Automatic Correction of Multiple-Choice Tests on Android Devices

Francisco de Assis Zampirolli, Rodrigo Teiske China,
Rogério Perino de Oliveira Neves, José Artur Quilici-Gonzalez
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Santo André, Brasil
E-mails: {fzampirolli, rogerio.neves, jose.gonzalez}@ufabc.edu.br, rodrigo.china@aluno.ufabc.edu.br

***Abstract*— There are currently several commercial automatic graders for multiple-choice test sheets, usually composed by a software and scanner bundle. But the wide spread of laptops, tablets and smartphones with integrated cameras offer new possibilities to perform the same task without the need for any extra hardware or software and with the benefit of low cost mobile execution. This paper presents an application for Android smartphones and tablets that uses the OpenCV library. Furthermore, a new image processing technique is presented for finding outer borders of objects. This application showed excellent results in tests for the correction of hundreds of multiple-choice test sheets in real conditions.**

Keywords—Image Processing; Image Segmentation; Mathematical Morphology; Multiple-Choice Test; Optimal Mark Recognition.

I. INTRODUÇÃO

Com a popularização de *smartphones*, *tablets* e *laptops* equipados com câmeras de alta resolução, várias tarefas antes realizadas por grandes máquinas se tornaram possíveis tendo à mão apenas um destes pequenos dispositivos [1]. Uma alternativa interessante é a possibilidade de corrigir testes de múltipla-escolha usando apenas um *smartphone* ou *tablet*, definido aqui por dispositivo móvel.

Embora a câmera desses equipamentos tenha qualidade suficiente para fotografar satisfatoriamente o gabarito do teste a ser avaliado, a variação da distância, do ângulo, da luminosidade na foto, entre outros fatores, dificultam o processamento das imagens, exigindo técnicas refinadas de segmentação de imagens.

Existem vários *softwares* populares para gerar testes, como o "testesdemultiplaescolha" [2] ou o "Quiz-Press", também há várias soluções recentes, como o "Eyegrade" [3], e há dispositivos e *softwares* comerciais específicos para corrigir testes, geralmente com custo elevado. Além disso, não há detalhes na literatura de metodologias e técnicas populares para implementar corretores de testes ou resolver problemas relacionados.

Através de uma tecnologia conhecida como *Optical Mark Recognition* (OMR) [4] é possível fazer a aquisição de dados discretos contidos em formulários predefinidos, detectando com a ajuda de um *scanner* a presença ou não de marcas nos espaços reservados para preenchimento. Esta tecnologia foi introduzida nos Estados Unidos em 1932 (IBM 805 *Test Scoring Machine*), sendo já naquela época utilizada para avaliar testes de múltipla-escolha. Para sua implementação são frequentemente utilizados *scanners* dedicados a OMR ou *scanners* de imagem convencionais. No caso de um *scanner* de imagem, um *software* OMR processa a imagem com eficácia similar a dos *scanners* dedicados a OMR [4], porém com desempenho baixo, mas com algumas vantagens tais como baixo custo e a possibilidade de usar formulários OMR específicos de acordo com a necessidade do usuário [5, 6, 7]. Há também vários programas de OMR e OCR (*Optical Character Recognition*) disponíveis, assim como aplicativos para *smartphones* com iOS e Android voltados para a decodificação de códigos de barras e códigos QR-code [5, 6, 7, 8, 9].

Neste artigo é descrita a implementação do MCTest, um corretor automático de testes de múltipla-escolha para dispositivos móveis com sistema operacional Android. O MCTest utiliza um *quadro de respostas* mais simples do que de formulários OMR convencionais, que pode ser construído para o número de questões e respostas desejadas em editores de texto comuns, além de oferecer vários recursos como a possiblidade de diferenciar gabaritos e testes de vários tipos diferentes e com pesos diferenciados para cada questão.

II. PROCEDIMENTOS

A. Descrição do MCTest

O MCTest é um aplicativo móvel (App) disponível gratuitamente na *Google Play Store* e pode ser instalado em dispositivos com o sistema operacional Android. Após instalado o App, o usuário pode criar Projetos para cada turma de alunos. As regras de construções nos quadros de respostas dos testes de múltipla-escolha devem seguir alguns critérios, detalhados em [10]. Em http://vision.ufabc.edu.br/MCTest há vários exemplos de formatos de testes para serem aplicados, assim como as regras de utilização deste App estão num tutorial. Neste artigo será detalhada a parte de aquisição e processamento das imagens.

As próximas figuras foram capturadas da tela de um *smartphone Samsung Galaxy Note 2*, com Android versão 4.3 e ilustram o processo de correções de testes de múltipla-escolha usando o MCTest.

Figura 1 – Quadro de respostas de um gabarito. Posicione o quadro de respostas do teste entre as linhas verdes, os 4 extremos deste quadro de respostas devem ficar na área branca, fora da elipse vermelha (importante: deixar fundo branco entre estas linhas, além do quadro de respostas) e fotografar (botão verde).

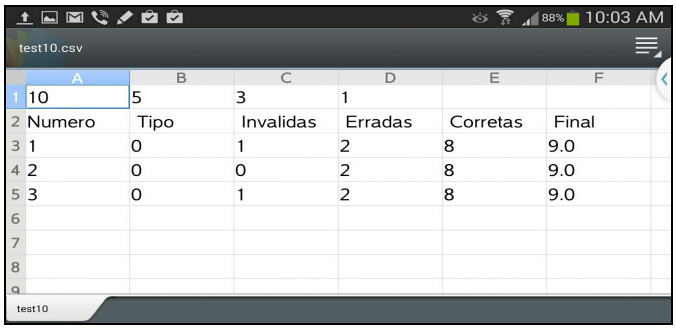
Figura 2 – Quadro de respostas de um aluno para correção.

(a) (b)

Figura 3 – (a) Após fotografar, se a figura contém todo o quadro de respostas, basta "Enviar para Correção". (b) Correção de um teste de um aluno.

A Figura 1 apresenta um quadro de respostas de um gabarito, primeira imagem a ser processada pelo App antes de corrigir os testes dos alunos.

Após o processamento de um ou mais gabaritos (para vários tipos diferentes de testes em um único Projeto), fotografar um teste para realizar a correção, como ilustra a Figura 2.



	A	B	C	D	E	F
1	10	5	3	1		
2	Numero	Tipo	Invalidas	Erradas	Corretas	Final
3	1	0	1	2	8	9.0
4	2	0	0	2	8	9.0
5	3	0	1	2	8	9.0
6						
7						
8						

Figura 4 – Conteúdo do arquivo CSV, referente a 3 testes corrigidos. Observa-se que as questões 5 e 7 do gabarito apresentado na Figura 1 valem 1,5, pois os quadrados da última linha destas questões estão marcados.

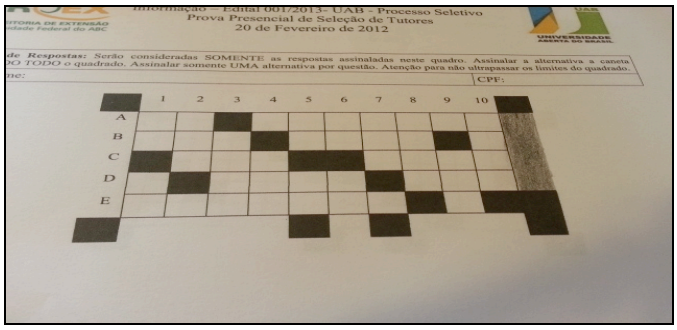


Figura 5 – Imagem `imgTemp` capturada pela câmera do dispositivo móvel.

A Figura 3-a mostra um teste já fotografado após várias transformações na imagem capturada na Figura 2. O passo seguinte é enviar para correção. Quando o MCTest inicia o processo de correção do teste, o gabarito é lido no *smartphone* e comparado com o teste em questão. Os resultados são mostrados na tela (Figura 3-b) para que o usuário possa verificar, aceitar ou repetir a correção, conforme desejar.

Imagens do quadro de respostas do aluno digitalizada e corrigida (nas Figuras 2 e 3-b) poderão ser salvas no *smartphone*. Também, um arquivo CSV contendo os resultados de todas as correções do Projeto é armazenado. A Figura 4 ilustra as informações do arquivo CSV que podem ser obtidas de cada teste, como questões inválidas, erradas, etc.

B. Processamento de Imagens

As imagens capturadas pelos dispositivos móveis são processadas usando a biblioteca de visão computacional OpenCV [11] importada pelo ambiente de desenvolvimento ADT (*Android Developer Tools*), versão 22.3 (disponível em http://developer.android.com/sdk), que inclui componentes essenciais para o desenvolvimento de aplicativos para Android, usando a linguagem de programação Java.

Após fotografada com a câmera do dispositivo móvel (Figuras 1 e 2), a imagem é salva na pasta do Projeto, (e.g. "…/MCTest/test10/imgTemp.jpg"). O exemplo que será utilizado para exemplificar as técnicas de processamento de imagens é apresentado na Figura 5. Esta imagem em particular representa um gabarito de uma prova do tipo 15, pois todos os "bits" da última coluna estão preenchidos. Este tipo 15 é interessante, pois ilustra bem o problema de detecção dos extremos do quadro de respostas, como veremos a seguir.

Em seguida, várias transformações são aplicadas na imagem armazenada na variável `imgTemp`, descritas a seguir. Optou-se por deixar trechos de código em Java, com as chamadas das transformações em imagens usando o OpenCV, pois são simples e próximos de algoritmos escritos em pseudocódigos, assim o leitor pode reproduzir facilmente os experimentos apresentados. Além disso, os códigos estão bem comentados, em verde, descrevendo os passos do código/algoritmo.

```
// ler arquivo do disco
Mat imgTemp =Highgui.imread(file.getAbsolutePath());
// recorta imagem, considera somente a área
// entre as linhas verdes
int H = imgTemp.height(), W = imgTemp.width();
int hh = (int) H / 4;
Rect roi = new Rect(0, hh, W, H - 2*hh + 20);
Mat imgRoi = new Mat(imgTemp, roi);
```

A imagem inicial é recortada, considerando apenas a área entre as linhas verdes (veja Figuras 1 e 2). O resultado é apresentado na Figura 6.

Em seguida, independente da resolução da câmera de vídeo do dispositivo móvel, é feito um redimensionamento com relação à largura fixa de imagem e também uma redução de cores para níveis de cinza, conforme ilustra o código e a imagem `imgGray` na Figura 7. Esta normalização é necessária para compatibilizar o MCTest com o maior número possível de dispositivos.

```
// Normalização da imagem com relação à largura
H = imgRoi.height();
W = imgRoi.width();
int rsize = (int) W / 500 + 1;
int rH = H % rsize;
H = H + rH;
rH = W % rsize;
W = W + rH;
Imgproc.resize(imgRoi, imgRoi, new Size(W/rsize,
H/rsize),0,0, Imgproc.INTER_CUBIC);
H = imgRoi.height();
W = imgRoi.width();
// converte imagem RGB para cinza
Mat imgGray = new Mat();
Imgproc.cvtColor(imgRoi, imgGray,
Imgproc.COLOR_BGR2GRAY);
```

Após a conversão para níveis de cinza, é feito um filtro para suavizar a imagem usando suavização Gaussiana conforme se vê na Figura 8. Outro filtro usado é o da mediana (veja Figura 9) [12].

```
// tamanho de Função Estruturante proporcional a H
int size_se = (int) H / 50;
if (size_se % 2 == 0)
  size_se++; // GaussianBlur requer size_se impar
Size se = new Size(size_se, size_se);
// filtro Gaussiano para suavizar imagem
Imgproc.GaussianBlur(imgGray, imgGray, se, 0);
// filtro da Mediana também para suavizar a imagem
Imgproc.medianBlur(imgGray, imgGray, 5);
```

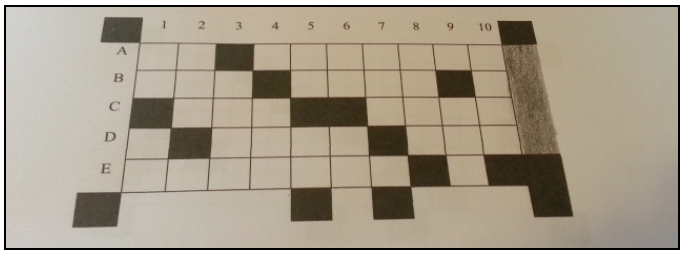


Figura 6 – Imagem `imgRoi` recortada, considerando apenas a área do quadro de respostas.

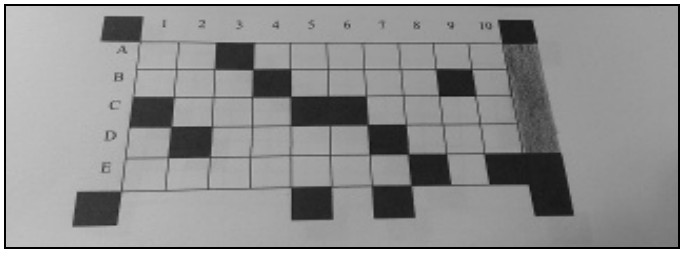


Figura 7 – Imagem em níveis de cinza `imgGray`.

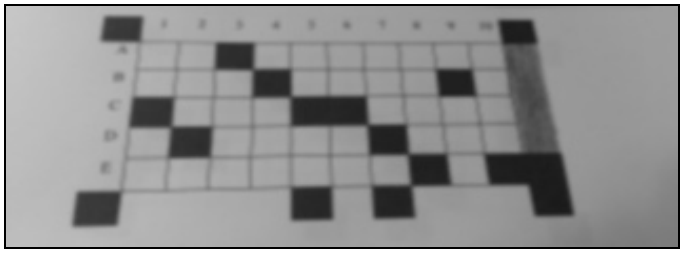


Figura 8 – Imagem após aplicar o filtro Gaussiano.

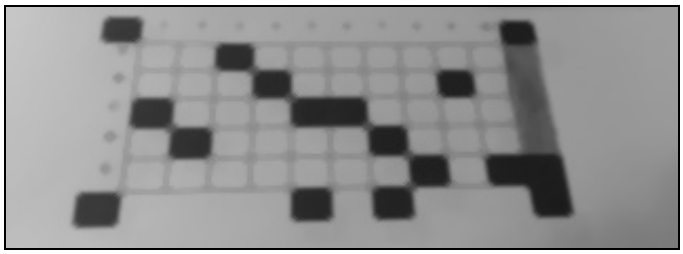


Figura 9 – Imagem após aplicar o filtro da Mediana.

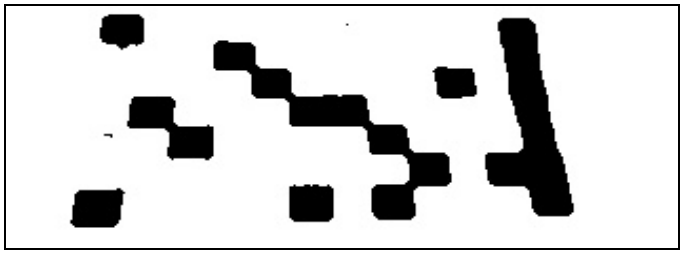


Figura 10 – Imagem após aplicar o *threshold adaptativo*.

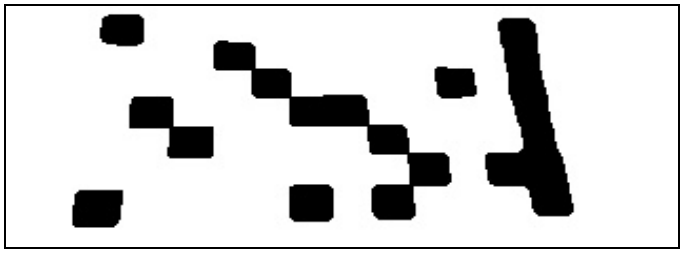


Figura 11 – Imagem `imgGray` após aplicar os filtros morfológicos.

Após estas transformações na imagem para suavizar os níveis de cinza, é realizado um *threshold adaptativo*, para tentar contornar as sombras que aparecem na imagem (Figura 10).

```
// binariza imagem usando um threshold adaptativo
Imgproc.adaptiveThreshold(imgGray, imgGray, 255,
Imgproc.ADAPTIVE_THRESH_MEAN_C,
Imgproc.THRESH_BINARY,61,15);
```

Agora, são aplicados mais dois filtros morfológicos, o primeiro chamado abertura morfológica, que é a erosão seguida da dilatação, com o objetivo de eliminar pequenos ruídos pretos. A outro filtro é a fechamento morfológico, que é a dilatação seguida da erosão, com o objetivo de preencher pequenos buracos brancos. Veja na Figura 11 o resultado destes dois filtros.

```
// define elemento estruturante,
int se_size = rsize;
if (se_size <= 0) se_size = 1;
Mat se = Imgproc.getStructuringElement(
Imgproc.MORPH_RECT,
new Size(2*se_size+1,2*se_size+1),
new Point(se_size, se_size));

// abertura para eliminar objetos pequenos
Imgproc.morphologyEx(imgGray, imgGray,
Imgproc.MORPH_OPEN, se);

// fechamento para preencher buracos pequenos
Imgproc.morphologyEx(imgGray, imgGray,
Imgproc.MORPH_CLOSE, se);
```

Após este pré-processamento da imagem, é finalmente realizada a segmentação do quadro de respostas. Para isto, é preciso identificar os 4 extremos deste quadro.

O primeiro método experimentado, que não apresentou bom desempenho, foi através da identificação dos extremos de cada contorno de objetos, como apresentado na Figura 12.

O problema de detecção de bordas é clássico e atual na área de processamento de imagens [13, 14]. Porém, para este caso específico de achar os 4 extremos do quadro de respostas, se faz necessário usar a técnica apresentada a seguir.

O segundo método de identificação dos extremos do quadro de respostas é usando a transformada de distância, método sendo originalmente apresentado neste trabalho. Inicialmente é criada uma imagem `imgCenterZero` com zeros em uma elipse no centro, conforme mostra a Figura 13.

```
// cria uma imagem com zeros em uma elipse no centro
Mat imgCenterZero = new Mat(H, W, CvType.CV_8U, new
Scalar(0));
Core.add(imgCenterZero, new Scalar(1000),
imgCenterZero);

// o tamanho da elipse é proporcional ao
// tamanho da imagem
Core.ellipse(imgCenterZero, new Point(W/2, H/2),
new Size(W/3,H/4),0,0,360, new Scalar(0,0,0),-1);
```

Figura 12 – Imagem após aplicar os filtros morfológicos, destacando os contornos e os 4 extremos por círculos.

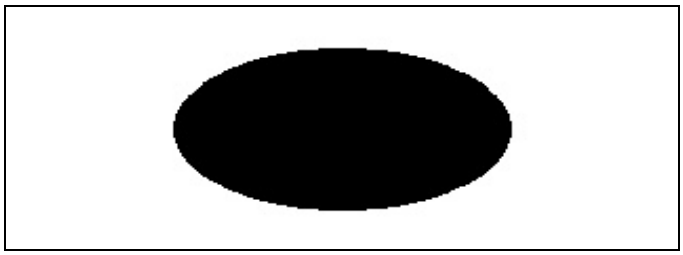


Figura 13 – Imagem `imgCenterZero` com centro em forma de elipse.

Figura 14 – Imagem destacando os 4 extremos do quadro de respostas.

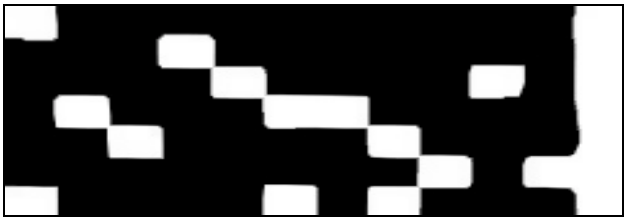


Figura 15 – Imagem `imgOutt` após aplicar a perspectiva usando os 4 extremos encontrados e visualizados na Figura 14.

Agora é calculada a transformada de distância `imgDist` sobre esta imagem da elipse.

```
// Transforma de distância
Mat imgDist = new Mat(H, W, CvType.CV_64FC1);
Imgproc.distanceTransform(imgCenterZero, imgDist,
Imgproc.CV_DIST_L2, 5);
```

A seguir, as imagens `imgGray` (Figura 11) e `imgDist` são divididas em 4 quadrantes. Para o primeiro quadrante é realizada a operação "mínimo(`imgGray1`,`imgDist1`)". O mesmo ocorre para os outros três quadrantes.

Finalmente, basta encontrar o pixel máximo destes quatro quadrantes para encontrar os extremos. O resultado pode ser visto na Figura 14.

Para ajustar a imagem da Figura 14, é usada a função de perspectiva (detalhes na documentação *online* do OpenCV [11]). A Figura 15 apresenta o resultado do ajuste.

```
// perspectiva usando getPerspectiveTransform e
// warpPerspective

// 4 pontos extremos calculados anteriormente
Point[] ap = new Point[4];
ap[0] = p1; ap[1] = p2; ap[2] = p3; ap[3] = p4;

// padroniza tamanho da imagem em função do
// número de questões e respostas
int H2 = 20 *
(Integer.parseInt(TelaLoad.strNumRespostas) + 2);
int W2 = 20 *
(Integer.parseInt(TelaLoad.strNumQuestoes) + 2);
Mat imgOutt = new Mat(W2, H2, CvType.CV_8UC4);

// nova perspectiva
Point[] pts2 = new Point[4];
pts2[0] = new Point(0, 0);
pts2[1] = new Point(W2, 0);
pts2[2] = new Point(W2, H2);
pts2[3] = new Point(0, H2);

MatOfPoint2f pi = new MatOfPoint2f(ap);
MatOfPoint2f pf = new MatOfPoint2f(pts2);

Mat map_matrix =
Imgproc.getPerspectiveTransform(pi,pf);
Imgproc.warpPerspective(imgGray, imgOutt,
map_matrix, new Size(W2,H2));
```

Sobre a imagem `imgOutt` (Figura 15) são realizadas as verificações de cada quadrado pintado. Observe que esta imagem foi normalizada, de forma que cada quadrado tenha dimensão 20x20 pixels. Isto facilita as verificações das respostas de cada questão do quadro de respostas.

III. Resultados e Discussões

Realização de testes preliminares: Antes de aplicar um teste em uma turma (principalmente com muitos alunos), é sugerido imprimir um gabarito e algumas folhas de teste e realizar experimentos para testar o funcionamento do aplicativo MCTest em seu dispositivo móvel. Alguns modelos de testes estão disponíveis em http://vision.ufabc.edu.br/MCTest.

Preenchimento correto dos quadros: Os experimentos realizados indicam que as respostas dos alunos que não preencheram completamente os quadrados (fizeram apenas algumas linhas à caneta, um "X" ou até mesmo bolinhas, deixando muito do fundo branco exposto) não foram todas detectadas pelo MCTest, resultando em pontuação inferior. Neste caso, o professor pode destacar a resposta do aluno preenchendo com um lápis o quadrado de forma adequada. Quando uma questão é inválida (sem resposta ou com mais de uma resposta por questão), o aplicativo destaca a coluna em amarelo. Em um simples experimento realizado em duas turmas (veja http://vision.ufabc.edu.br/MCTest/Experiments), a primeira turma *classA2* com 41 testes com 4 tipos diferentes de gabaritos, apenas um dos testes apresentou problemas, que foi resolvido facilmente corrigindo com um lápis os quadrados pintados de forma incorreta (na pasta *ClassA2*, veja teste número 19 e a sua correção no teste número 42). A segunda turma *classB2*, com 64 testes, também com os mesmos 4 tipos diferentes de gabaritos, apenas um dos testes apresentou o mesmo problema (na pasta *classB2*, veja teste número 49). Portanto, é fortemente recomendado ao aluno pintar por completo o quadrado correspondente à resposta escolhida, de preferência com caneta preta. O filtro do MCTest considera marcados quadrados com pelo menos 50% da área pintada.

Fundo branco: ao fotografar**,** deixar algum fundo branco em torno do quadro de respostas (entre as linhas verdes da Figuras 1 e 2), evitando surgimento de ruídos nos extremos do quadro.

Iluminação: Ao fotografar um quadro de respostas, má iluminação pode afetar as correções. Assim, escolha uma ambiente bem iluminado, sem sombras, sem reflexões de luz, e que em todo o quadro de respostas tenha a mesma intensidade de luz.

Folha plana: O quadro de respostas deve estar em uma superfície plana ao ser fotografado, principalmente para testes com muitas questões, e, consequentemente, quadrados de respostas pequenos. Isso pode fazer com que uma resposta de uma questão fique cortada ao meio numa folha não plana, seguindo o modelo de solução implementado no MCTest.

Desempenho: Outro fator importante a ser observado na utilização do aplicativo é o tempo de processamento de cada prova. Uma versão preliminar, em MATLAB, executava correções quase instantaneamente, porém precisava de uma estrutura específica e muito menos portátil para sua execução [10]. Como esperado, o tempo de processamento é relativo à velocidade do dispositivo móvel utilizado. Mesmo assim, testando-o com diversas especificações de dispositivos foi possível obter um resultado que favorece a utilização do aplicativo móvel em comparação à versão para PC. Além da praticidade e mobilidade, o processo mostrou-se rápido até mesmo em celulares de baixo custo. O tempo total de correção, incluindo desde a obtenção de imagem até a exibição do resultado, corresponde a aproximadamente 7 segundos nos celulares com baixo poder de processamento, chegando a ser quase instantâneo em celulares mais poderosos. Celulares populares, como um Motorola XT918 utilizado em testes apresentaram tempo total de execução da tarefa de 3 segundos. O *smartphone Samsung Galaxy Note 2*, com Android versão 4.3 e processador 1.6 GHz Quad Core e câmera com 8 Mp, processa e corrige uma prova em menos de 1 segundo.

Trabalhos futuros: Versão para iOS e outros dispositivos móveis. Está sendo considerado criar uma versão para armazenar ou até corrigir os testes remotamente em um servidor, utilizando protocolos como o FTP, o que permitiria a cooperação de vários dispositivos para a correção de um mesmo Projeto. Nesta modalidade, o dispositivo móvel envia a foto ou a correção do teste para um servidor que realizaria o armazenamento ou até as correções. Partes do código para o dispositivo móvel e para o servidor já existem. No servidor, o código foi implementado em Python com a biblioteca OpenCV [11]. Outro estudo é sobre a integração dos dados com programas de gerenciamento de notas, comumente usados nas escolas e universidades, embora questões de segurança estejam envolvidas no controle de acesso e identificação automática dos testes para incorporação em algum sistema integrado de

ensino. Seria interessante também melhorar o processamento de imagens para aceitar "X" em vez de pintar todo o quadrado da alternativa da questão. Porém, quando isso foi testado, o aplicativo deixou de ser robusto.

CONCLUSÃO

Foi apresentado um corretor de testes de múltipla-escolha, implementado no sistema operacional Android utilizando biblioteca OpenCV [11]. As técnicas de processamento foram detalhadas neste trabalho, principalmente um método para detecção de extremos de objetos foi definido. Os testes realizados em várias dezenas de provas mostram que o corretor automático de testes de múltipla-escolha apresenta taxa de acerto de 100%, quando as questões são respondidas/pintadas corretamente. Alguns aperfeiçoamentos na segmentação das imagens estão sendo feitos para permitir que mesmo gabaritos preenchidos apenas parcialmente possam ser reconhecidos pelo MCTest.

REFERÊNCIAS

[1] K.M. Saipullah, A. Anuar, N.A. Ismail, e Y. Soo, "Measuring power consumption for image processing on Android smartphone", American Journal of Applied Sciences, vol. 9(12), pp. 2052-2057, 2012.

[2] Cienciamao. Disponível em: www.cienciamao.usp.br. Acesso em 10.04.15.

[3] J.A. Fisteus, A. Pardo, e N.F. Garcia, "Grading Multiple Choice Exams with Low-Cost and Portable Computer-Vision Techniques", Journal Of Science Education And Technology, vol. 22(4), pp. 560-571, 2013.

[4] Optical Mark Recognition. Disponível em: www.omrsolutions.com. Acesso em 10.04.15.

[5] A. Spadaccini, "A Multiple-Choice Test Recognition System based on the Gamera Framework", Document Image Analysis with the Gamera Framework, vol. 8, pp. 5-15, 2009.

[6] T.D. Nguyen, Q.H. Manh, P.B. Minh, L.N. Thanh, e T.M. Hoang, "Efficient and reliable camera based multiple-choice test grading system", International Conference on Advanced Technologies for Communications, 2011.

[7] D.J. Sen, R.N. Patel, e U.Y. Patel, "Modern Database Technology Needs Optical Mark Reading", International Journal of Pharmaceutical and Applied Sciences 1(2), 56, 2010.

[8] A.F. Mollah, N. Majumder, S. Basu, e M. Nasipuri, "Design of an Optical Character Recognition System for Camera-based Handheld Devices", International Journal of Computer Science Issues 8(1), 2011.

[9] K. Chinnasarn, e Y. Rangsanseri, "An image-processing oriented optical mark reader", Conference Series, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 3808, pp. 702–708, 1999.

[10] F.A. Zampirolli, J.A. Guilici-Gonzalez, e R. Neves. Automatic Correction of Multiple-Choice Tests using Digital Cameras and Image Processing. Workshop de Visão Computacional (WVC), Rio de Janeiro, 2013.

[11] OPENCV. *Open Source Computer Vision*. Disponível em: http://opencv.org. Acesso em 08.04.15.

[12] R.C. Gonzalez, e R. Woods. Processamento Digital de Imagens. São Paulo: Pearson Pentice Hall, 2010.

[13] L. S. Davis. "A survey of edge detection techniques." Computer graphics and image processing 4.3: 248-270, 1975.

[14] C. Lopez-Molina, B. De Baets, H. Bustince, "Quantitative error measures for edge detection." *Pattern Recognition* 46.4: 1125-1139, 2013.