens em formato RGB24 de dimensões 1280x720 que é o formato padrão utilizado pelo xLupa Embarcado.

O processo de obtenção de imagens é realizado por meio da retirada do *buffer* da fila de saída do driver com chamada *ioctl* com o operador VIDIOC_DQBUF. Depois de utilizado, o *buffer* é colocado no fim da fila e o *device driver* trata de preencher o *buffer* com uma nova imagem através de uma chamada *ioctl* com o operador VIDIOC_QBUF.

O código que armazena a imagem capturada para envio ao Tesseract foi inserido entre as chamadas *ioctl* VIDIOC_DQBUF e VIDIOC_QBUF conforme apresentado na figura 5.11.

```
126     xioctl(fd, VIDIOC_DQBUF, &buf);
127     if (flagOCR == 1){
128             sprintf(out_name, "out%03d.ppm", i);
129             fout = fopen(out_name, "w");
130             if (!fout) {
131                     perror("Cannot open image");
132                     exit(EXIT_FAILURE);
133             }
134             fprintf(fout, "P6\n%d %d 255\n", WIDTH, HEIGHT);
135             fwrite(buffers[buf.index].start, buf.bytesused, 1, fout);
136             fclose(fout);
137             chamaOCR;
138             flagOCR = 0;
139     }
140     process_image ((unsigned char *)buffers[buf.index].start);
141     xioctl(fd, VIDIOC_QBUF, &buf);
142 }
143 void chamaOCR (){
144     system("tesseract out1.ppm out1 -l por");
145     system("espeak -f out1.txt -v mb-br4 -s 180");
146 }
```

Figura 5.11: Algoritmo de captura de tela

Na linha 126 é feita a chamada *ioctl* VIDEOC_DQBUF para desenfileirar o *buffer*, como explicado anteriormente.

Na linha 134 são preenchidas as informações do cabeçalho da imagem no formato Netpbm, onde P6 indica o tipo da imagem – neste caso, PPM, que significa *Portable PixMap* – a altura e a largura da imagem.

Na linha 135, é escrita no arquivo PPM a imagem armazenada no *buffer*.

O programa só executa essa ação caso uma *flag* chamada flagOCR esteja setada para 1, o que significa que o botão *user* da placa BeagleBoard foi pressionado. Após a captura de tela ser realizada, é feita uma chamada para a função chamaOCR para executar o OCR e o sintetizador de voz através da função *system*, que executa um comando do *shell* do Linux.

Assim como o Tesseract, o eSpeak também é um programa executado através de linha de comando. Dessa forma, para ler um texto chamado "out1.txt" o comando seria:

espeak -f out1.txt -v mb-br4 -s 180

Onde -v determina uma voz a ser usada, neste caso "mb-br4" e -s determina a velocidade de leitura em palavras por minuto, neste caso setada para 180 (a velocidade padrão é 175). Esta velocidade foi escolhida por ter apresentado resultado satisfatório em termos de compreensão da dicção durante testes realizados.

5.5 Pré-Processamento antes do OCR

Nos testes apresentados no subcapítulo 5.4 foi verificado que o processamento de uma página pode levar em torno de um minuto e meio para ser processada pelo Tesseract na placa BeagleBoard-XM. Para diminuir o tempo de resposta para o usuário, foi feita uma modificação no algoritmo de captura de tela para que ele divida a imagem em três partes, levando para cada pedaço da imagem, um terço do tempo que levaria para processar a imagem inteira. A versão modificada do algoritmo pode ser vista na figura 5.12.

```
307             xioctl(fd, VIDIOC_DQBUF, &buf);
308             if (flagOCR == 1){
309                     sprintf(out_name, "out1.ppm");
310                     fout = fopen(out_name, "w");
311                     if (!fout) {
312                             perror("Cannot open image");
313                             exit(EXIT_FAILURE);
314                     }
315                     fprintf(fout, "P6\n%d %d 255\n", WIDTH, 240);
316                     fwrite(buffers[buf.index].start, buf.bytesused, 1, fout);
317                     fclose(fout);
318
319                     sprintf(out_name, "out2.ppm");
320                     fout = fopen(out_name, "w");
321                     if (!fout) {
322                             perror("Cannot open image");
323                             exit(EXIT_FAILURE);
324                     }
325                     fprintf(fout, "P6\n%d %d 255\n", WIDTH, 240);
326                     fwrite(&(buffers[buf.index].start[1280*240*3]), buf.bytesused, 1, fout);
327                     fclose(fout);
328
329                     sprintf(out_name, "out3.ppm");
330                     fout = fopen(out_name, "w");
331                     if (!fout) {
332                             perror("Cannot open image");
333                             exit(EXIT_FAILURE);
334                     }
335                     fprintf(fout, "P6\n%d %d 255\n", WIDTH, 240);
336                     fwrite(&(buffers[buf.index].start[1280*240*3*2]), buf.bytesused, 1, fout);
337                     fclose(fout);
338
339                     chamaOCR();
340                     flagOCR = 0;
341             }
342             process_image ((unsigned char *)buffers[buf.index].start);
343       xioctl(fd, VIDIOC_QBUF, &buf);
344   }
345   void chamaOCR (){
346             system("tesseract out1.ppm out1 -l por");
347             system("espeak -f out1.txt -v mb-br4 -s 180");
348
349             system("tesseract out2.ppm out2 -l por");
350             system("espeak -f out2.txt -v mb-br4 -s 180");
351
352             system("tesseract out3.ppm out3 -l por");
353             system("espeak -f out3.txt -v mb-br4 -s 180");
354   }
```

Figura 5.12: Algoritmo de captura de tela modificado e chamadas para o OCR e o sintetizador de voz

Para essa alteração, foi necessário repetir duas vezes o bloco de código que gerava a imagem, alterando a informação de altura da imagem no cabeçalho de cada imagem a ser

gerada para 240 (um terço de 720, que era a altura original), e modificar o *pixel* de ponto de partida a ser usado para geração da segunda e terceira imagem. Este *buffer* enxerga a imagem não como uma matriz de *pixels*, mas como um vetor unidimensional, portanto para percorrer linhas é preciso realizar uma multiplicação de forma que ele salte todos *pixels* correspondentes às linhas e colunas da imagem anterior. Na linha 326 é ordenado que ele comece a varrer o buffer a partir da linha 241. Cada pixel possui 3 *bytes*, que servem para guardar as informações de R, G e B da imagem. Assim, para começar a varrer o *buffer* a partir da linha 241, é passado como índice do vetor de *buffer* a expressão *1280*240*3*, que é a multiplicação do número de linhas, colunas e *bytes* de 1/3 da imagem. Analogamente, o mesmo é feito para gerar a terceira imagem, com a diferença que agora a imagem nova deve ser gerada a partir de 2/3 da imagem armazenada no *buffer*, então basta multiplicar por 2 a expressão anteriormente descrita.

Entretanto, estes recortes podem ocorrer em uma linha que passe pelo meio das palavras, danificando o texto a ser gerado pelo OCR. Para evitar que isto ocorra, uma segunda alteração foi realizada para verificar se a linha a ser cortada contem apenas pixels brancos. Este tratamento pode ser visto nas figuras 5.13, 5.14 e 5.15.

A ideia básica deste algoritmo é percorrer as colunas das linhas que demarcarão o primeiro corte da imagem, verificando se existem pixels pretos nela. Caso existam, significa que esta linha passa por uma palavra e ela não deve ser cortada, então o algoritmo desce para a próxima linha e repete o processo até encontrar uma linha que não tenha pixels pretos, significando que ela é ideal para servir de corte.

Na linha 379 o comando *for* que percorre as linhas inferiores do primeiro tercil em busca de *pixels* pretos. Foi definida uma margem de erro de 20 *pixels* tanto para cima quanto para baixo da linha que representa a altura de 1/3 da imagem. Assim, quando existirem caracteres exatamente nessa altura, ele realizará o corte antes dela caso as letras sejam pequenas, ou para conseguir pular uma quantidade suficiente de linhas caso as letras sejam grandes.

Na linha 381, o comando *for* que percorrerá as colunas da matriz de *pixel*.

Na linha 383, a variável do tipo inteiro “k” recebe a média da soma dos valores RGB do pixel analisado.

Na linha 384 é verificado o valor de “k”, sendo que quanto mais próximo de zero for o resultado, mais preta é a cor deste pixel e quanto mais próximo de 255, mais branca é a cor

deste *pixel*. O limiar escolhido para definir o que será considerado como *pixel* preto ou branco foi o valor 140, que mediante testes realizados, apresentou resultados mais satisfatórios.

Caso o pixel seja preto, um contador é incrementado. No final do laço, caso este contador esteja com um valor maior que zero, significa que a linha verificada continha *pixels* pretos, significando que ela não é ideal para ser cortada, e inicia-se a verificação nos *pixels* da linha de baixo. Caso o contador se mantenha zerado, a variável "*crop*" recebe o valor da variável "i", que representa a linha verificada e que será utilizada para realizar o corte na imagem. A variável "temp" também recebe o valor desta linha, para uso na verificação da altura do próximo corte.

Após encontrado a linha que servirá de corte, um *break* é efetuado para sair do laço e parar a busca. Os passos a seguir são parecidos com os da versão anterior do algoritmo, com a diferença que agora a altura da imagem definida no cabeçalho não está mais setada para 240, mas para o valor da altura encontrada, que estava guardada na variável "*crop*", e o campo que representa o tamanho do arquivo foi alterado para "*crop**1280*3", para que o tamanho seja equivalente ao número de linhas e colunas utilizados até a altura definida para o corte.

```
378     //define o tamanho do primeiro crop
379     for(i = 219; i<=259;i++){
380         contador = 0;
381         for(j=0;j<=1279;j+=3){
382             p =  (unsigned char*)buffers[buf.index].start;
383             k = (p[i*1280*3 + j] + p[i*1280*3 + j + 1] + p[i*1280*3 + j + 2])/3;
384             if (k <= 140)
385                 contador = contador + 1;
386         }
387         if (contador == 0){
388             crop = i;
389             temp = crop;
390             break;
391         }
392         crop = i;
393         temp = crop;
394     }
395
396         //crop de 1/3 da imagem
397         sprintf(out_name, "out1.ppm");
398         fout = fopen(out_name, "w");
399         if (!fout) {
400             perror("Cannot open image");
401             exit(EXIT_FAILURE);
402         }
403         fprintf(fout, "P6\n%d %d 255\n", WIDTH, crop);
404         fwrite(buffers[buf.index].start, crop*1280*3, 1, fout);
405         fclose(fout);
```

Figura 5.13: Determinação da altura do primeiro recorte

O algoritmo que define a altura do segundo corte funciona de forma análoga ao anterior, com a diferença que agora a busca pela linha a ser cortada é realizada em torno do limite inferior do segundo tercil da imagem, e o valor da nova linha a ser cortada não é passado para a variável "temp". Em vez disso, no campo do cabeçalho que define a altura da imagem, é subtraído o valor da altura do corte atual com a do corte anterior, para definir qual é a altura dessa segunda imagem na linha 430. De forma similar, a variável "temp" também é usada na conta que define o tamanho do arquivo a ser escrito, na linha 431.

```
408     //segundo crop
409     for(i = 459; i<=499;i++){
410         contador = 0;
411         for(j=0;j<=1279;j+=3){
412             p =  (unsigned char*)buffers[buf.index].start;
413             k = (p[i*1280*3 + j] + p[i*1280*3 + j + 1] + p[i*1280*3 + j + 2])/3;
414             if (k <= 140)
415                 contador = contador + 1;
416         }
417         if (contador == 0){
418             crop = i; //agora crop guarda a altura do crop da segunda imagem
419             break;
420         }
421         crop = i;
422     }
423
424         sprintf(out_name, "out2.ppm");
425         fout = fopen(out_name, "w");
426         if (!fout) {
427             perror("Cannot open image");
428             exit(EXIT_FAILURE);
429         }
430         fprintf(fout, "P6\n%d %d 255\n", WIDTH, crop - temp);
431         fwrite(&(buffers[buf.index].start[temp*1280*3]), (crop - temp)*1280*3, 1, fout);
432         fclose(fout);
```

Figura 5.14: Determinação da altura do segundo recorte

Por fim, a terceira imagem não precisa realizar a verificação da altura do corte, mas apenas a definição da linha que representará o início da imagem, o que acontece dentro do índice do vetor na linha 443, que multiplica o número de linhas usadas até o recorte anterior pela largura da imagem e pelos três *bytes* que cada pixel contém. No cabeçalho, a altura da imagem fica sendo a altura total da imagem menos a altura do recorte anterior e o tamanho do arquivo fica sendo a altura total da imagem menos a altura do recorte anterior, multiplicado pela largura da imagem e pelos 3 *bytes* de cada *pixel*.

```
435         //terceiro crop
436             sprintf(out_name, "out3.ppm");
437             fout = fopen(out_name, "w");
438             if (!fout) {
439                 perror("Cannot open image");
440                 exit(EXIT_FAILURE);
441             }
442             fprintf(fout, "P6\n%d %d 255\n", WIDTH, HEIGHT - crop);
443             fwrite(&(buffers[buf.index].start[crop*1280*3]), (HEIGHT - crop)*1280*3, 1, fout);
444             fclose(fout);
445 
446             chamaOCR();
447             flagOCR = 0;
448         }
449 
450         process_image ((unsigned char *)buffers[buf.index].start);
451 
452       if (-1 == xioctl (fd, VIDIOC_QBUF, &buf))
453             errno_exit ("VIDIOC_QBUF");
454         return 1;
455     }
```

Figura 5.15: Determinação da altura do terceiro recorte

É importante notar que a execução do Tesseract e do eSpeak são feitas de forma sequencial, o que significa que o OCR só irá gerar o texto da próxima imagem depois que o sintetizador de voz concluir a leitura do texto da imagem anterior. Sendo assim, seria possível implementar a execução do Tesseract e do eSpeak de forma paralela, de forma que o OCR geraria os próximos textos durante o tempo em que o eSpeak estivesse executando. Também é preciso notar que o pré-processamento realizado para determinar a altura dos recortes trabalha com textos escritos em cores escuras sobre fundos claros. Uma modificação para verificar se a imagem possui textos escritos em cores claras sobre fundos escuros pode ser efetuada antes da busca pela linha de corte, de forma que o algoritmo da determinação da altura do recorte funcionaria de forma similar, alterando apenas a verificação da cor do pixel, que agora verificaria se a linha contem *pixels* brancos.

A seguir, é feito uma comparação dos resultados das figuras 5.7, 5.8. 5.9 e 5.10, com o uso do algoritmo da determinação da altura do recorte e sem o uso desse algoritmo.

Os resultados dos recortes da Figura 5.7 podem ser vistos nas figuras 5.16 à 5.21. No primeiro recorte em cima da Figura 5.7, assim como havia acontecido no teste realizado utilizando a imagem sem recortes, o OCR ignorou completamente as duas primeiras linhas, resultando em menos texto a ser convertido. Entretanto, o segundo recorte sem uso do algoritmo acabou ficando com mais texto, resultando em um tempo maior de processamento se comparado com o segundo recorte com o uso do algoritmo, como apresentado nos tempos de processamento representados na Tabela 5.8. Vale notar que o alto grau de enviesamento no

texto dessa imagem prejudicou a taxa de acertos de palavras, visto que tanto com o uso do algoritmo quanto utilizando altura fixa, algumas palavras no final da linha serão recortadas. Isto acontece, pois na altura fixa o recorte ocorre na altura 240 sem se importar se havia texto, enquanto que o recorte feito com o uso do algoritmo que determina a altura, quando não encontra uma linha inteira sem pixels pretos dentro do limite de altura estipulado (20% abaixo e acima da altura de 240 *pixels*), ele recorta na altura máxima de 260 *pixels*, para o caso do primeiro recorte. Uma possível solução para este problema seria aplicar métodos de transformações geométrias para corrigir o grau de enviesamento. Sendo assim, o método do recorte só é recomendado para imagens com texto enviesado, desde que a altura do espaço entre as linhas seja grande o suficiente para não prejudicar o recorte, o que deve acontecer desde o texto não esteja enviesado demais e que a captura seja feita a partir de uma distância adequada, que não deixe os caracteres com menos de 15 *pixels* de altura.

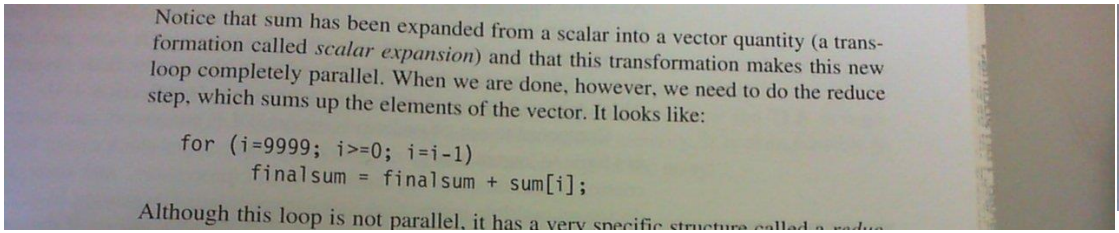
Tabela 5.8: Resultados dos recortes da Figura 5.7

	Tempo de processamento			
	Primeiro tercil	Segundo tercil	Terceiro tercil	Total
Com algoritmo	8 s	29 s	18 s	55 s
Sem algoritmo	8 s	33 s	20 s	61 s
Inteira				54 s

Os recortes da Figura 5.8 podem ser vistos nas figuras 5.22 à 5.27. No primeiro recorte da Figura 5.8, coube mais texto na versão com o uso do algoritmo que determina a altura de recorte do que na versão sem o uso deste algoritmo, resultando em um tempo maior de processamento. Já nos outros dois recortes, a versão com o uso do algoritmo acabou alocando menos palavras, resultando em um tempo menor de processamento, como pode ser visto na Tabela 5.9. Nessa imagem ficaram mais evidentes os benefícios de se usar o algoritmo para determinar a altura do recorte do que fazer o recorte na altura fixa, pois o texto foi recortado no final do primeiro tercil e início do segundo tercil, prejudicando o resultado gerado pelo OCR.

Tabela 5.9: Resultados dos recortes da Figura 5.8

	Tempo de processamento			
	Primeiro tercil	Segundo tercil	Terceiro tercil	Total
Com algoritmo	37 s	31s	29 s	1m35s
Sem algoritmo	36 s	34 s	30 s	1m38s
Inteira				1m33s

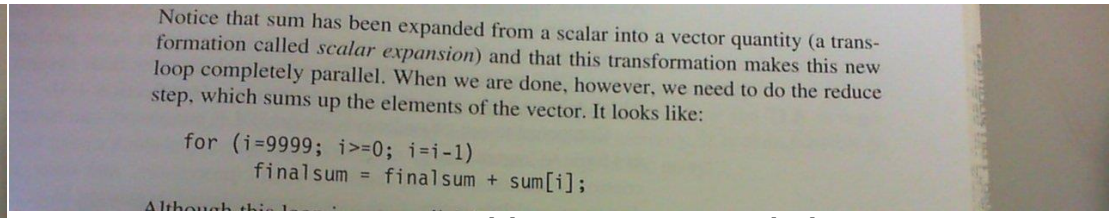


Notice that sum has been expanded from a scalar into a vector quantity (a transformation called *scalar expansion*) and that this transformation makes this new loop completely parallel. When we are done, however, we need to do the reduce step, which sums up the elements of the vector. It looks like:

```
for (i=9999; i>=0; i=i-1)
      finalsum = finalsum + sum[i];
```

Although this loop is not parallel, it has a very specific structure called a *reduc-*

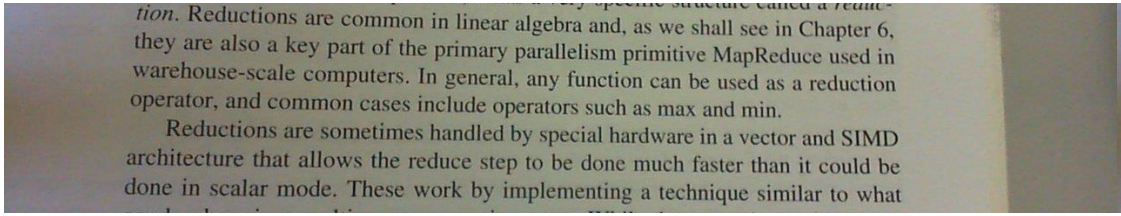
Figura 5.16: Primeiro tercil da Figura 5.7 com o uso do algoritmo



Notice that sum has been expanded from a scalar into a vector quantity (a transformation called *scalar expansion*) and that this transformation makes this new loop completely parallel. When we are done, however, we need to do the reduce step, which sums up the elements of the vector. It looks like:

```
for (i=9999; i>=0; i=i-1)
      finalsum = finalsum + sum[i];
```

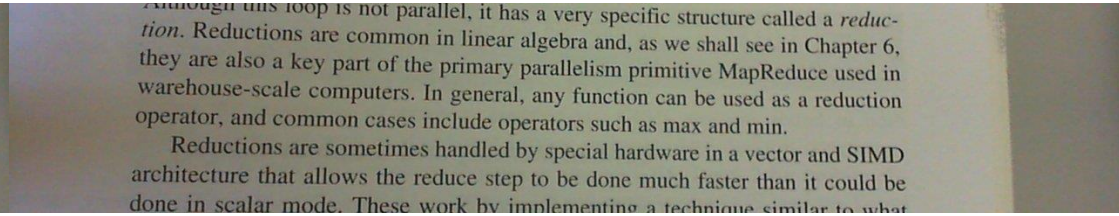
Figura 5.17: Primeiro tercil da Figura 5.7 sem o uso do algoritmo



tion. Reductions are common in linear algebra and, as we shall see in Chapter 6, they are also a key part of the primary parallelism primitive MapReduce used in warehouse-scale computers. In general, any function can be used as a reduction operator, and common cases include operators such as max and min.

Reductions are sometimes handled by special hardware in a vector and SIMD architecture that allows the reduce step to be done much faster than it could be done in scalar mode. These work by implementing a technique similar to what

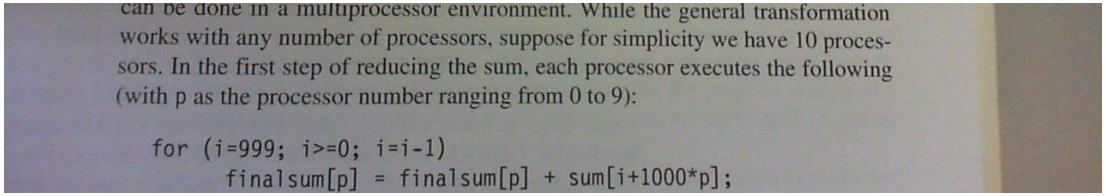
Figura 5.18: Segundo tercil da Figura 5.7 com o uso do algoritmo



Although this loop is not parallel, it has a very specific structure called a *reduction*. Reductions are common in linear algebra and, as we shall see in Chapter 6, they are also a key part of the primary parallelism primitive MapReduce used in warehouse-scale computers. In general, any function can be used as a reduction operator, and common cases include operators such as max and min.

Reductions are sometimes handled by special hardware in a vector and SIMD architecture that allows the reduce step to be done much faster than it could be done in scalar mode. These work by implementing a technique similar to what

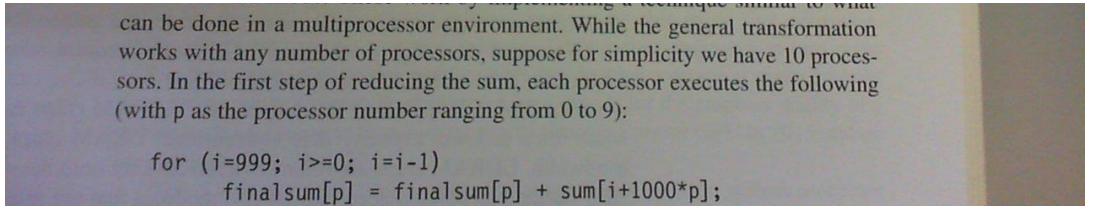
Figura 5.19: Segundo tercil da Figura 5.7 sem o uso do algoritmo



can be done in a multiprocessor environment. While the general transformation works with any number of processors, suppose for simplicity we have 10 processors. In the first step of reducing the sum, each processor executes the following (with p as the processor number ranging from 0 to 9):

```
for (i=999; i>=0; i=i-1)
      finalsum[p] = finalsum[p] + sum[i+1000*p];
```

Figura 5.20: Terceiro tercil da Figura 5.7 com o uso do algoritmo



can be done in a multiprocessor environment. While the general transformation works with any number of processors, suppose for simplicity we have 10 processors. In the first step of reducing the sum, each processor executes the following (with p as the processor number ranging from 0 to 9):

```
for (i=999; i>=0; i=i-1)
      finalsum[p] = finalsum[p] + sum[i+1000*p];
```

Figura 5.21: Terceiro tercil da Figura 5.7 sem o uso do algoritmo

As figuras 5.28 à 5.33 representam os recortes da Figura 5.9. Nestes recortes houve pouca variação no número de palavras por recorte, resultando em pouca variação no tempo de processamento deles, como pode ser visto na Tabela 5.10. O tempo maior se comparado com teste feito sobre a imagem inteira deve-se a concorrência pelo uso da CPU com o Xorg (a interface gráfica do Ubuntu) na hora de processar os recortes, em que o Xorg estava consumindo muito processamento.

Tabela 5.10: Resultados dos recortes da Figura 5.9

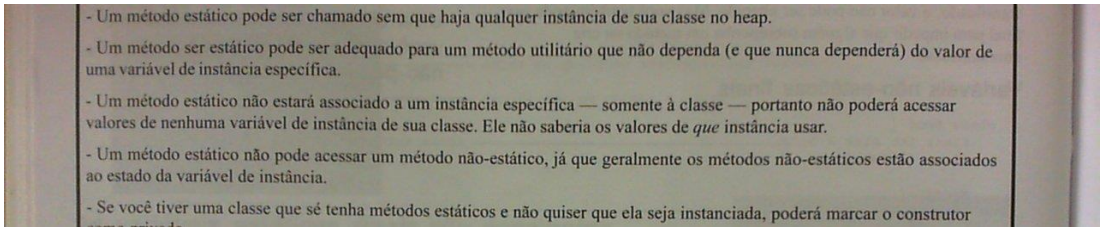
	Tempo de processamento			
	Primeiro tercil	Segundo tercil	Terceiro tercil	Total
Com algoritmo	11 s	20 s	19 s	50 s
Sem algoritmo	11 s	20 s	21 s	52 s
Inteira				38 s

As figuras 5.34 à 5.39 representam os recortes feitos em cima da Figura 5.10. Quanto aos tempos de processamento, na Tabela 5.11 é possível ver que não houve grande diferença entre os dois métodos de recorte, uma vez que a imagem tinha pouco texto e houve pouca variação no número de palavras nos recortes com uso do algoritmo para determinar a altura e no de altura fixa.

Tabela 5.11: Resultados dos recortes da Figura 5.10

	Tempo de processamento			
	Primeiro tercil	Segundo tercil	Terceiro tercil	Total
Com algoritmo	9 s	5 s	1 s	15 s
Sem algoritmo	7 s	7 s	1 s	15 s
Inteira				11 s

Como pode ser visto nas figuras 5.9 e 5.10, o método do recorte não é útil para textos divididos em coluna, pois textos neste formato requerem leitura por blocos de colunas. Quando o algoritmo recorta a imagem, ele divide as colunas em três partes, o que tiraria todo o sentido do texto na hora que o sintetizador de voz fosse ler o texto gerado a partir destes recortes, pois cada bloco estaria descontinuado, uma vez que o resto de seu texto estaria no próximo recorte. Uma solução para este problema seria aplicar métodos de segmentação de imagem como o algoritmo RLS, antes de efetuar os recortes.


- Um método estático pode ser chamado sem que haja qualquer instância de sua classe no heap.

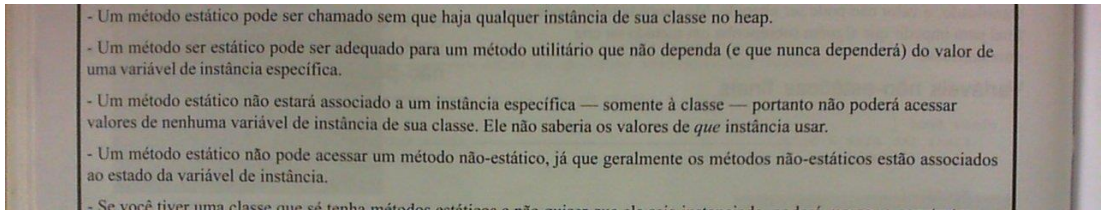
- Um método ser estático pode ser adequado para um método utilitário que não dependa (e que nunca dependerá) do valor de uma variável de instância específica.

- Um método estático não estará associado a um instância específica — somente à classe — portanto não poderá acessar valores de nenhuma variável de instância de sua classe. Ele não saberia os valores de *que* instância usar.

- Um método estático não pode acessar um método não-estático, já que geralmente os métodos não-estáticos estão associados ao estado da variável de instância.

- Se você tiver uma classe que sé tenha métodos estáticos e não quiser que ela seja instanciada, poderá marcar o construtor

Figura 5.22: Primeiro tercil da Figura 5.8 com o uso do algoritmo

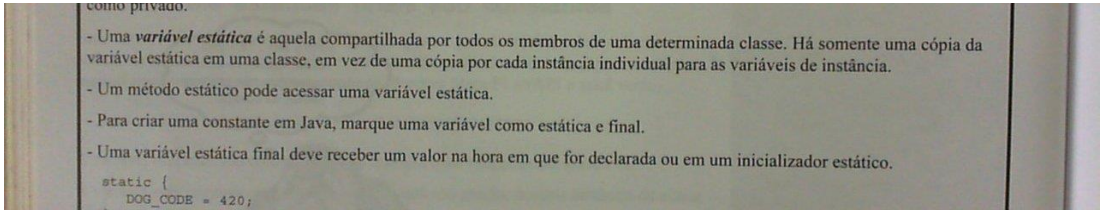

- Um método estático pode ser chamado sem que haja qualquer instância de sua classe no heap.

- Um método ser estático pode ser adequado para um método utilitário que não dependa (e que nunca dependerá) do valor de uma variável de instância específica.

- Um método estático não estará associado a um instância específica — somente à classe — portanto não poderá acessar valores de nenhuma variável de instância de sua classe. Ele não saberia os valores de *que* instância usar.

- Um método estático não pode acessar um método não-estático, já que geralmente os métodos não-estáticos estão associados ao estado da variável de instância.

Figura 5.23: Primeiro tercil da Figura 5.8 sem o uso do algoritmo


como privado.

- Uma ***variável estática*** é aquela compartilhada por todos os membros de uma determinada classe. Há somente uma cópia da variável estática em uma classe, em vez de uma cópia por cada instância individual para as variáveis de instância.

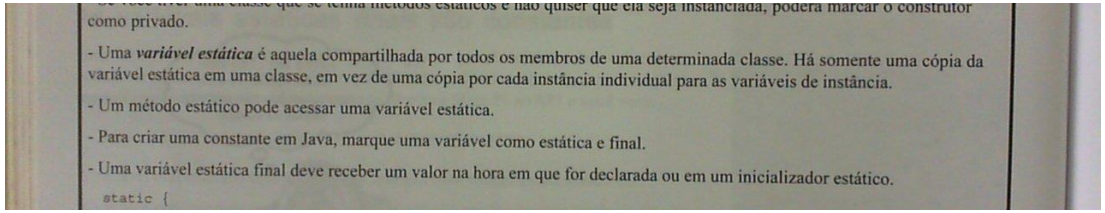
- Um método estático pode acessar uma variável estática.

- Para criar uma constante em Java, marque uma variável como estática e final.

- Uma variável estática final deve receber um valor na hora em que for declarada ou em um inicializador estático.

```
static {
   DOG_CODE = 420;
```

Figura 5.24: Segundo tercil da Figura 5.8 com o uso do algoritmo


como privado.

- Uma ***variável estática*** é aquela compartilhada por todos os membros de uma determinada classe. Há somente uma cópia da variável estática em uma classe, em vez de uma cópia por cada instância individual para as variáveis de instância.

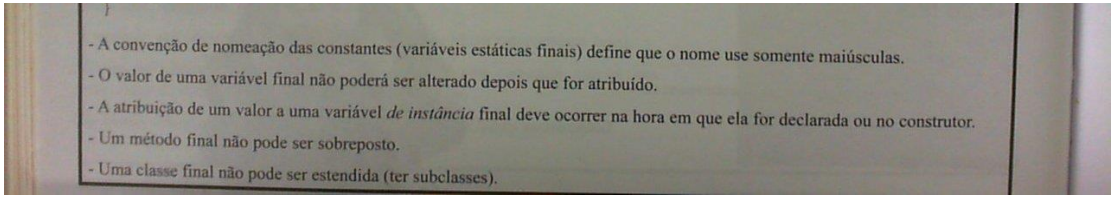
- Um método estático pode acessar uma variável estática.

- Para criar uma constante em Java, marque uma variável como estática e final.

- Uma variável estática final deve receber um valor na hora em que for declarada ou em um inicializador estático.

```
static {
```

Figura 5.25: Segundo tercil da Figura 5.8 sem o uso do algoritmo


}

- A convenção de nomeação das constantes (variáveis estáticas finais) define que o nome use somente maiúsculas.

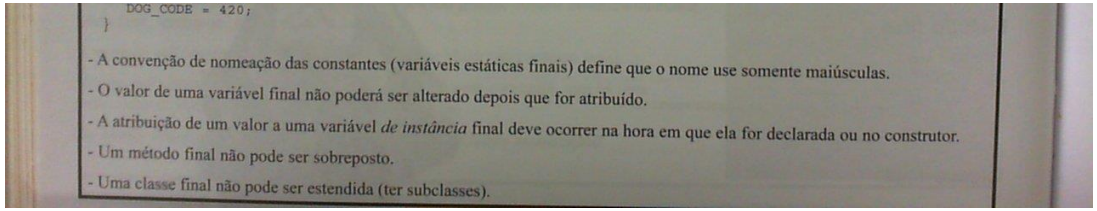
- O valor de uma variável final não poderá ser alterado depois que for atribuído.

- A atribuição de um valor a uma variável *de instância* final deve ocorrer na hora em que ela for declarada ou no construtor.

- Um método final não pode ser sobreposto.

- Uma classe final não pode ser estendida (ter subclasses).

Figura 5.26: Terceiro tercil da Figura 5.8 com o uso do algoritmo


```
   DOG_CODE = 420;
}
```

- A convenção de nomeação das constantes (variáveis estáticas finais) define que o nome use somente maiúsculas.

- O valor de uma variável final não poderá ser alterado depois que for atribuído.

- A atribuição de um valor a uma variável *de instância* final deve ocorrer na hora em que ela for declarada ou no construtor.

- Um método final não pode ser sobreposto.

- Uma classe final não pode ser estendida (ter subclasses).

Figura 5.27: Terceiro tercil da Figura 5.8 sem o uso do algoritmo


Aumento de leitos sem planejamento em cidades como M

Expectativa é que redes substituam estabelecimentos

Para empresários e associações, na concorrência que deve se instalar após o torneio, os mais prejudicados

12 mil, em 2009, para 17 mil durante a Copa —alta de 41%.

"Cuiabá tem a situação mais alarmante. Não tem uma

Figura 5.28: Primeiro tercil da Figura 5.9 com o uso do algoritmo


Aumento de leitos sem planejamento em cidades como M

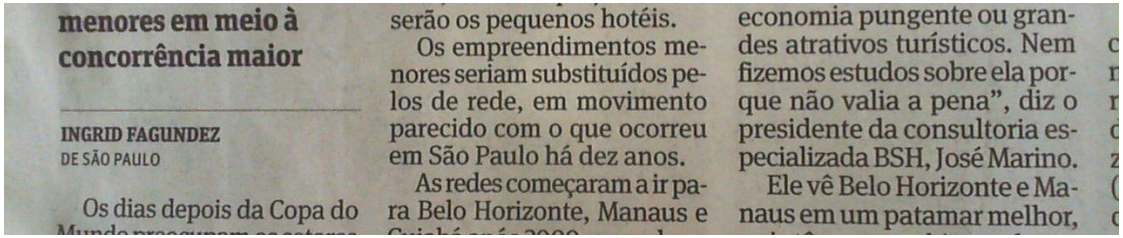
Expectativa é que redes substituam estabelecimentos

Para empresários e associações, na concorrência que deve se instalar após o torneio, os mais prejudicados

12 mil, em 2009, para 17 mil durante a Copa —alta de 41%.

"Cuiabá tem a situação mais alarmante. Não tem uma

Figura 5.29: Primeiro tercil da Figura 5.9 sem o uso do algoritmo


menores em meio à concorrência maior

INGRID FAGUNDEZ
DE SÃO PAULO

Os dias depois da Copa do

serão os pequenos hotéis.

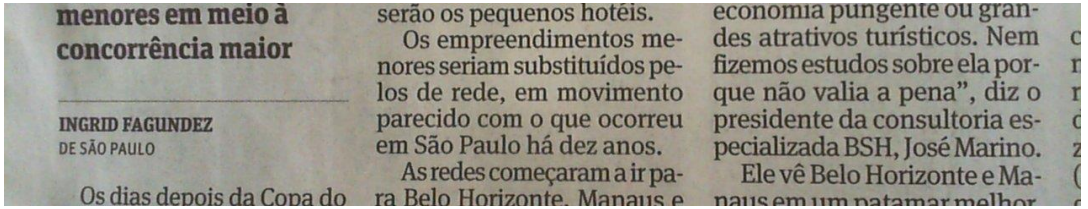
Os empreendimentos menores seriam substituídos pelos de rede, em movimento parecido com o que ocorreu em São Paulo há dez anos.

As redes começaram a ir para Belo Horizonte, Manaus e

economia pungente ou grandes atrativos turísticos. Nem fizemos estudos sobre ela porque não valia a pena", diz o presidente da consultoria especializada BSH, José Marino.

Ele vê Belo Horizonte e Manaus em um patamar melhor,

Figura 5.30: Segundo tercil da Figura 5.9 com o uso do algoritmo


menores em meio à concorrência maior

INGRID FAGUNDEZ
DE SÃO PAULO

Os dias depois da Copa do

serão os pequenos hotéis.

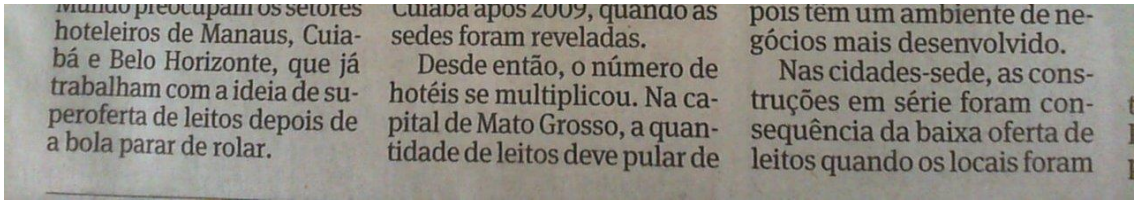
Os empreendimentos menores seriam substituídos pelos de rede, em movimento parecido com o que ocorreu em São Paulo há dez anos.

As redes começaram a ir para Belo Horizonte, Manaus e

economia pungente ou grandes atrativos turísticos. Nem fizemos estudos sobre ela porque não valia a pena", diz o presidente da consultoria especializada BSH, José Marino.

Ele vê Belo Horizonte e Manaus em um patamar melhor,

Figura 5.31: Segundo tercil da Figura 5.9 sem o uso do algoritmo


Mundo preocupam os setores hoteleiros de Manaus, Cuiabá e Belo Horizonte, que já trabalham com a ideia de superoferta de leitos depois de a bola parar de rolar.

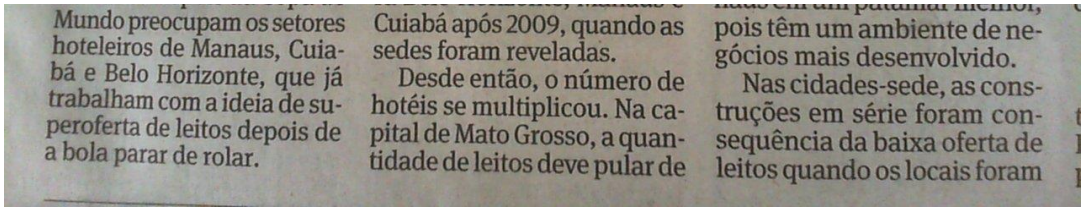
Cuiabá após 2009, quando as sedes foram reveladas.

Desde então, o número de hotéis se multiplicou. Na capital de Mato Grosso, a quantidade de leitos deve pular de

pois têm um ambiente de negócios mais desenvolvido.

Nas cidades-sede, as construções em série foram consequência da baixa oferta de leitos quando os locais foram

Figura 5.32: Terceiro tercil da Figura 5.9 com o uso do algoritmo


Mundo preocupam os setores hoteleiros de Manaus, Cuiabá e Belo Horizonte, que já trabalham com a ideia de superoferta de leitos depois de a bola parar de rolar.

Cuiabá após 2009, quando as sedes foram reveladas.

Desde então, o número de hotéis se multiplicou. Na capital de Mato Grosso, a quantidade de leitos deve pular de

pois têm um ambiente de negócios mais desenvolvido.

Nas cidades-sede, as construções em série foram consequência da baixa oferta de leitos quando os locais foram

Figura 5.33: Terceiro tercil da Figura 5.9 sem o uso do algoritmo

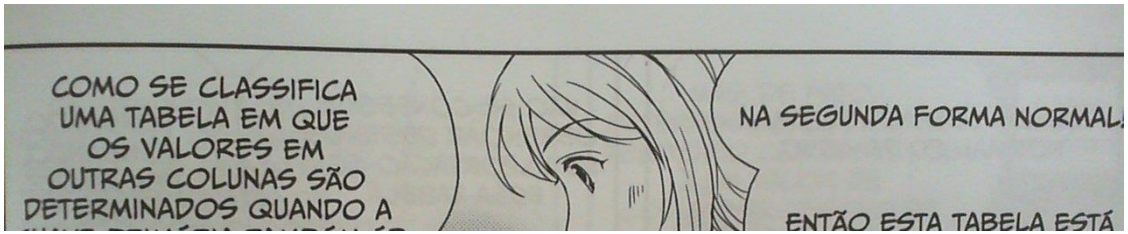


Figura 5.34: Primeiro tercil da Figura 5.10 com o uso do algoritmo

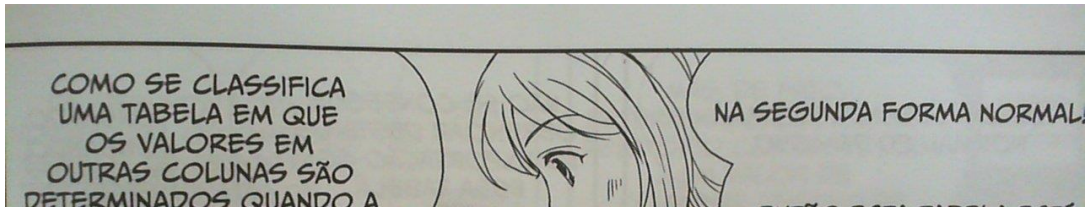


Figura 5.35: Primeiro tercil da Figura 5.10 sem o uso do algoritmo

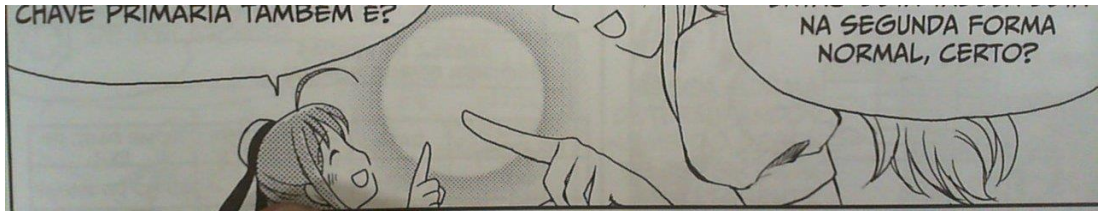


Figura 5.36: Segundo tercil da Figura 5.10 com o uso do algoritmo

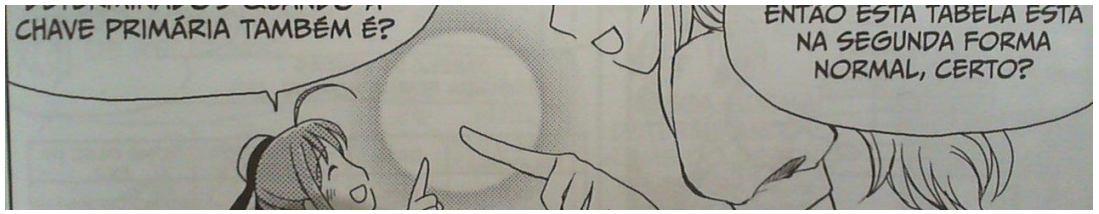


Figura 5.37: Segundo tercil da Figura 5.10 sem o uso do algoritmo

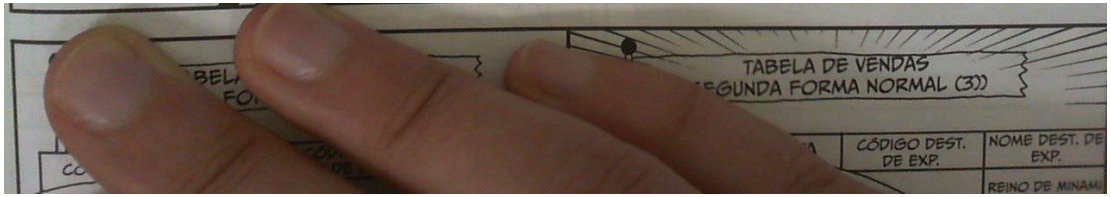


Figura 5.38: Terceiro tercil da Figura 5.10 com o uso do algoritmo

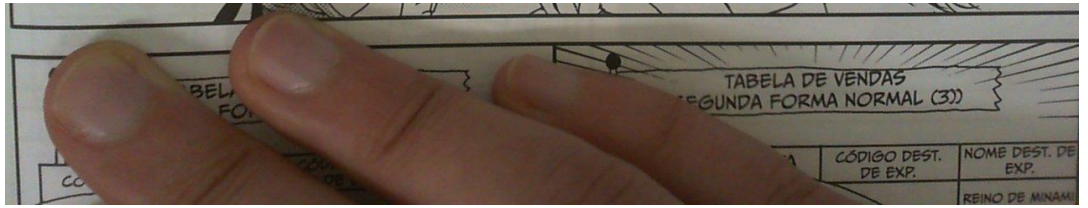


Figura 5.39: Terceiro tercil da Figura 5.10 sem o uso do algoritmo

Com os tempos de processamento apresentados nas tabelas 5.8 à 5.11, nota-se que o tempo para processar cada recorte é de aproximadamente um terço do tempo que leva para processar a imagem inteira, dependendo do tamanho de cada recorte e do tanto de texto presente em cada recorte (por exemplo, um recorte menor que contem mais texto, leva mais tempo para processar do que um recorte maior que conteve menos texto), de modo que o tempo gasto para processar os três recortes é sempre próximo do tempo levado para processar a imagem inteira. Vale lembrar que a variação de tempo gasto para processar as imagens também é influenciado pelo fato de o Linux ser um sistema de tempo compartilhado.

Capítulo 6

Considerações Finais

6.1 Conclusões

Sistemas de OCR são utilizados de diversas formas para auxiliar a vida das pessoas, em diferentes tipos de ferramentas.

Neste trabalho foi feito um estudo dos métodos computacionais envolvidos no processo de OCR, mostrando as principais etapas necessárias desde a captação da imagem até o processamento da informação e a exibição.

Adicionalmente, foi feito um levantamento das ferramentas OCR de código aberto disponíveis e uma avaliação delas para determinar qual seria a ideal para ser integrada ao xLupa embarcado junto com um sintetizador de voz para leitura do texto gerado pelo OCR.

Para realizar a integração no xLupa embarcado, uma modificação no código foi feita para que, ao pressionar de um botão, um arquivo da imagem capturada pelo ampliador seja salva e o processo de OCR seja realizado. O arquivo texto resultando é então enviado para o sintetizador de voz para a leitura do documento. Posteriormente o código modificado para dividir a imagem capturada em três partes, de modo a diminuir o tempo de espera do resultado.

A maior dificuldade na hora de examinar uma captura de tela a partir de uma *webcam* com o Tesseract está no fato de que a imagem precisa estar muito bem alinhada para que ele consiga apresentar resultados satisfatórios em termos de reconhecimento de caracteres, além do fato de que qualquer dobra em uma página e até mesmo sombras podem afetar o resultado.

O objetivo do trabalho foi alcançado, de verificar a viabilidade do uso de um OCR na placa BeagleBoard-xM, e a integração do mesmo com o xLupa embarcado para servir como tecnologia assistiva e auxiliar pessoas com baixa visão na leitura de documentos impressos. No entanto é importante frisar que os tempos de processamento são altos e proibitivos.

6.2 Trabalhos Futuros

Durante os testes realizados com as capturas de tela após a integração do OCR ao xLupa embarcado, foi constatado a dificuldade do mesmo em interpretar palavras em situações que ocorreram enviesamento ou distorções no papel causadas por dobras. Outra questão é que o algoritmo que determina a altura do recorte errou em algumas situações em que o tamanho das letras do texto analisado eram pequenas, devido a ruídos causados na imagem capturada por causa da iluminação e dobras no papel. Para melhorar o desempenho deste algoritmo na decisão da altura do recorte, seria possível aplicar uma binarização da imagem antes de analisá-la. Para tornar o método do recorte útil para textos dispostos em colunas, seria possível aplicar métodos de segmentação de imagem para destacar as colunas antes de recortar a imagem. Nas imagens com enviesamento, o texto gerado a partir dos recortes pode ser prejudicado devido ao fato de que o recorte é feito na vertical em linha reta. Para melhorar o acerto de palavras do OCR, pode ser feito um pré-processamento que corrija estas distorções através de métodos de transformação de imagem. Outra sugestão seria a de analisar o código do Tesseract com o objetivo de verificar se é possível otimizá-lo para funcionar melhor com o hardware da BeagleBoard-xM e também implementar a execução do Tesseract e do eSpeak de forma paralela.

Referências Bibliográficas

[1] "*BeagleBoard*". Disponível em: <http://beagleboard.org/>. Acessado em 20/07/13.

[2] HACHMANN, D.; SANTIAGO, P.; BIDARRA, J.; OYAMADA, M. Um ampliador de tela embarcado utilizando arquiteturas heterogêneas. In: 12º Fórum Internacional Software Livre fisl12/workshop de Software Livre, 2011, Porto Alegre. Workshop de Software Livre - WSL/FISL12. Pelotas - RGS: Editora e Gráfica Universitária - Univ. Federal de Pelotas, 2011. p. 1-6.

[3] "*Saber: Dados do IBGE sobre deficiência*". Disponível em: <http://louroassis.blogspot.com.br/2011/12/saber-dados-do-ibge-sobre-deficiencia.html>. Acessado em 30/10/2014.

[4] BUNKE, H.; WANG, P. *Handbook of Character Recognition and Document Image Analysis*. 2. ed. Singapura: Uto-Print, 2000. capítulo: Image Processing Methods for Document Image Analysis, p. 1-44.

[5] FILHO, O.; NETO, H. *Processamento Digital de Imagens*. 1. ed. Rio de Janeiro: Brasport, 1999. capítulo: Aquisição e digitalização de imagens, p. 19-21.

[6] "A Survey Paper on Character Recognition". Disponível em <http://www.academia.edu/6829079/A_Survey_Paper_on_Character_Recognition> . Acessado em 24/11/13.

[7] SCHANTZ, H. *The history of OCR, optical character recognition*. 1. ed. Estados Unidos: Recognition Technologies Users Association, 1982.

[8] MORI, S.; NISHIDA, H.; YAMADA, H. *Optical Character Recognition*. 1. ed. Estados Unidos: Wiley-Interscience, 1999.

[9] "*OCR Introduction*". Disponível em <www.dataid.com/aboutocr.htm>. Acessado em 20/07/13.

[10] "*How does OCR document scanning work?*". Disponível em < www.explainthatstuff.com/how-ocr-works.html >. Acessado em 20/07/13

[11] "*The basic patter recognition and classification with openCV | Damiles*". Disponível em <http://blog.damiles.com/2008/11/the-basic-patter-recognition-and-classification-with-opencv/ >. Acessado em 20/07/13.

[12] "*Tesseract-ocr*". Disponível em: <https://code.google.com/p/tesseract-ocr/>. Acessado em 04/04/13.

[13] "*OpenOCR*". Disponível em: <http://en.openocr.org/download/>. Acessado em 04/04/13.

[14] "*An Overview of the Tesseract OCR Engine*". Disponível em <http://tesseract-ocr.googlecode.com/svn/trunk/doc/tesseracticdar2007.pdf>. Acessado em 20/07/13.

[15] "*Optical Character Recognition (OCR) – How it works*". Disponível em <http://www.nicomsoft.com/optical-character-recognition-ocr-how-it-works/>. Acessado em 20/07/13.

[16] "*GOCR*". Disponível em: <http://jocr.sourceforge.net/>. Acessado em 04/04/13.

[17] "*Ocrad*". Disponível em: <http://www.gnu.org/software/ocrad/>. Acessado em 04/04/13.

[18] *"$150 board sports Cortex-A8*". Disponível em: <http://archive.is/20120711061852/http://linuxdevices.com/news/NS5852740920.html>. Acessado em 20/07/13.

[19] "*BeagleBoard | Arch Linux ARM*". Disponível em <http://archlinuxarm.org/platforms/armv7/ti/beagleboard>. Acessado em 29/09/13.

[20] "Processors - ARM". Disponível em: <http://www.arm.com/products/processors/>. Acessado em 05/11/2014.

[21] "*ARM Cores Climb Into 3G Territory*". Disponível em: <http://www.extremetech.com/extreme/52180-arm-cores-climb-into-3g-territory>. Acessado em 29/09/13.

[22] "*The Two Percent Solution*". Disponível em: <http://www.embedded.com/electronics-blogs/significant-bits/4024488/The-Two-Percent-Solution>. Acessado em 29/09/13.

[23] "*ARM Holdings eager for PC and server expansion*" Disponível em: <http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers/>. Acessado em 29/09/13.

[24] "*ARM from zero to billions in 25 short years*". Disponível em: <http://blogs.arm.com/smart-connected-devices/204-arm-from-zero-to-billions-in-25-short-years/>. Acessado em: 29/09/13.

[25] STALLINGS, W. *Fundamentals Of Computer Organization and Architecture.*7. ed. Estados Unidos: John Wiley & Sons, 2005.

[26] "*ARM Processor Architecture*". Disponível em: <http://www.arm.com/products/processors/instruction-set-architectures/index.php>. Acessado em 29/09/13.

[27] "*Cortex-A8 Processor - ARM*". Disponível em <http://www.arm.com/products/processors/cortex-a/cortex-a8.php>. Acessado em 29/09/13.

[28] WACHENFELD, S.; KLEIN, H.; JIANG, X. Annotated Databases for the Recognition of Screen-Rendered Text. In: Ninth International Conference on Document Analysis and Recognition (ICDAR2007), 2007. Alemanha: IEEE Computer Society Press, 2007, p. 1-5

[29] "*eSpeak: Speech Synthesizer*". Disponível em <http://espeak.sourceforge.net>. Acessado em 13/10/14.

[30] HACHMANN, D. R. *Distribuição de tarefas em MPSoC Heterogêneo: estudo de caso no OMAP3530*. Trabalho de Conclusão de Curso: Ciência da Computação, UNIOESTE, Cascavel, 2011. Monografia.