| | |
|---|---|
| INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui | **INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**<br>**Campus Birigui**<br>**Bacharelado em Engenharia de Computação** |

| | |
|---|---|
| **Disciplina:** Processamento Digital de Imagem | **Atividade** |
| **Professor:** Prof. Dr. Murilo Varges da Silva | **Data:** 12/09/2023 |
| **Nome do Aluno:** Henrique Akira Hiraga | **Prontuário:** BI300838X |

# DESAFIO STANFORD

No dia 12 de Setembro de 2023 foi realizado em sala o desafio Stanford, ele se baseia na elaboração de dois códigos para o processamento digital de imagem. Como desafio, foi proposto um primeiro desafio de analisar duas imagens de PCB, uma normal e outra com defeito, e como objetivo, era necessário encontrar o erro computacionalmente utilizando as duas imagens. O segundo desafio tinha como objetivo detectar a movimentação em um vídeo.

**Análise de Imagem PCB**

**Objetivo:**

O objetivo deste código é comparar duas imagens de placas de circuito impresso (PCB), uma sem defeito e outra com defeito, e identificar as áreas onde as diferenças ocorrem.

**Descrição do Código:**

1. **Importação de Bibliotecas:**
   - O código começa importando as bibliotecas cv2 (OpenCV) e numpy para processamento de imagem e manipulação de matrizes.

2. **Carregamento das Imagens:**
   - Duas imagens são carregadas usando a função cv2.imread(). A primeira imagem, 'pcbCroppedTranslated.png', representa a placa sem defeito, e a segunda imagem, 'pcbCroppedTranslatedDefected.png', representa a placa com defeito. Ambas as imagens são carregadas como imagens em escala de cinza.

3. **Verificação de Carregamento:**
   - O código verifica se as imagens foram carregadas corretamente. Se uma das imagens não puder ser carregada, ele imprime uma mensagem de erro.

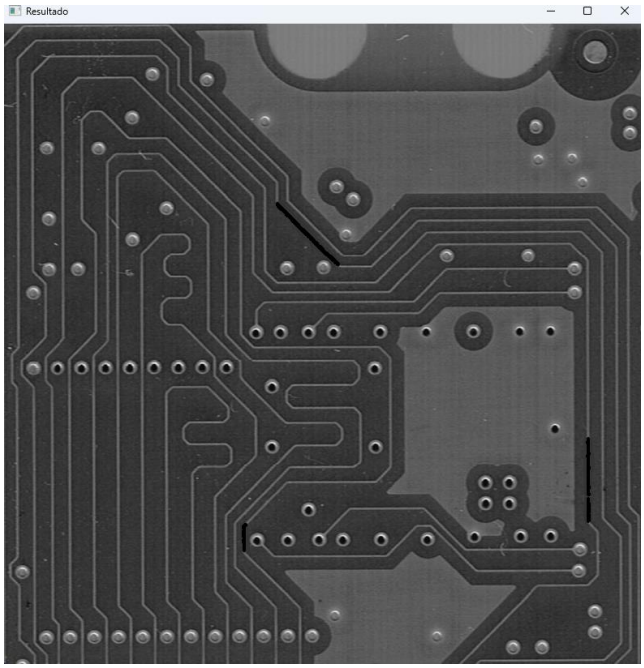
4. **Subtração de Imagens:**
    - O código calcula a diferença entre as duas imagens usando cv2.absdiff(). Isso resulta em uma nova imagem que destaca as áreas onde as imagens são diferentes.

5. **Aplicação de Limiarização:**
    - É aplicado um limiar para destacar as diferenças significativas. O valor de limiar é definido como 30. A função cv2.threshold() é usada para criar uma imagem binária onde os pixels que excedem o limiar são definidos como brancos (255) e os pixels abaixo do limiar são definidos como pretos (0).

6. **Encontrar Contornos:**
    - O código encontra contornos nas áreas destacadas da imagem binária usando cv2.findContours(). Os contornos são armazenados na variável contornos.

7. **Desenho de Contornos:**
    - Os contornos encontrados são desenhados na imagem original da placa com defeito usando cv2.drawContours(). Eles são desenhados em vermelho (0, 0, 255) com uma espessura de 2 pixels.

8. **Exibição do Resultado:**
    - O resultado final, que inclui a imagem da placa com defeito com os contornos destacados, é exibido em uma janela com a função cv2.imshow(). A janela é fechada quando uma tecla é pressionada.




**Detecção de Movimento em Vídeo**

**Objetivo:**

O objetivo deste código é capturar um vídeo (no caso, 'output.avi') e detectar áreas de movimento dentro do vídeo em tempo real. Ele usa o algoritmo de subtração de fundo MOG2 para identificar áreas onde ocorreu movimento.

**Descrição do Código:**

1. **Importação de Bibliotecas:**
   - O código começa importando as bibliotecas cv2 (OpenCV) e numpy para processamento de imagem e manipulação de matrizes.

2. **Inicialização da Captura de Vídeo:**
   - O objeto de captura de vídeo é inicializado com cv2.VideoCapture('output.avi'), que carrega o vídeo a ser processado ('output.avi' no exemplo).

3. **Inicialização do Algoritmo de Subtração de Fundo:**
   - O algoritmo de subtração de fundo MOG2 é inicializado com cv2.createBackgroundSubtractorMOG2(). Este algoritmo é usado para identificar áreas em movimento no vídeo.

4. **Loop Principal:**
    - O código entra em um loop infinito (while True) para processar cada quadro do vídeo.

5. **Leitura do Próximo Quadro:**
    - A função cap.read() é usada para ler o próximo quadro do vídeo. A variável frame contém o quadro lido, e ret indica se a leitura foi bem-sucedida.

6. **Aplicação do Algoritmo de Subtração de Fundo:**
    - O algoritmo MOG2 é aplicado ao quadro atual com fgbg.apply(frame), resultando em uma máscara binária (fgmask) que destaca as áreas onde o movimento foi detectado.

7. **Limiarização:**
    - O código aplica uma operação de limiarização à máscara fgmask usando cv2.threshold(). Isso pode ser útil para destacar as regiões de movimento de interesse. Neste exemplo, um limiar de 128 é aplicado, transformando os valores acima de 128 em 255 (branco) e os valores abaixo em 0 (preto).

8. **Encontrar Contornos:**
    - Os contornos das áreas de movimento são encontrados na máscara usando cv2.findContours(). Os contornos são armazenados na variável contornos.

9. **Desenho de Contornos:**
    - Os contornos encontrados são desenhados na imagem original (frame) com a cor verde (0, 255, 0) e uma espessura de linha de 2 pixels.

10. **Exibição da Imagem Resultante:**
    - A imagem resultante com os contornos de movimento destacados é exibida em uma janela com o título 'Detecção de Movimento' usando cv2.imshow().

11. **Condição de Saída:**
    - O código aguarda uma tecla ser pressionada (30 milissegundos) e verifica se a

tecla 'Esc' (código ASCII 27) foi pressionada. Se 'Esc' for pressionado, o loop é interrompido.

**12. Liberação de Recursos:**

- Após a saída do loop, o objeto de captura de vídeo é liberado com cap.release() e todas as janelas são fechadas com cv2.destroyAllWindows().