Corrigindo um Modelo de Gabarito com C++ e OpenCV

Gabriel Antonio Pereira dos Santos Carneiro, Rejanio das Neves de Moraes Filho

Departamento de Ciências Exatas (DEXA) – Universidade Estadual de Feira de Santana (UEFS), Bahia – Brasil

gabri14el@gmail.com, rejaniomoraes@gmail.com

Abstract. *This meta-paper describes the steps of the development of a software that the objective is correct a template. The respective software meets the yours objective satisfactorily. Tests of validation and reliability were not made, but in simple tests it had an excellent performance.*

Resumo. *Este meta-artigo descreve os passos para o desenvolvimento de um programa que corrige um gabarito de vestibular. O programa cumpre o seu objetivo de maneira satisfatória. Vale ressaltar que não foram feitos testes fortes (matematicamente) de validação e confiabilidade, porém em testes básicos o software teve um excelente desempenho.*

1. Introdução

Hoje em dia a computação está presente em todos os segmentos da civilização: educação, economia, saúde, diversão, etc. Vale apena observar que a sua utilização na educação está cada vez maior, seja como objeto de aprendizagem, como facilitador de gerenciamento ou como virtualização da sala de aula.

Visando a necessidade por facilitadores baratos nas correções de provas, nós alunos da matéria de Computação Visual do curso de Engenharia de Computação da Universidade Estadual de Feira de Santana, decidimos fazer um software que corrija de maneira rápida e prática um determinado modelo de gabarito.

O software construído cumpre com o seu objetivo e corrige de maneira satisfatória, entregando ao usuário as respostas contida na folha de gabarito, seja ela uma letra, questão em branco, ou anulação por mais de uma letra marcada.

Este trabalho relata todo o desenvolvimento da solução (desde as bases de conhecimento necessárias até os testes executados no software) e está divido em quatro seções: Fundamentação Teórica, Metodologia, Resultados e Discussões e Conclusões. Fundamentação Teórica traz os conhecimentos básicos necessários para a construção; Metodologia descreve como chegamos à solução; Resultados e discussões descreve como funciona a solução e os resultados que obtivemos com ela; Conclusões traz as nossas conclusões sobre o software.

2. Fundamentação Teórica

Cada conjunto de sistema de computação visual tem suas características, necessidades e particularidades; isso partindo do objeto a ser analisado e indo ao ambiente que o mesmo se encontra.

A biblioteca OpenCV adjunto aos conceitos de segmentação de imagens e morfologia matemática foram de grande valia durante todo o desenvolvimento do nosso programa e de suas minuciosidades. Nesta seção descreveremos rapidamente cada um desses itens.

2.1. Bibilioteca OpenCV

Um recurso sem cobranças para o meio acadêmico, a OpenCV , é um conjunto bibliotecas disponíveis para algumas linguagens de programação dentre elas C++, Python e Java. Desenvolvida em 2000, oferece uma grande quantidade de funções para auxiliar no processamento de imagem, visão computacional, reconhecimento de padrões, entre outras aplicações.

A OpenCV é estruturada em módulos que contemplam o estudo da aprendizagem de máquina, estrutura de dados, entre outros. No desenvolvimento do nosso produto usamos apenas o módulo básico com algoritmos de visão computacional.

2.2. Segmentação de Imagens

A segmentação de uma imagem é um conjunto de técnicas usadas no pré-processamento de uma imagem que tem por finalidade "subdividir uma imagem em regiões ou objetos que a compõe" [Gonzalez e Woods, 2010] .

Gonzalez e Woods (2010) traz que "o nível de detalhe em que a subdivisão é realizada depende a ser resolvido. Ou seja, a segmentação deve parar quando os objetos ou regiões de interesse de uma aplicação forem detectados".

2.3. Limiarização

Um tipo de técnica que existe classificada na segmentação de imagem, a limiarização baseia-se em dividir um objeto em grupo de cinza com intensidade semelhante, gerando dessa forma o fundo e o objeto.

Esse é um processo simples quando na imagem tem níveis de cinza distintos. Em regiões que intensidade é uniforme, resulta em picos no histograma. Esse processo sofre com falhas, devido luminosidade em excesso que pode atrapalhar na classificação.

De uma forma básica a limiarização é uma função matemática que define a classificação dos pixels usando o fator *limiar* como parâmetro. Porém para a escolha do número para ser o fator limiar, é em alguns casos, uma tarefa não trivial, pois dependendo da do número escolhido, esse poderá não ajudar a dividir as regiões do objeto estudado.

2.4. Morfologia Matmática

Gonzalez e Woods (2010) define morfologia matemática como uma técnica para analisar ou descrever uma imagem que pode ser utilizada para várias funcionalidades.

Gonzalez e Woods (2010) traz também que "a linguagem da morfologia matemática é a teoria dos conjuntos. Como tal, oferece uma abordagem unificada e poderosa para vários problemas de processamento de imagens. Os conjuntos em morfologia matemática representam os objetos encontrados em uma imagem".

Dentro do contexto de operações básicas da morfologia, existem duas: erosão e dilatação. A primeira operação, erosão, os pixels que não atendem a um dado padrão são apagados da imagem. Na dilatação, uma pequena área relacionada a um pixel é alterada para um dado padrão. Entretanto, a depender tipo de imagem processada a definição de operação pode mudar [Motoso, 2012].

3. Metodologia

Todo conhecimento necessário para iniciar o desenvolvimento do produto foi obtido em sala de aula, durante o curso do componente curricular Computação Visual. O primeiro passo para o desenvolvimento foi uma rápida busca por trabalhos semelhantes/correlatos. Os resultados obtidos acabaram por não serem tão relevantes durante a criação do produto.

O passo seguinte seria a escolha de um modelo de gabarito que o nosso produto visava corrigir. Primeiro escolhemos um modelo, disponível gratuitamente na internet, porém no decorrer da escrita do código fonte encontramos problemas com o reconhecimento dos marcadores de páginas, optando então por um modelo básico criado num editor de texto (vide Apêndice 1).

Com o conhecimento adquirido em sala sobre processamento de imagens e a biblioteca *OpenCV* (C++), os passos básicos do desenvolvimento se iniciaram. Os dois maiores objetivos, computacionalmente falando, para o nosso produto foram o de consertar a perspectiva da imagem do gabarito que será corrigido e o de corrigir esse mesmo gabarito.

O primeiro passo para alcançar os objetivos anteriormente referidos é fazer um pré-processamento na imagem do gabarito que vamos corrigir. As etapas de pré-processamento utilizados foram: redimensionamento, conversão para escala de cinza, borramento, limiarização e aplicação de morfologia matemática.

Redimensionamos a imagem por dois motivos: menos esforço computacional para o processamento e maior domínio sobre regiões, ou seja, domínio sobre quais pixels formam uma determinada região (utilizada na correção do gabarito). A conversão para escala de cinza se fez necessária por também poupar esforço computacional e pelo fato da limiarização exigir uma imagem de canal único.

Como o dispositivo de captura de onde obteremos imagens é um digitalizador/*scanner*, geralmente as imagens resultantes vêm com pequenos ruídos, por isso fizemos um borramento utilizando o método por mediana. Escolhemos esse método por conta de sua eficácia na remoção de ruídos sal e pimenta.

Já na limiarização, utilizamos o método de *threshold* adaptativo, por conta de não definir um limiar fixo e assim ser mais confiável. Após observamos algumas falhas nas regiões formadas a partir da limiarização, utilizamos a morfologia matemática fazendo um fechamento em toda a imagem.

Vale ressaltar que tivemos um problema ao equalizar o histograma junto à utilização desse método de limiar, ocasionando um atraso no decorrer do desenvolvimento, pois a equalização causa picos no contraste, gerando problemas no método de limiarização.

Iniciamos então a primeira fase do processamento onde: buscamos os contornos, os momentos e seus centros de massa, e assim obtemos as regiões que nos interessavam e seus respectivos centros de massa (correspondentes aos *page markers*) com o objetivo de corrigir a perspectiva da imagem no momento tratada.

A seleção das regiões que nós buscamos nessa primeira fase é feita por dois fatores: área da região e localização da região.

Para utilizarmos a área como fator de descarte, observamos treze tamanhos diferentes dos marcadores de página, calculamos o seu desvio padrão que teve um resultado aproximadamente de 23, diminuímos isso da menor região computada e somamos esse mesmo valor à maior região computada, gerando então intervalo de [238, 355], ou seja as áreas de seus momentos são maiores ou iguais a 238 e menores ou iguais a 355.

Depois de verificar se a área tem um tamanho necessário para ser uma candidata à *page marker*, verificamos a sua localização na imagem. Para isso dividimos as colunas da imagem em cinco regiões e suas as linhas em sete, resultando assim trinta e cinco regiões, quatro dessas regiões estão nos extremos da imagem, caso o centro de massa da região esteja dentro de uma dessas quatro e seu centro de massa seja o mais extremo dentro dela, ele é selecionado como marcador de página.

Com os centros de massa podemos então corrigir a perspectiva da imagem, e assim teremos uma nova imagem corrigida e com ela saberemos exatamente na imagem onde está a próxima informação que procuramos: as alternativas do gabarito.

A essa nova imagem aplicamos também um borramento e a conversão para escala cinza, pois ela é uma cópia da foto original, sem qualquer pré-processamento. A mudança ocorre na maneira como a imagem é binarizada, aqui nós utilizamos um threshold simples, apenas para conversão da escala cinza em imagem binária.

Assim, podemos ir em cada retângulo do gabarito, pois conhecemos as coordenadas deles, e verificar se o mesmo foi marcado. Nessa etapa o software compara a quantidade de pixels correspondente ao preenchimento, caso eles sejam mais que os não correspondentes, então a letra é dada como marcada.

4. Resultados e Discussões

O software resultante abre uma imagem de nome “1.jpg” no mesmo diretório onde está localizado, processa e imprime, num terminal de linha de comando (vide Figura 2), as coordenadas dos centros de massa das regiões correspondentes aos *page markers*, a representação binária do gabarito, onde 0 representa não marcado e 1 representa marcado, e por último a reposta para cada alternativa do gabarito, inclusive se o usuário não marcou nenhuma opção, ou marcou 2 ou mais e a questão foi anulada. Vale ressaltar que ele corrige apenas o modelo mostrado no Apêndice 1, com somente 10 questões.

Para o funcionamento adequado do software a imagem já deve está posicionada, de maneira que não precise de rotação, pois ele não gira a imagem. As imagens devem ser adquiridas através de *scanner* e não devem estar muito tortas na hora da digitalização. O gabarito deve ser totalmente preenchido com caneta de cor preta. Um

mau preenchimento pode fazer com que ele ignore a marcação e caneta de outra cor pode atrapalhar o seu funcionamento de maneira que ele não consiga efetuar a correção.

A primeira fase do processamento (consertar a perspectiva), pode ser visualmente observado na Figura 1. Em Figura 1 (b) observamos o resultado do threshold adaptativo, enquanto em (c) já vemos apenas as áreas que procuramos (correspondente aos marcadores de página) e seus respectivos centros de massa (ponto impresso no meio da região), enquanto em (d) estamos diante da figura original (que pode ser vista em (a)) já com a perspectiva consertada.

A segunda fase do processamento consiste na binarização da imagem resultado da primeira fase e a computação e impressão dos itens marcados para o usuário. A saída do programa pode ser observada na Figura 2.

Para testar a eficácia do software, nós o colocamos a prova em sete cenários diferentes: três simples (mistura de: questões bem marcadas, questões em branco, questões mal marcadas e questões com mais de uma marcação), dois extremos (todas as letras de todas as questões marcadas e gabarito sem marcação alguma), um tortuoso (folha muito torta no digitalizador) e o último com marcação de caneta verde. Todas as imagens foram escaneadas numa multifuncional da marca HP, modelo Deskjet F2050. Todos os testes foram executados na distribuição Linux Ubuntu 16.04.

Os resultados dos testes foram satisfatórios. Os casos simples e extremos obtiveram os resultados esperados: o software conseguiu corrigir o gabarito de maneira correta, sem qualquer imprevisto ou falha. No teste tortuoso, o software não conseguiu encontrar um dos marcadores de página, pois ele não se encontrava em uma das zonas que nós taxamos de extremas. No teste de marcação com a caneta verde o software só conseguiu localizar dois marcadores de páginas.

Vale ressaltar que nenhuma biblioteca restrita a sistema operacional foi utilizada no desenvolvimento, logo o código do programa pode ser recompilado para qualquer sistema operacional com suporte à biblioteca OpenCV e um compilador C++.

Não fizemos testes de validade e confiabilidade profundos. Logo, numa futura atualização do software deve-se aplicar o ground-trouth no nosso conjunto de testes para termos números concretos sobre os acertos do programa.

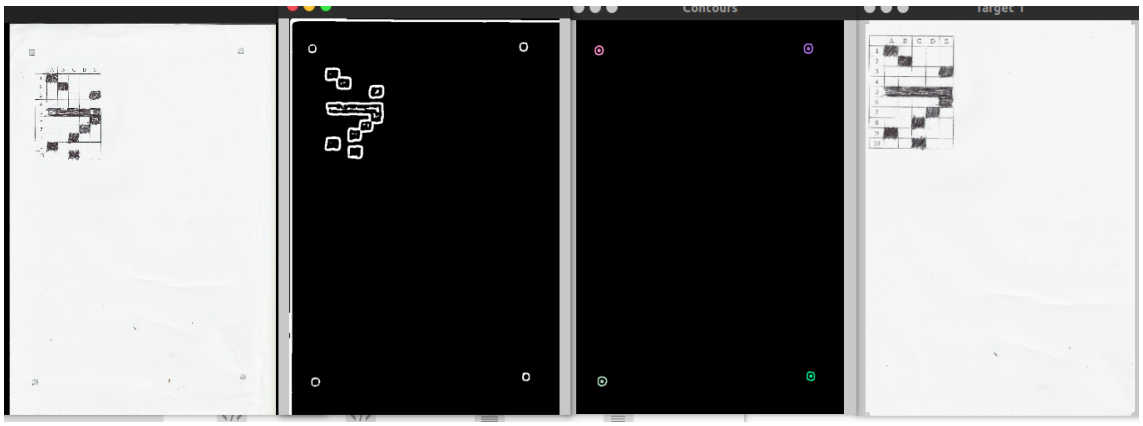

Figura 1. a) imagem original; b) threshold adaptativo; c) contornos dos page markers; d) foto com perspectiva consertada

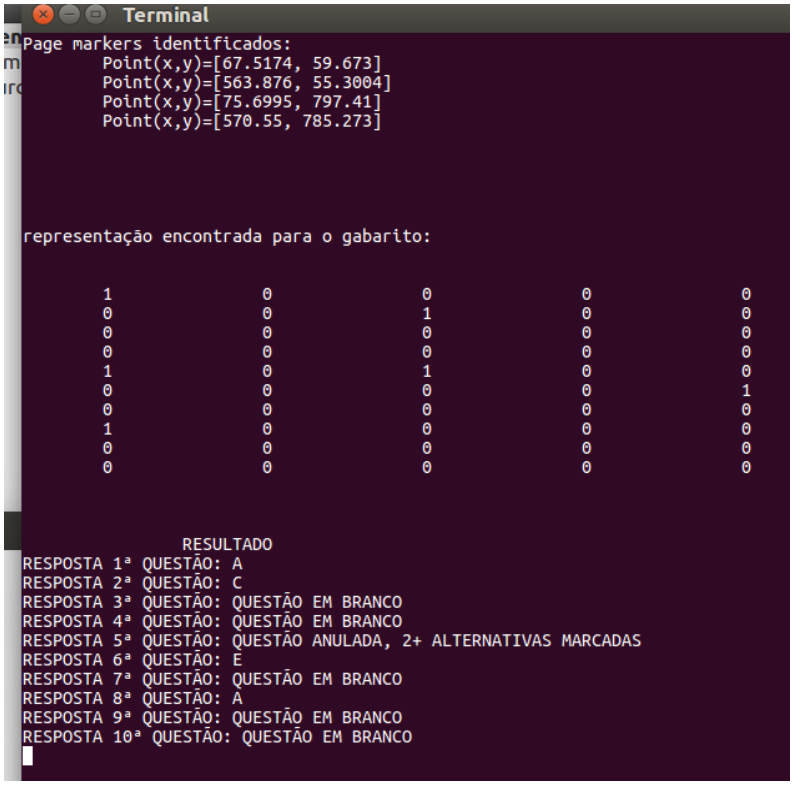

Figura 2. Exemplo da saída de uma execução do programa

Com relação a utilização, vários aspectos podem ser melhorados. O software pode passar a abrir um conjunto de imagens e não só uma, pode ler um gabarito com a marcação correta e peso de cada questão e já trazer a nota pronta, entre outras melhorias.

5. Conclusões

O software que teve o desenvolvimento aqui relatado cumpre seu objetivo de corrigir um gabarito, processando imagens e utilizando a biblioteca OpenCV para esse fim. Nos testes realizados ele consegiu um excelente desempenho, funcionando corretamente em todas as situações tidas como normais (preenchimento de todo o quadro, caneta preta, imagem de entrada não muito torta, modelo do apêndice). Nos testes de casos anormais (imagem de entrada muito torta, preenchimento com caneta verde) ele não funcionou.

Para o usuário a utilização ainda não está tão simplificada, pois o mesmo tem que renomear todas as imagens de gabarito que deseja corrigir para ".1jpg" e só é possível corrigir uma imagem por vez.

Para os trabalhos futuros temos teste mais aprofundado de validação e confiabilidade, facilitação de uso do software (leitura de vários arquivos, leitura de um gabarito correto para cálculo de nota, etc), dinamismo no modelo que usamos para correção (usuário poder gerar modelo com quantas questões ele quiser e software reconhecer a quantidade automaticamente), girar imagem quando preciso, entre outros.

References

Gonzalez, R. and Woods, R. (2010) "Processamento digital de imagens ", Editora

Peason.,São Paulo.

Motoso, Adson (2012) "Segmentação adaptativa baseada em histograma de imagem sanguíenas" http://tcc.ecomp.poli.br/20122/Adson%20Matoso%20Segmentacao%20Adaptativa%20baseada%20em%20Histograma%20FINAL.pdf, Janeiro. Recife.

Apêndice 1 – Modelo que o Software Reconhece

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					