Relatório do Projeto Demonstrativo 1: Explorando o OpenCV

Aluno: Jessé Barreto de Barros
Matrícula: 17/0067033
Sistemas Mecatrônicos - PPMEC-UnB
Disciplina: Visão Computacional
Turma: A
Data: 01/04/2017

1. Objetivos

O principal objetivo deste projeto demonstrativo é a de apresentar e possibilitar a familiarização do aluno com a ferramenta que será utilizada no decorrer da disciplina para o desenvolvimento e uso de algoritmos voltados para a visão computacional e processamento de imagens, o OpenCV[1] (`http://opencv.org/`).

2. Introdução

2.1. OpenCV

O OpenCV [1] é uma ferramenta, um conjunto de bibliotecas escritas em C/C++, desenvolvida para a utilizada no desenvolvimento de aplicações na área de visão computacional.

Além de ser multiplataforma o OpenCV também permite a sua utilização nas linguages de programação Python e Java. A otimização do OpenCV permite o seu uso em aplicações em tempo-real e que possuem intenso foco em desempenho.

O OpenCV é uma ferramenta *open-source* e possui uma lincensa que permite abertamente o seu uso para desenvolvimento acadêmico ou comercial. Por esse motivo o OpenCV é uma das principais ferramentas em projetos de áreas como a robótica e o processamento de imagens.

2.2. Cores

De acordo com [2], as cores são percebidas pelo sistema de percepção visual humano na identificação de luzes com diferentes comprimentos de onda. As cores são um tema que abrange diferentes conhecimentos como a física por trás da sua geração, a sua relação com o sistema visual humano e as diferentes maneiras que essa informação pode ser extraída e descrita digitalmente através de diferentes espaços de cores, tema que será descrito com mais detalhe no subtópico 2.3.

2.3. Espaço de Cores

A descrição e a representação correta das cores em um espaço uniforme e padronizado é de grande importância para finalidades comerciais e acadêmicas porque permite a reprodutibilidade da representação de um mesma cor em diferentes dispositivos. Normalmente, conforme descrito em [1], um espaço de cores utiliza de três valores para representar qualquer luz colorida em suas formas primárias. Todas as outras cores são combinações ponderadas dessas três cores.

Durante esse trabalho irá ser utilizado o espaço de cores RGB e a representação em escala de cinza.

2.3.1 Espaço de Cores RGB

Nesse espaço de cores as cores são representadas pelas cores primárias: Vermelho Verde e Azul. Imagens que utilizam cada um dos seus canais possuindo discretizações da intensidade luminosa para cada uma dessas cores primárias. O OpenCV, no entanto, utiliza como padrão de cores BGR para armazenar as imagens, ou seja, trocando as posições dos canais das cores Azul e Vermelho. Mas essa diferença pode ser facilmente revertida para RGB.

2.3.2 Representação em *Grayscale*

O nível de cinza, ou *grayscale*, é uma forma de representar a luz apenas pela sua intensidade e não pelas suas diferentes componentes com diferentes comprimentos de onda.

Dispositivos que utilizam esse padrão possuem imagens que variam entre as cores branca e preta com diferentes tonalidades de cinza entre essas cores. Essa representação armazena apenas um canal e por isso podem utilizar menos recursos computacionais para serem representadas ou alteradas através de algoritmos.

3. Metodologia

O projeto foi divido em quatro requisitos:

- **Requisito 1**: Elaborar uma aplicação que abra um arquivo de imagem (tipo JPG) e que permita ao usuário clicar (botão esquerdo do mouse) sobre um ponto na área da imagem na tela e após realizado o clique mostre no terminal a coordenada do ponto (x,y) na imagem, informando os valores do pixel RGB, quando a imagem for colorida ou mostrando o valor da intensidade do pixel quando a imagem for em nível de cinza (grayscale).
- **Requisito 2**: Repita o procedimento desenvolvido no Requisito 1, de forma a criar uma rotina que além de atender os itens do Requisito 1, após o clique com o botão esquerdo, mude a cor na imagem aberta em tela de todos os pixels da imagem que foi selecionado com valores variando em 5% do valor do pixel que foi clicado para a cor vermelha. Observem que isto deve estar preparado para imagens coloridas e também em nível de cinza.
- **Requisito 3**: Repita o procedimento desenvolvido no Requisito 2, em que ao invés de abrir uma imagem, abra um arquivo de vídeo (padrão avi ou x264) e realize os mesmos procedimentos do Requisito 2 durante toda a execução do vídeo.
- **Requisito 4**: Repita o procedimento desenvolvido no Requisito 3, em que ao invés de abrir um arquivo de vídeo, a aplicação abra o streaming de vídeo de uma câmera USB conectada ao computador e realize todos os procedimentos solicitados no requisito 3.

3.1. Requisito 1

Para o requisito 1 foi implementada uma aplicação em C++ utilizando o OpenCV que abre uma imagem padrão, lena512.jpg, que deve estar armazenada junto com os binários da aplicação, ou o caminho de uma segunda imagem como parâmetro da execução do binário via terminal.

No código fonte as seguintes etapas foram efetuadas para realizar esse requisito:

- Abre a imagem e a salva em uma classe `cv::Mat` com o método `cv::imshow`.
- Captura os eventos do mouse sobre a imagem aberta com o método `cv::setMouseCallback` e chamando a função `mouseHandler` para gerir esses eventos.
- Imprime as coordenadas da imagem e os valores dos canais de cores do pixel daquela coordenada.
- A aplicação é finalizada quando o usuário aperta alguma tecla no teclado.

3.2. Requisito 2

Para o requisito 2 foi implementada uma aplicação em C++ utilizando o OpenCV que abre uma imagem padrão, lena512.jpg, que deve estar armazenada junto com os binários da aplicação, ou o caminho de uma segunda imagem como parâmetro da execução do binário via terminal.

No código fonte as seguintes etapas foram efetuadas para realizar esse requisito:

- Executa os mesmos procedimentos do **Requisito 1**.
- No método `mouseHandler` cria uma máscara para ser aplicada na imagem dado os valores dos pixels daquela coordenada utilizando a função `cv::inRange`.
- Repinta os pixels da imagem utilizando a máscara criada para aquela determinada cor com o método `cv::Mat::setTo`.

3.3. Requisito 3

Para o requisito 3 foi implementada uma aplicação em C++ utilizando o OpenCV que abre um video padrão, viptraffic.mp4, que deve estar armazenada junto com os binários da aplicação, ou o caminho de um segundo video como parâmetro da execução do binário via terminal.

No código fonte as seguintes etapas foram efetuadas para realizar esse requisito:

- Abre um vídeo e o salva em uma classe `cv::VideoCapture`.
- Salva o próximo *frame* do vídeo em um objeto da classe `cv::Mat` até o usuário apertar algum botão do teclado. Caso o vídeo acabe ele é reiniciado.
- Captura os eventos do mouse sobre a imagem aberta com o método `cv::setMouseCallback` e executa a função `mouseHandler` para gerir esses eventos.
- No método `mouseHandler` imprime a posição e os valores do pixel daquele frame do vídeo, semelhante ao **Requisito 1**, porém salva a cor daquele pixel em um vetor de cores do tipo `std::vector` utilizando o método `std::vector::push_back`.
- Executa a função `paintColoursToRed` que caso o vetor de cores não esteja vazio, cria uma máscara para aquela cor naquele determinado *frame* utilizando a função `cv::inRange` e aplica essa máscara pintando os pixels de vermelho com o método `cv::Mat::setTo`.

3.4. Requisito 3

Para o requisito 3 foi implementada uma aplicação em C++ utilizando o OpenCV que abre um video padrão, viptraffic.mp4, que deve estar armazenada junto com os binários da aplicação, ou o caminho de um segundo video como parâmetro da execução do binário via terminal.

No código fonte as seguintes etapas foram efetuadas para realizar esse requisito:

- Abre um vídeo e o salva em uma classe `cv::VideoCapture`.
- Salva o próximo *frame* do vídeo em um objeto da classe `cv::Mat` até o usuário apertar algum botão do teclado. Caso o vídeo acabe ele é reiniciado.
- Captura os eventos do mouse sobre a imagem aberta com o método `cv::setMouseCallback` e executa a função `mouseHandler` para gerir esses eventos.
- No método `mouseHandler` imprime a posição e os valores do pixel daquele frame do vídeo, semelhante ao **Requisito 1**, porém salva a cor daquele pixel em um vetor de cores do tipo `std::vector` utilizando o método `std::vector::push_back`.
- Executa a função `paintColoursToRed` que caso o vetor de cores não esteja vazio, cria uma máscara para aquela cor naquele determinado *frame* utilizando a função `cv::inRange` e aplica essa máscara pintando os pixels de vermelho com o método `cv::Mat::setTo`.

3.5. Requisito 4

Para o requisito 4 foi implementada uma aplicação em C++ utilizando o OpenCV que abre a câmera padrão do sistema como um *stream* de vídeo.

No código fonte as mesmas etapas do **Requisito 3** foram realizadas.

4. Resultados

4.1. Requisito 1

Para o requisito 1 a imagem padrão (*lena.jpg*) foi utilizada para coletar as posições e cores de diferentes pixels da imagem pressionando diferentes pixels da imagem com o botão direito do mouse. Na Figura 1 é possível ver a execução do programa e cada um dos pontos vermelhos foram pontos que foram pressionados na imagem. No entanto, a adição dos pontos vermelhos foram feitas apenas para destacar a posição dos pixels na imagem e não foi efetuada pela execução do programa.

Para imagens em preto-e-branco o mesmo deve acontecer. Porém, substituindo os valores nos três canais de cores, apenas a intensidade luminosa daquele ponto é visualizada. Conforme a Figura 2.

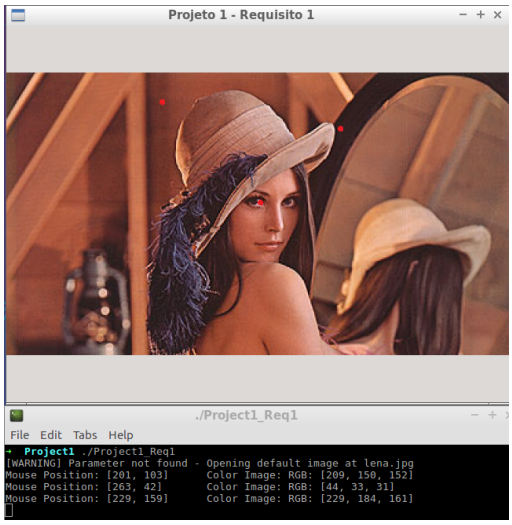

Figura 1. Imagem colorida (lena.jpg) mostrando os resultados do requisito 1. É possível visualizar as cores dos pontos clicados como saídas no terminal.

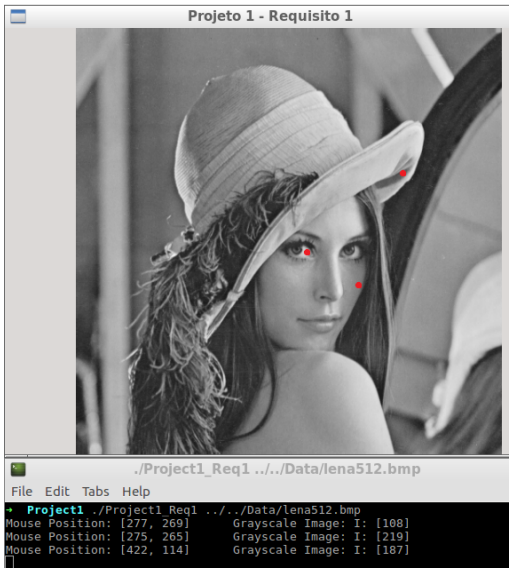

Figura 2. Imagem em escala de cinza (lena512.bmp) mostrando os resultados do requisito 1. É possível visualizar as intensidades luminosa dos pontos clicados como saídas no terminal.

4.2. Requisito 2

Para o requisito 2 a imagem padrão (lena.jpg) foi utilizada para coletar as posições e cores de diferentes pixels da imagem e conforme esses pontos foram pressionados todos os pixels com cores semelhantes ($\pm 5\%$) foram repintadas para a cor vermelho, conforme a Figura 3.

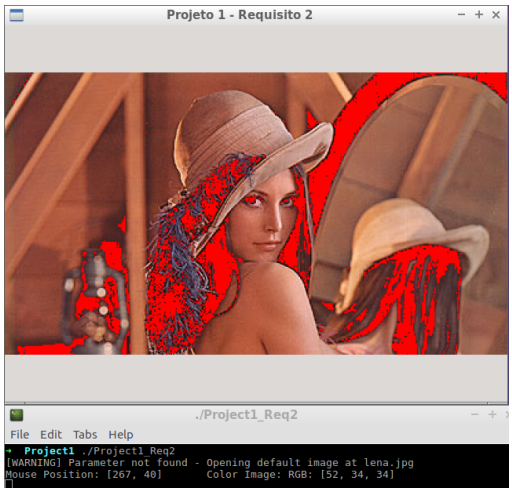

Figura 3. Imagem colorida (lena.bmp) mostrando os resultados do requisito 2. Ao clicar em uma parte escura do cabelo da Lena é possível visualizar que todos os pixels que possuem cores semelhantes foram repintados de vermelho em toda a imagem.

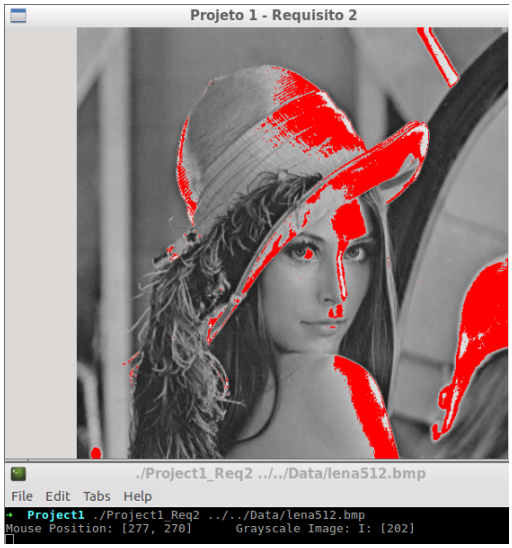

Figura 4. Imagem em escala de cinza (lena512.bmp) mostrando os resultados do requisito 2. É possível visualizar as intensidades luminosa dos pontos clicados como saídas no terminal e também todos os pixels com cores semelhantes foram repintados para vermelho.

Para imagens em nível de cinza o mesmo comportamento acontece, no entanto, em vez de mostrar os valores das cores daquele ponto da imagem é mostrado o valor da intensidade luminosa (Figura 4).

4.3. Requisito 3

Para o requisito 3 o vídeo padrão (viptraffic.mp4) foi utilizado para coletar as cores em pontos de diferentes frames

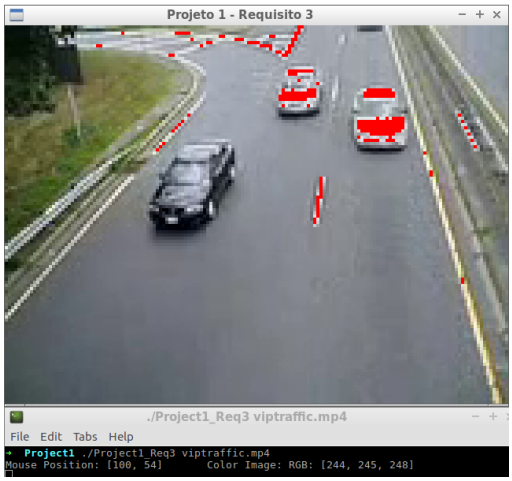

Figura 5. Imagem em escala de cinza (lena512.bmp) mostrando os resultados do requisito 2. É possível visualizar as intensidades luminosa dos pontos clicados como saídas no terminal e também todos os pixels com cores semelhantes foram repintados para vermelho.

da imagem e conforme esses pontos são pressionados todas as cores semelhantes no próximos frames são repintadas para a cor vermelha, conforme 5.

4.4. Requisito 4

O requisito 4 possui a mesma saída que o Requisito 3, no entanto, a fonte do vídeo é a webcam conectada ao computador.

5. Discussão e Conclusões

O projeto demonstrativo concluiu os objetivos propostos e com esse projeto foi possível aprender mais sobre a ferramenta utilizada e sobre conceitos básicos sobre imagens digitais: resolução, espaço de cores, canais de cor e *bit-depth* em imagens.

Com o requisito 3 foi possível perceber que apesar de uma determinada cor for pressionada isso não garante que cores, que, para a percepção humana, são iguais serão repintadas de vermelho. Isso se deve devido as variações de luminosidade pela cena da imagem e também devido à ruídos no processo de captura da imagem. O mesmo efeito acontece ao clicar e

Com os requisitos que utilizam vídeos foi possível também comparar a eficiência de diferentes abordagens na busca de cores de interesse na imagem comparando a queda da taxa de frames do vídeo com diferentes abordagens.

Referências

[1] OpenCV Library official website. `http://opencv.org/`.

[2] D. A. Forsyth and J. Ponce. A modern approach. *Computer vision: a modern approach*, 2003.