Disciplina: Visão Computacional

Professor: Kelson Romulo Teixeira Aires

Autor: Luis Felipe do Nascimento Moura

Descrição: Este relatório aborda conceitos e técnicas utilizados no processamento digital de imagens e visão computacional, exemplificados em um código prático que realiza a detecção de movimento e aplicação de rastro em vídeos. São explorados tópicos como a conversão de imagens para escala de cinza, aplicação de desfoque gaussiano, operações morfológicas e criação de máscaras de movimento.

Relatório Atividade Prática 01

Teresina-PI 10/2024

1. Captura de Vídeo e Leitura de Frames

O processo de leitura de um vídeo é iniciado com a função `cv2.VideoCapture()`, que abre um arquivo de vídeo especificado (neste caso, "videoVisao.mp4"). Também é possível utilizar um vídeo ao vivo, bastando passar o valor 0 como parâmetro da função. Em seguida, a função `video.read()` é chamada em um loop para capturar e processar os frames do vídeo um a um.

Cada frame é representado como uma matriz tridimensional (imagem colorida) ou bidimensional (imagem em escala de cinza), onde cada elemento da matriz corresponde a um pixel da imagem.

2. Conversão para Escala de Cinza

A conversão do frame para escala de cinza foi feita usando a função cv2.cvtColor(). A imagem em tons de cinza é importante porque simplifica o processamento e para a natureza do projeto a cor se demonstrou desnecessária, uma vez que o foco estava nas mudanças de intensidade entre frames. A conversão reduziu a quantidade de informações a serem processadas e facilitou a aplicação de técnicas de processamento de imagem.

```
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

3. Desfoque Gaussiano

O desfoque gaussiano foi aplicado com a função cv2.GaussianBlur(), suavizando a imagem para reduzir ruídos e detalhes pequenos que estavam interferindo no processo de detecção de movimento. O filtro gaussiano foi escolhido por ser um dos mais comuns no processamento de imagens, e pela sua natureza de atenuar detalhes e ruídos sem a perda de características de maior escala.[referência]

```
gray_frame = cv2.GaussianBlur(gray_frame, (21, 21), 0)
```

4. Detecção de Movimento por Subtração de Background

Para detectar movimento, o código utiliza a técnica de **subtração de background**, onde a diferença entre o frame atual e o frame anterior é calculada com cv2.absdiff(). Pixels que mudam significativamente de um frame para outro indicam movimento. [referencia]

```
frame_diff = cv2.absdiff(previous_frame, gray_frame)
```

5. Threshold

Após calcular a diferença entre frames, a função cv2.threshold() é usada para aplicar um limiar de intensidade, esse limiar precisou ser definido para direcionar a detecção de movimento para apenas áreas relevantes. Pixels cuja diferença excede o valor de threshold são destacados como movimento. Isso converte a imagem de diferença em uma imagem binária, onde áreas de movimento são marcadas com valores brancos (255) e o fundo é preto (0). [referência]

```
_, thresh = cv2.threshold(frame_diff, 25, 255, cv2.THRESH_BINARY)
```

6. Operações Morfológicas

Operações morfológicas, como a **fechamento morfológico** (ou **closing**), foram aplicadas para eliminar pequenos buracos ou ruídos na imagem binária (gerada pela Threshold). Isso foi feito usando um elemento estruturante (kernel) com a função cv2.morphologyEx(). Essas operações ajudaram a conectar regiões de movimento fragmentadas, melhorando a integridade dos objetos detectados. [referência]

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
```

7. Criação de Máscara de Movimento

O código cria uma máscara de movimento, onde os pixels em movimento são destacados com a cor vermelha (apenas para efeitos de depuração visual). Isso é feito ao aplicar a máscara binária de threshold sobre o frame original, colorindo os pixels detectados em vermelho.

```
mask = np.zeros_like(frame, dtype=np.uint8)
mask[thresh > 0] = [0, 0, 255]
```

8. Rastro de Movimento

O conceito de rastro é implementado ao utilizar uma imagem acumulativa (trail_frame), onde o movimento detectado nos frames anteriores é progressivamente atenuado. A variável trail_frame armazena os rastros dos objetos em movimento, e sua intensidade vai diminuindo ao longo do tempo com o fator de esmaecimento (decay_rate).

```
trail_frame = cv2.multiply(trail_frame, 1 - decay_rate)
```

9. Combinação de Frames

O frame original com a máscara vermelha de movimento é combinado com o rastro usando cv2.addWeighted(). A função permite ajustar a intensidade de cada uma das camadas sobrepostas, criando um efeito visual mais suave. Esse método de combinação ponderada cria um efeito visual onde o movimento recente é destacado, enquanto os rastros antigos desaparecem gradualmente.

```
combined_frame = cv2.addWeighted(frame_with_trail, 0.7,
trail_frame.astype(np.uint8), 0.3, 0)
```

10. Salvamento de Frames

Os frames processados são salvos como imagens JPEG na pasta especificada, usando a função cv2.imwrite(). A numeração dos arquivos garante que cada frame seja salvo em sequência. Isso foi implementado para facilitar a depuração do fluxo de detecção de movimento, que pode ser visto na pasta “frames_capturados” que é gerada/sobreposta a cada execução do programa.

```
frame_filename = f"{output_folder}/frame_{frame_count:04d}.jpg"
cv2.imwrite(frame_filename, combined_frame)
```

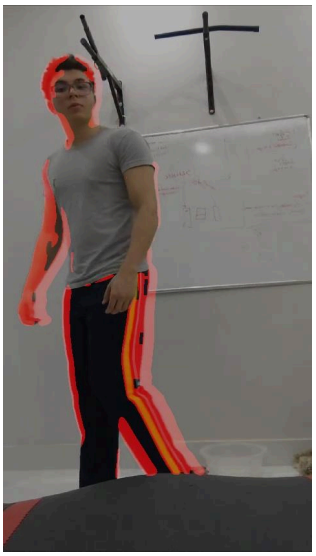
11. Encerramento e Liberação de Recursos

Ao final do processamento, o vídeo é liberado da memória com video.release() e as janelas abertas pelo OpenCV são fechadas com cv2.destroyAllWindows(), liberando os recursos utilizados pelo programa.

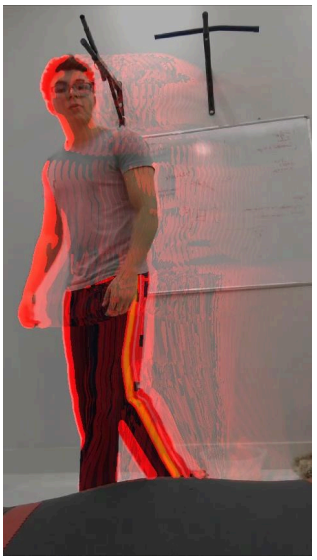
12. Considerações Finais

Gostaria de reforçar que a depuração detalhada da execução do código pode ser vista na pasta “frames_capturados”, onde a detecção de movimento pode ser vista com mais calma

exemplos:



detecçãoDeMovimentoContorno.py



detecçãoDeMovimentoExpressiva.py