Relatório Laboratório 7 - Processamento de Vídeo

1. Introdução

Este relatório documenta os procedimentos e resultados do Laboratório 7 da disciplina de Processamento de Vídeo, cujo objetivo principal foi implementar técnicas para detecção de objetos e rostos, focando principalmente no método de cascata Haar, dispostos na biblioteca do OpenCv.

O experimento focou principalmente na detecção de objetos e rostos primeiramente através de arquivos de imagens previamente já salvos e posteriormente através de imagens da webcam. Desta forma, este documento visa servir como referência técnica para a equipe sobre a programação de manipulação de mídias e câmeras.

2. Fundamentos Básicos

A detecção de objetos por meio de cascatas Haar baseia-se no uso de características visuais simples chamadas Haar-like features, que representam contrastes entre regiões claras e escuras da imagem. Essas características são extraídas rapidamente graças ao uso da imagem integral, uma técnica que permite calcular somas de pixels em regiões retangulares com poucas operações, tornando o método adequado para aplicações em tempo real.

Durante o processo de treinamento, utiliza-se o algoritmo AdaBoost para selecionar as características mais relevantes entre milhares de possibilidades e combinar diversos classificadores fracos em classificadores fortes. Esses classificadores são então organizados em uma estrutura em cascata, onde cada estágio avalia se uma região da imagem pode conter o objeto de interesse. A cascata permite descartar rapidamente áreas improváveis, reduzindo o custo computacional e aumentando a eficiência do sistema. Bibliotecas como o OpenCV implementam essa técnica de forma otimizada, oferecendo modelos pré-treinados e ferramentas para aplicar a detecção de forma simples, normalmente por meio do carregamento de um classificador em formato XML e da utilização da função detectMultiScale.

Apesar de métodos mais modernos baseados em redes neurais terem se popularizado, a cascata Haar permanece uma solução prática, leve e eficaz para tarefas de detecção clássicas, especialmente em sistemas embarcados ou aplicações com restrições de processamento.

3. Materiais e métodos

3.1 Materiais:

3.1.1. Ubuntu (Linux OS)

O que é: Uma distribuição Linux de código aberto.

Finalidade: Fornece o sistema operacional e o ambiente de desenvolvimento onde todas as ferramentas serão instaladas e executadas.

3.1.2. Terminal (Linux Shell)

O que é: Interface de linha de comando (CLI).

Finalidade: Utilizado para executar comandos do sistema, gerenciar ambientes, instalar softwares e rodar programas.

3.1.3. Conda / Miniconda3

O que é: Miniconda é um gerenciador de pacotes e ambientes leve para Python.

Finalidade:

- Gerenciar versões do Python e suas dependências.
- Criar ambientes isolados (como o PV25) para evitar conflitos entre projetos.

3.1.4. OpenCV

O que é: Biblioteca de Visão Computacional de Código Aberto.

Finalidade: Principal biblioteca usada para tarefas de visão computacional e processamento de imagens/vídeos (detecção de objetos, transformações, extração de características etc.).

Variantes:

- OpenCV (compilado do código-fonte): Instalação completa com suporte a C++ e Python.
- OpenCV-Contrib: Módulos extras desenvolvidos pela comunidade (ex.: reconhecimento facial, SIFT, SURF).
- opencv-python (pip): Pacote pré-compilado do OpenCV para Python.
- opencv-contrib-python (pip): Adiciona os módulos extras à instalação em Python.

3.1.5. Ferramentas de Compilação e Dependências

Essenciais para compilar e executar o OpenCV a partir do código-fonte:

- build-essential: Compilador e ferramentas básicas de compilação.
- cmake: Configura e gera os arquivos de build do OpenCV.
- git: Usado para clonar os repositórios do OpenCV.
- pkg-config: Auxilia no gerenciamento de caminhos de bibliotecas e dependências.
- libgtk-3-dev: Suporte de interface gráfica para exibição de imagens.
- libavcodec-dev, libavformat-dev, libswscale-dev: Codecs de vídeo/áudio.
- libjpeg-dev, libpng-dev, libtiff-dev: Suporte a formatos de imagem.
- libxvidcore-dev, libx264-dev: Bibliotecas de codificação de vídeo.
- gfortran, libatlas-base-dev: Aceleração matemática.
- python3-dev, python3-numpy: Cabeçalhos Python + NumPy.
- libtbb-dev: Suporte a multithreading.
- libopenexr-dev: Suporte para imagens HDR.
- libv4l-dev, libdc1394-dev: Suporte a captura de vídeo.

- libgstreamer-dev: Suporte a streaming de vídeo.

3.1.6. pkg-config

O que é: Ferramenta auxiliar para configurar flags de compilação e linkagem de bibliotecas.

Finalidade: Verifica se o OpenCV está instalado e acessível (pkg-config --modversion opencv4).

3.1.7. Python e C++

O que sao: Linguagens de programação usadas para rodar os scripts com OpenCV.

Finalidade: Fornece o ambiente de programação para desenvolver tarefas de visão computacional.

3.1.8. VLC Media Player

O que é: Reprodutor multimídia gratuito e de código aberto.

Finalidade: Permite visualizar vídeos e imagens fora do OpenCV, útil para testes e validação de resultados do processamento de vídeo.

3.2 Procedimento experimental

Ao início de toda prática, estudamos a teoria por trás dos experimentos para a fomentação, desenvolvimento e aplicação das ideias experimentais, com isso, exminamos o repositório disposto em: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html, visando explorar as possibilidades e aplicar a teoria da matéria.

Primeira Parte

Após o estudo, seguimos o roteiro a fim de elaborar um código no qual o programa fazia a leitura de imagens dos membros da equipe e aplicava o filtro Haar cascade para obter a detecção dos respectivos rostos. Para isso, utilizamos modelos pré treinados para a execução do programa, que estão dispostos a seguir: https://github.com/opencv/opencv/tree/master/data/haarcascades. Foram integrados ao código os modelos haarcascade_eye_tree_eyeglasses.xml e haarcascade_frontalface_alt.xml, ambos em formato xml. Os resultados para esta etapa estão dispostos na próxima seção.

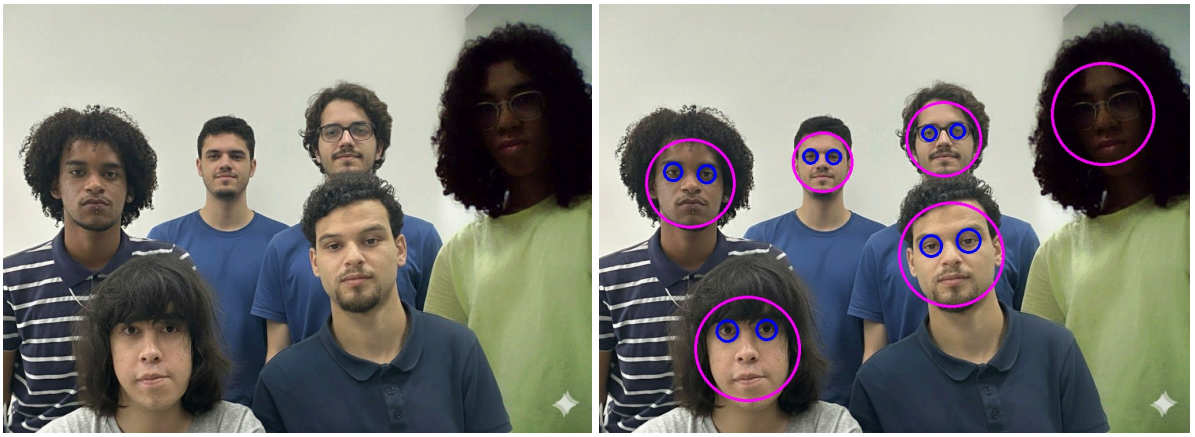
Segunda Parte

Para a segunda parte o roteiro pedia capturas de imagens da webcam para a detecção dos rostos utilizando o Haar cascade. Com isso, modificamos o código utilizado anteriormente para implementação da webcam e aplicação do filtro em tempo real e a função de capturar a imagem ao pressionar uma tecla.

4. Resultados e análises

Para a primeira parte, temos abaixo os resultados obtidos ao rodar o código desenvolvido para a leitura e aplicação do filtro Haar cascade em uma imagem fixa:

Abaixo, temos outro exemplo:



Já para a segunda parte, executamos o programa a fim de capturar uma imagem em tempo real através da webcam e aplicar o filtro instantaneamente. Abaixo, temos o resultado para esta parte: