MAC0468/MAC5915 - Laboratório de Visão Computacional e Processamento de Imagens

Relatório da Atividade #1

Nome: Andrew Ijano Lopes **NUSP:** 10297797

1 Introdução

Com as frequentes avaliações empregadas para grandes conjuntos de alunos, a existência de uma ferramenta para leitura e correção automatizada delas se torna essencial. Um dos softwares criados para esse fim é o *Auto Multiple Choice* (AMC) (Bienvenüe, 2008).

Dessa forma, este relatório descreve o programa desenvolvido para fazer leitura de códigos de prova, implementando os métodos *Simple Blob Detector* e *Determinant of Hessian* para identificação das marcações circulares nos cantos.

Na Seção 2 é feita uma descrição geral do programa implementado, na Seção 3 é mostrado o modo de uso do programa, na Seção 4 são apresentados os resultados dos experimentos feitos e na Seção 5 é feita uma discussão final sobre os resultados.

2 Programa desenvolvido

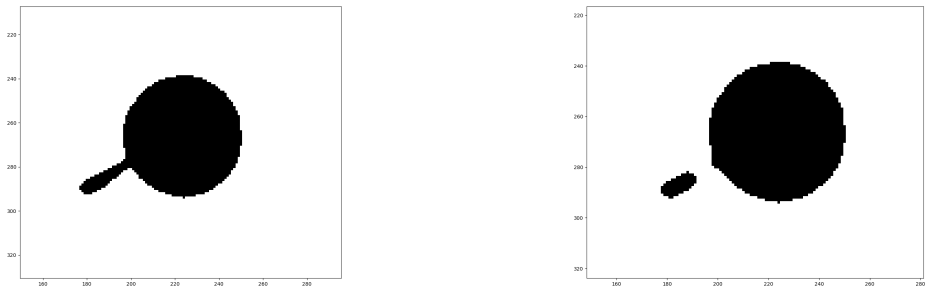
A estrutura geral do programa responsável por ler as informações da prova está descrito no Programa 1.

Programa 1 Leitor de prova.

```
1  FUNCAO leia_a_prova(imagem)
2      leia a imagem
3      pré processe a imagem
4      identifique as marcações circulares
5      corrija a rotação e perspectiva da prova
6      leia o código da prova
7      se o código é inválido
8          rotacione a imagem em 180°
9          leia novamente o código da prova
10     se for a primeira página, leia o número USP
11     devolva as informações lidas da prova
12 fim
```

2.1 Pré processamento

O pré processamento consiste de quatro operações: desfocar a imagem, converter para escala de cinza, aplicar *adaptative threshold* e operação de *closing*. O desfoque, feito através da operação de *Gaussian Blur* (*Image filtering* s.d.), é usado para remover os ruídos da imagem. A conversão de BGR para escala de cinza simplifica a imagem e necessário para os algoritmos de detecção de *blobs*. A operação de *adaptative threshold* é usado para binarizar uma imagem tratando de possíveis variações de luminosidade na folha. Por fim, a operação morfológica de *closing* é usada para remover marcas ou ruídos ao redor das marcações circulares como mostra a Figura 1.



(a) *Sem closing* **(b)** *Com closing*

Figura 1: *Diferença da aplicação da operação de closing em uma marcação circular com ruído*

2.2 Identificação das marcações circulares

Para a identificação das marcações circulares, foram utilizados dois métodos: *Simple Blob Detector* e *Determinant of Hessian*, detalhados em breve. Cada um desses métodos identifica um conjunto de marcações circulares na prova.

Após essa identificação, são extraídos os quatro pontos mais perto dos cantos, ordenados no sentido horário. Isso por ser definido da seguinte forma: dado um conjunto K de pontos (x, y) identificados, o conjunto ordenado de quatro pontos K' é dado por:

$$\begin{aligned} K' := (& \arg\min_{(x,y)\in K}(y+x), \\ & \arg\min_{(x,y)\in K}(y-x), \\ & \arg\max_{(x,y)\in K}(y+x), \\ & \arg\max_{(x,y)\in K}(y-x)). \end{aligned}$$

Ainda, se, a partir das identificações, a imagem estiver em paisagem, os pontos são reordenados

Para permitir a identificação a partir de imagens de diferentes resoluções, o programa utiliza uma constante que define a razão entre a raíz quadrada da área da prova e o raio

de uma marcação circular, igual a 0,00725. Essa constante é usada para calcular o raio esperado r da marcação circular para uma imagem dada.

Simple Blob Detector

Esse método utiliza a classe `SimpleBlobDetector` do OpenCV, que aceita vários parâmetros na sua configuração. Como esperamos encontrar regiões circulares pretas, o *threshold* foi definido no intervalo de 0 e 60; os parâmetros de circularidade, convexidade e inércia mínimos foram definidos como 0,8; e para a área mínima utilizou-se o resultado de $\pi \cdot (r - r/2)^2$, onde r é o raio esperado. Esses valores proporcionam uma margem para pequenas variações na imagem.

Determinant of Hessian (DoH)

Esse método utiliza `blob_doh` do scikit-image (Walt *et al.*, 2014). Ele detecta os *blobs* encontrando máximos da determinante do hessiano da imagem. Como o *Laplace of Gaussian*, esse método também utiliza um kernel gaussiano nos seus cálculos mas é considerado muito mais rápido. Como parâmetros da função, o desvio padrão do kernel varia no intervalo $[r - r/8, r + r/2]$, onde r é o raio esperado, com 5 valores intermediários considerados e um *threshold* de 0,06.

2.3 Correção da rotação e perspectiva

Para a correção da rotação e perspectiva, é criada uma matriz de transformação a partir dos pontos identificados e a matriz é aplicada na imagem por uma interpolação linear.

Supondo que as marcações circulares foram identificadas corretamente, o resultado da transformação sempre produzirá uma imagem na posição retrato. Por isso, caso haja falha na identificação do código, a única possibilidade é a prova estar rotacionada em 180°.

2.4 Leitura dos códigos

Com a rotação corrigida, a leitura dos códigos é feita localizando seus respectivos retângulos e, para cada quadrado, sua proporção de pixels brancos é calculada.

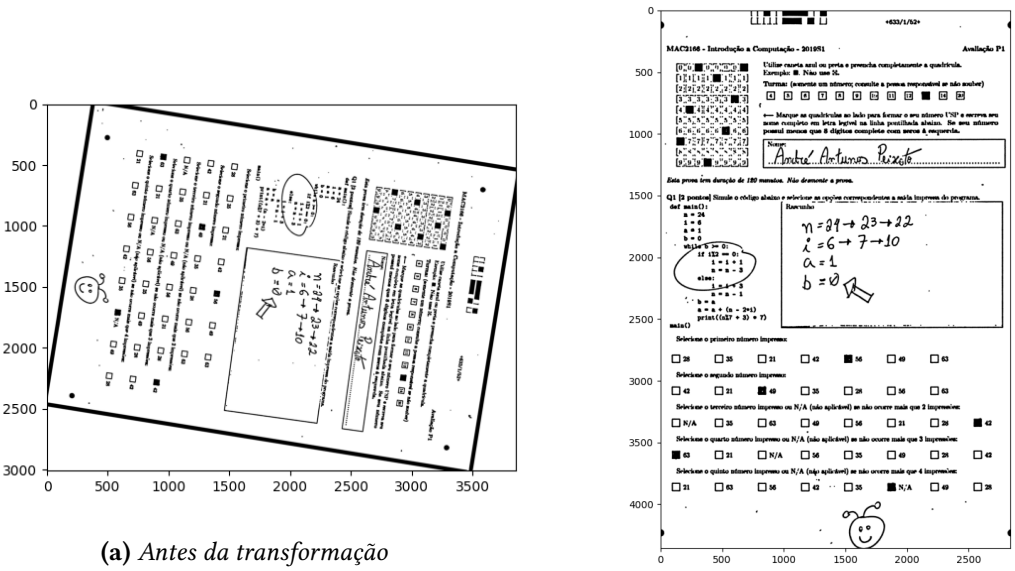
Para o retângulo do código da prova, um quadrado é considerado preenchido se menos de 40% da sua área possui pixels brancos. Já para o retângulo do número USP, o dígito considerado é aquele com a menor quantidade de pixels brancos.

3 Uso do programa

O programa desenvolvido pode ser executado a partir do arquivo `read_test.py` com Python 3.9.

Primeiro, é necessário instalar as dependências com:

```
pip install -r requirements.txt
```



(a) *Antes da transformação*

(b) *Depois da transformação*

Figura 2: *Resultado da correção da rotação e perspectiva da imagem de uma prova*

Em seguida, dada a imagem de uma prova `amc001.jpg`, a leitura da prova pode ser feita das seguintes formas:

1. Utilizando o método *Simple Blob Detector*

```
python read_test.py amc001.jpg --detector simple
```

2. Utilizando o método *Determinant of Hessian*

```
python read_test.py amc001.jpg --detector doh
```

Assim, o resultado de uma leitura bem sucedida é impresso na saída padrão como no exemplo abaixo

```
Successful detection:
test number: 633
page: 1 code: 52
nusp: 74091630
```

4 Resultados

Para validar a eficácia do programa desenvolvido, foram feitos experimentos utilizando dois *datasets*:

1. `scans`, com 240 provas escaneadas de forma uniforme;

2. `scans2`, com 42 provas com diferentes características, iluminações e rotações.

Nesse experimento, para cada método de identificação das marcações, o programa foi executado para cada prova dos *dataset*. Uma leitura era considerada bem sucedida apenas se *o código* lido era válido.

O resultado do experimento pode ser encontrado na Figura 3. É possível observar que ambos os métodos leram corretamente o mesmo número de provas. Em particular, apenas quatro provas (do `scans2`) falharam na leitura, pois possuíam uma rasura em uma das marcações, fazendo com que o círculo não fosse identificado, como pode ser visto na Figura 4.

Apesar do mesmo resultado na leitura das provas, os dois métodos possuem uma grande diferença de tempo de execução. Observa-se que o método *Determinant of Hessian* pode ser até 7 vezes mais custoso que o *Simple Blob Detector*. Como as imagens em `scans` tinham resoluções superiores à média do conjunto `scans2`, seu tempo de execução também foi superior; porém, como a variação das imagens em `scans2` é maior e pode exigir mais de uma leitura para identificar o código, o desvio padrão das amostras também é maior. Dessa forma, ao medir o tempo médio em razão da quantidade de pixels da imagem, é possível ver uma diferença similar entre os métodos de até 5 vezes para ambos os conjuntos de dados.

5 Conclusão

Assim, tem-se que o algoritmo proposto e implementado funciona corretamente em uma variedade de cenários em ambos os métodos de identificação de círculos. Um caso problemático encontrado são de rasuras em cima das marcações circulares nos cantos.

Como o método do *Simple Blob Detector* se mostrou quase 7 vezes mais rápido que o *Determinant of Hessian*, ele é o método mais indicado para o uso do programa.

Referências

[Bienvenüe 2008] Alexis Bienvenüe. *AMC*. 2008. URL: https://www.auto-multiple-choice.net/index.en.

[*Image filtering* s.d.] *Image filtering*. URL: https://docs.opencv.org/3.3.1/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1.

[Walt *et al.* 2014] Stéfan van der Walt *et al.* "Scikit-image: image processing in Python". Em: *PeerJ* 2 (jun. de 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

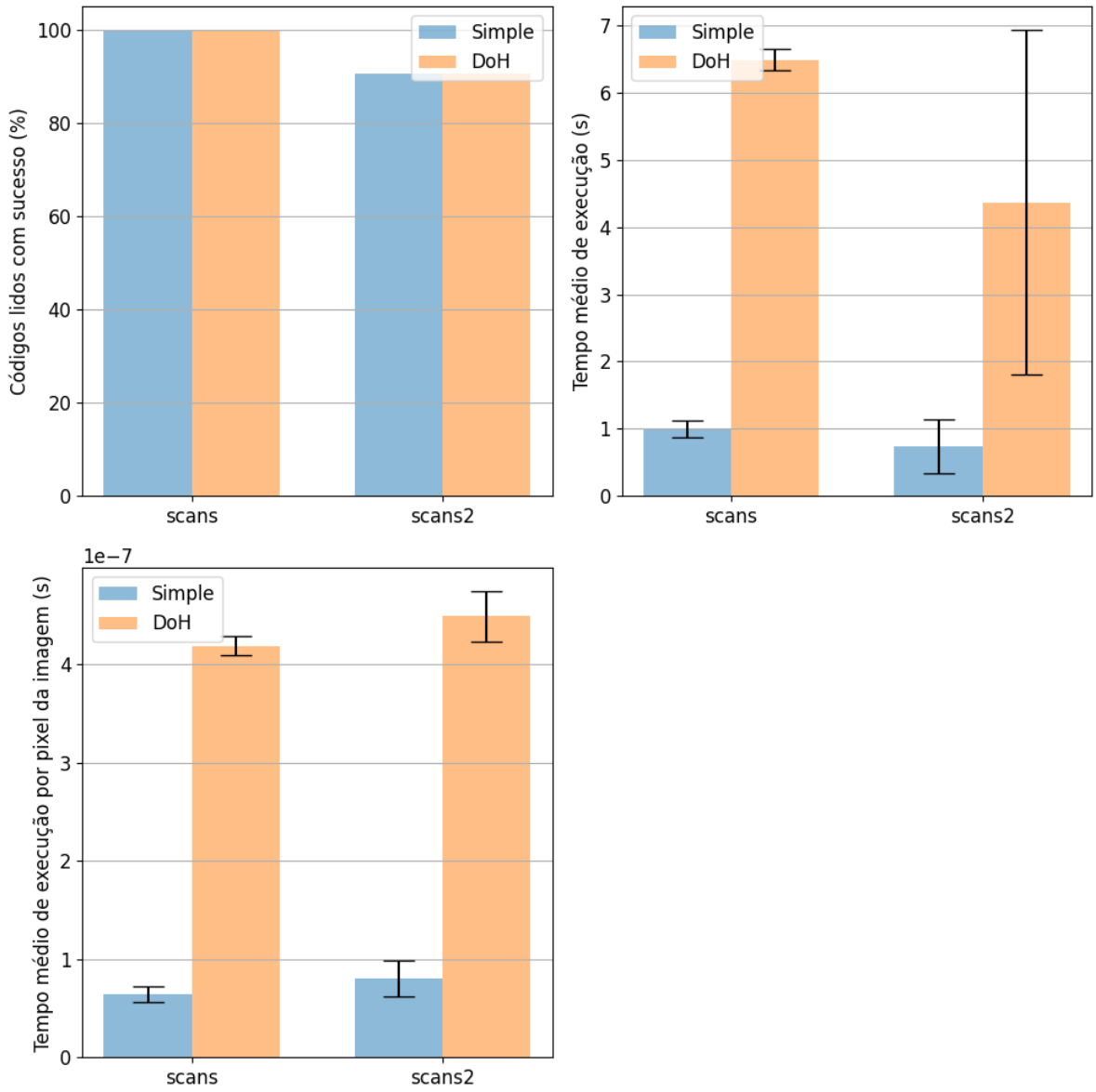


Figura 3: *Comparação da taxa de sucesso e tempo de execução para diferentes métodos e datasets*

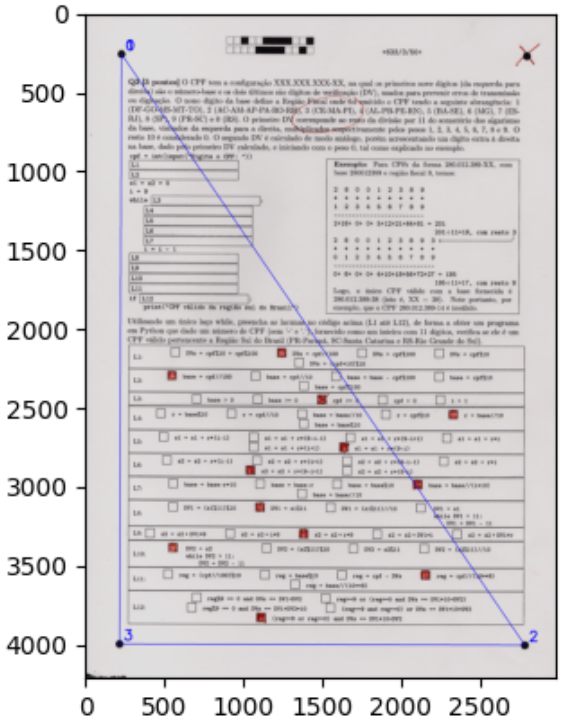


Figura 4: *Exemplo da identificação mal sucedida de uma prova com rasura em uma das marcações*