Processamento de Cartões-Resposta Utilizando Visão Computacional

Giovanni Rosa Marcomini, Weslem Cristiano, Luís Fernando da Costa
Curso de Ciência da Computação
Universidade Estadual do Norte do Paraná

***Abstract*—Este artigo propõe uma abordagem baseada em visão computacional para a automação do processo de correção de cartões-resposta utilizados em processos seletivos. Utilizando Python e a biblioteca OpenCV, desenvolveu-se um algoritmo capaz de identificar e interpretar marcações em cartões fotografados com dispositivos convencionais, eliminando a necessidade de scanners especializados e aumentando a acessibilidade, agilidade e rastreabilidade da correção.**

***Index Terms*—Visão Computacional, Processamento de Imagens, Cartões-Resposta, OpenCV, Python.**

I. Introdução

O presente artigo sugere uma abordagem baseada em visão computacional para a problemática da necessidade de correção de um grande volume de cartões-resposta decorrentes do processo seletivo de ingresso aos cursos de graduação (vestibular). A automação dessa tarefa oferece diversas vantagens inerentes à informatização das atividades institucionais, como a redução de erros humanos, graças à aplicação de critérios padronizados e consistentes; a rastreabilidade, uma vez que os resultados podem ser armazenados e auditados digitalmente; e, por fim — mas não menos importante — a agilidade, possibilitando a correção de diversas avaliações em um tempo significativamente menor do que o exigido por métodos manuais.

Atualmente, organizações responsáveis pela aplicação de concursos, avaliações ou pesquisas de opinião frequentemente utilizam cartões-resposta como forma de gabarito, uma vez que sua correção pode ser realizada por scanners especializados. No entanto, esses dispositivos apresentam alto custo e baixa flexibilidade, o que limita seu uso em contextos com recursos restritos. Em contrapartida, a evolução do Processamento Digital de Imagens (PDI), estimulada pelo aumento de aplicações que utilizam imagens para oferecer diversas funcionalidades, tem viabilizado o desenvolvimento de sistemas capazes de interpretar cartões-resposta a partir de fotografias capturadas por câmeras de dispositivos convencionais.

Dessa forma, o algoritmo proposto tem como objetivo processar uma base de dados composta por 499 cartões-resposta digitalizados, oriundos da aplicação de uma prova objetiva contendo 60 questões de múltipla escolha, com alternativas entre A, B, C, D e E. O sistema é projetado para lidar com variações de iluminação, contraste, rotação e posicionamento entre os cartões, a fim de identificar com precisão as alternativas marcadas em cada questão. Em seguida, essas respostas são comparadas com um gabarito previamente definido, possibilitando a correção automatizada das avaliações.

II. Recursos Utilizados

A implementação do algoritmo proposto exigiu a utilização de recursos tecnológicos que combinasse praticidade no desenvolvimento com eficiência no processamento de imagens. Essa característica é fundamental para garantir uma taxa de acerto satisfatória na correção dos cartões-resposta, de modo a minimizar a necessidade de intervenção humana em eventuais auditorias manuais. Nesse sentido, optou-se pela utilização de ferramentas já consolidadas no campo de visão computacional, como a linguagem de programação Python, pela versatilidade e facilidade de integração e a biblioteca OpenCV que fornece a maior parte das operações necessárias para o processamento de imagens.

A. Python

Python é uma linguagem de programação de alto nível criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI), seu foco originalmente estava em dar suporte a trabalhos exigissem um volume de cálculos como aqueles realizados por físicos e engenheiros (BORGES, 2014). Contudo, apesar de dar atenção a estes domínios ela foi projetada para ser concisa e fácil legibilidade, tornando-a normalmente a primeira linguagem de programação de muitos desenvolvedores e cientistas de dados [3].

A linguagem fornece estruturas de alto nível (listas, dicionários, data/hora, complexos entre outras) e uma vasta coleção de módulos prontos para uso, com estes se referendo a arquivos Python (.py) que contém definições de funções, classes ou variáveis dos quais são possíveis de importar e reutilizar. Python conta com diversos frameworks que podem ser adicionados ao ambiente de desenvolvimento conforme a necessidade do projeto, o que acelera o desenvolvimento de software. Ela é Multiparadigma, dando suporte a programação modular, funcional, e orientada a objetos. A linguagem é interpretada através de bytecode pela máquina virtual Python, o que torna seu código portável [2].

B. OpenCV

OpenCV, abreviação de Open Source Computer Vision Library, é uma biblioteca de código aberto desenvolvida em C e C++, compatível com as principais plataformas, como Linux, Windows e macOS. Embora tenha sido originalmente concebida para linguagens de paradigma procedural e orientado a objetos, como C e C++, a biblioteca oferece suporte

a interfaces para outras linguagens, incluindo Python, Ruby e MATLAB. Isso é possível por meio de mecanismos de ligação (bindings), que permitem que funções escritas em Python, por exemplo, invoquem diretamente suas correspondentes implementadas em C ou C++. Essas funções são então executadas nativamente, e seus resultados são retornados à linguagem de alto nível de forma transparente para o desenvolvedor [1]

O OpenCV foi projetado visando a eficiência computacional e com um forte foco em aplicações em tempo real, sendo assim ela foi escrita em C otimizado para tirar proveito de processadores multicore. Ela oferece mais de 500 funções que abrangem diversas áreas tais como inspeção de produtos em fábricas, imagens médicas, segurança, interface do usuário, calibração de câmeras, visão estéreo e robótica [1]

III. Desenvolvimento

Esta seção aborda a organização do trabalho, descrevendo a estratégia adotada e os métodos aplicados que a tornam viável. São apresentados os passos seguidos no desenvolvimento do algoritmo, passando pelas etapas de pré-processamento das imagens dos cartões-resposta, segmentação das áreas de interesse — correspondentes aos campos destinados a cada alternativa das questões — e interpretação das respostas extraídas, que são posteriormente comparadas com um gabarito previamente definido para a prova em questão.

A. Pré-processamento

A etapa de pré-processamento tem como propósito preparar a imagem para as operações subsequentes de segmentação e análise, minimizando interferências visuais e promovendo a padronização da área de interesse. Inicialmente, o foco principal foi a detecção dos triângulos localizados nos cantos do cartão-resposta, os quais delimitam com precisão os limites da grade onde estão dispostas as marcações das respostas do participante. Esses marcadores visuais são fundamentais para o correto enquadramento e transformação da perspectiva da imagem

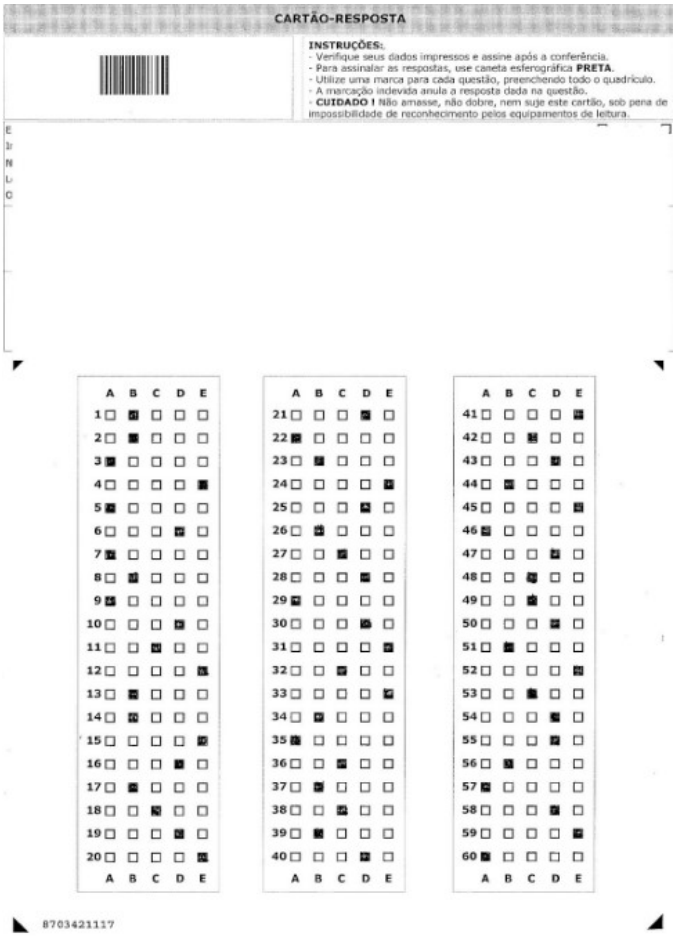


Fig. 1: Exemplo de cartão-resposta com a área de interesse delimitada por triângulos marginais.

O primeiro passo após carregar a imagem é a sua conversão para escala de cinza, essa transformação é necessária levando em conta o objetivo de facilitar a detecção de bordas e eliminar a informação de cor, que para este trabalho é irrelevante. A imagem em escala de cinza serve então de entrada para a função de filtragem espacial que aplica o desfoque Gaussiano. O filtro opera atenuando ruídos e diferenças abruptas nos níveis de cinza ao custo da nítidez da imagem. A imagem filtrada serve de parâmetro para o algoritmo de detecção de detecção de bordas Canny, que destaca os contornos da imagem para facilitar a localização dos triângulos à margem dos cartões-resposta. Com base nesse resultado, os contornos são identificados por meio da função cv2.findContours, e, posteriormente, simplificados através do método cv2.approxPolyDP, que aproxima suas formas a figuras geométricas com menor número de vértices. A partir dessa análise, são selecionadas apenas as figuras que possuem exatamente três vértices — ou seja, triângulos — e que satisfazem critérios mínimos de área, com o intuito de eliminar ruídos e formas indesejadas.

```
gray_tri = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur_tri = cv2.GaussianBlur(gray_tri, (5, 5), 0)
edges_tri = cv2.Canny(blur_tri, 50, 150)
contours, _ = cv2.findContours(edges_tri, cv2.
    RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

triangles = []
for cnt in contours:
   approx = cv2.approxPolyDP(cnt, 0.02 * cv2.
       arcLength(cnt, True), True)
   if len(approx) == 3 and cv2.contourArea(cnt) >
       150:
       M = cv2.moments(cnt)
       if M["m00"] != 0:
           cx = int(M["m10"] / M["m00"])
           cy = int(M["m01"] / M["m00"])
           triangles.append((cx, cy))
```

Código 1: Trecho de código fonte referente a aplicação dos filtros e identificação dos triângulos marginais

Fig. 2: Pares de triângulos detectados (contorno verde) na parte superior da grade de respostas

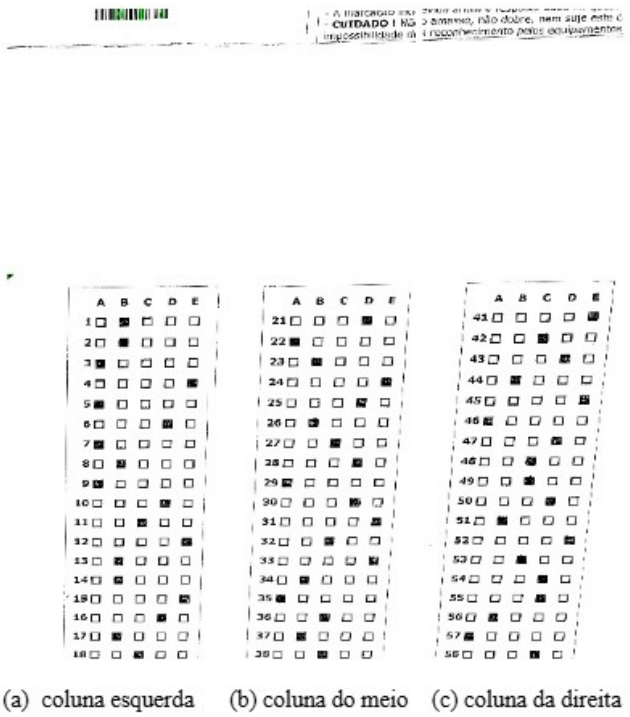
Fig. 3: Pares de triângulos detectados (contorno verde) na parte inferior da grade de respostas

B. Segmentação

A etapa de segmentação é outra parte fundamental do processamento digital de imagens, tratando da divisão da imagem em regiões ou objetos com características semelhantes. Nela o intuito foi realizar as correções de perspectiva para tratamento das distorções de geometria, advindos das variações na captura da imagem como inclinação e ângulo. Além disso, nesse estágio sucedeu-se a divisão da área útil do cartão-resposta em unidades menores, que possibilitou a identificação precisa de cada campo de resposta.

O ajuste de perspectiva é apoiado pelos triângulos marginais da grade de respostas, dos quais fornecem pontos de referência para o ajuste da imagem via homografia. Ao final do procedimento a saída gerada é uma imagem ortogonal e proporcional da grade de alternativas, o que ampara os próximos passos do processamento.

Na sequência, realizou-se a segmentação da grade de questões, inicialmente a partir de sua divisão em colunas. Cada coluna foi recortada aplicando-se um deslocamento vertical inicial fixo (ALTURA_INICIAL), bem como um corte ao final para eliminar áreas irrelevantes. Esse procedimento assegura que apenas a parte útil da imagem seja considerada na análise. Para cada cartão-resposta pré-processado, foram gerados três arquivos de imagem, cada um correspondente a uma das três colunas do documento, facilitando o processamento subsequente de cada linha individualmente.



(a) coluna esquerda (b) coluna do meio (c) coluna da direita

Fig. 4: Colunas recortadas a partir da grade de respostas, (a) coluna da esquerda, (b) coluna do meio, (c) coluna da direita.

```
col_width = largura // NUM_COLUNAS
columns = [warped_masked[:, i * col_width:(i + 1) *
    col_width] for i in range(NUM_COLUNAS)]
columns = [col[ALTURA_INICIAL:-85, :] for col in
    columns]
```

Código 2: Trecho de código fonte referente à divisão das colunas na imagem corrigida

A segmentação das alternativas de cada questão é realizada por meio da detecção de contornos nas linhas previamente extraídas das colunas da imagem. A estratégia adotada para o recorte de cada linha consiste na divisão proporcional da altura útil de cada coluna, com base na quantidade total de questões. Assim, cada coluna é segmentada verticalmente em 20 partes iguais, correspondendo a uma linha para cada questão, o que garante que todas as alternativas sejam isoladas de forma uniforme e estruturada para análise subsequente.

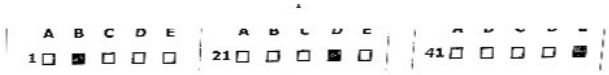


Fig. 5: Linhas recortadas a partir de cada uma das colunas segmentadas da grade de respostas

Em seguida, a função detectar_caixas_por_contorno utiliza o cv2.findContours para identificar os contornos presentes na versão binarizada de cada linha. Esses contornos são então filtrados com base em critérios geométricos, como largura,

altura, proporção entre largura e altura e área, a fim de selecionar apenas aqueles que provavelmente correspondem às caixas de marcação válidas.

Fig. 6: Alternativas identificadas (contorno em verde) para cada linha recortada dentro das colunas

```
if 8 < w < 50 and 8 < h < 50 and 0.6 < aspecto < 1.4
     and area > 30:
    caixas_detectadas.append((x, y, w, h))
```

Código 3: Trecho de código fonte referente à filtragem dos contornos detectados nas caixas de marcação da questão

C. Interpretação

A etapa de interpretação, dentro do fluxo de processamento de imagens, representa o momento em que os dados extraídos por meio da segmentação passam a ser analisados e compreendidos com base em seus atributos visuais. Ela envolve a análise das células segmentadas para determinar quais campos foram efetivamente marcados pelo candidato. Para cada célula, avalia-se a densidade de pixels escuros, com base na imagem binarizada previamente obtida.

Para identificar a alternativa escolhida em cada questão, o algoritmo contabiliza a quantidade de pixels brancos em cada caixa de resposta na imagem binarizada invertida, pois as marcações dos candidatos aparecem como regiões escuras na imagem original. A alternativa que apresenta maior quantidade de pixels brancos, indicando maior preenchimento, é selecionada como marcada, desde que ultrapasse um limiar mínimo pré-estabelecido. Caso nenhuma alternativa atinja esse critério, a questão é classificada como não respondida, garantindo assim a confiabilidade da interpretação.

```
for i, (x, y, w, h) in enumerate(caixas[:5]):
    regiao = linha_binaria[y:y + h, x:x + w]
    brancos = np.count_nonzero(regiao == 255)
    letra = letras[i]
    resultados[letra] = brancos
```

Código 4: Trecho de código fonte responsável por percorrer as caixas de marcação detectadas e medir a quantidade de pixels brancos.

Além disso, o sistema compara automaticamente as respostas extraídas em relação ao gabarito oficial, este que esta armazenado em um arquivo no formato CSV. Cada uma delas é classificada como correta e incorreta, o que determina o valor na computação dos acertos. Ao final do processamento, o sistema gera um relatório detalhado contendo todas as respostas detectadas (respostas.txt), indicando para cada questão qual alternativa foi marcada, qual seria a correta e se houve acerto ou erro. Além disso, calcula-se a quantidade total de acertos e o respectivo percentual de aproveitamento, permitindo uma análise objetiva do desempenho do candidato.

```
for col_idx, col in enumerate(columns):
    for i in range(QUESTOES_POR_COLUNA):
        ...
        marcada, marcacoes = analisar_linha(
            linha_bin, linha.copy(), numero_questao)
        correta = gabarito_correto[numero_questao -
            1].upper()
        acertou = marcada == correta
        ...
        relatorio.append(f"{numero_questao:02d}. 
            Marcada:{marcada} | Correta: {correta} {
            simbolo}")
        numero_questao += 1
```

Código 5: Trecho de código fonte responsável pela avaliação de cada questão e elaboração do relatório

Fig. 7: Relatório gerado a partir da análise das informações das linhas recortadas e o gabarito oficial.

IV. Resultados e Discussões

Para a avaliação da sua eficácia e eficiência foram processados um conjunto de dados composto por 499 imagens correspondentes as fotografias digitalizadas dos cartões-respostas da avaliação de cada participante da prova. Cada uma delas contava com uma resolução de 2479 x 3508 px. Ao longo da execução do algoritmo, cada imagem é submetida sucessivamente às fases do protocolo de processamento e interpretação descritas na seção anterior (subseções 3.1, 3.2 e 3.3), a fim

de extrair as alternativas. Na etapa final do processamento, os resultados obtidos pelo sistema são comparados com o gabarito oficial.

A acurácia global do sistema com relação a interpretação das respostas obtidas pelo sistema foi mensurada a partir da razão entre o número de respostas extraídas corretamente pelo mesmo e o total de comparações realizadas. Ademais, outras métricas como tempo de execução (segundos) e pico de memória utilizado também foram avaliadas. A fórmula que expressa este cálculo e tabela com as métricas do sistema apuradas são apresentadas abaixo.

$$\textit{Acurácia global} = \frac{\textit{Respostas corretas}}{\textit{Comparações realizadas}} \times 100; \quad (1)$$

TABLE I: Métricas do sistema avaliadas e seus respectivos resultados

Métrica	**Valor**
Acurácia global	69.70%
Total de comparações válidas	14.656
Total de acertos	10.215
Tempo de execução	1.00 segundo(s)
Pico de memória	9601.69 kb

Fonte: Autoria própria.

REFERENCES

[1] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, Sebastopol: O'Reilly Media, 2008.

[2] L. E. Borges, *Python para Desenvolvedores: Aborda Python 3.3*, São Paulo: Novatec Editora, 2014.

[3] K. Walker, "6 Reasons Why Python is the Easiest Coding Language to Learn First," CodeOp. Available: https://codeop.tech/6-reasons-why-python-is-the-easiest-coding-language-to-learn-first/. Accessed: May 18, 2025.