Laboratório Experimental do SPV (Lex) - Grupo INFOMinds
Integrantes:

Ana Paula Sales	**RA: 11201811710**
Caio Henrique Falcheti Nunes	**RA: 11201920936**
Edson Felipe	**RA: 11201922149**
Gabriel César Napóles	**RA: 11201722756**

Parte 1: Configurações iniciais:
Antes de iniciar a execução do experimento, é necessário se certificar de que o usuário esteja utilizando uma máquina com sistema operacional UBUNTU LTS 22.04. Ademais, é necessário ter, também, o software “OpenCV” instalado e configurado. Caso necessário, é possível seguir os passos deste link para sua instalação: https://moodle.ufabc.edu.br/pluginfile.php/217762/mod_resource/content/1/L1_parte1_instalar_opencv.txt.

Ademais, para a operação do software desenvolvido para estes experimentos, também é necessário a utilização de uma WebCam.
Por fim, todos os códigos e demais arquivos utilizados neste laboratório estão disponíveis no seguinte repositório do GitHub: https://github.com/salesana/processamento-video/blob/main/videoProcessing_comentado.zip

Parte 2: Coleta de dados
O primeiro programa a ser desenvolvido tem como objetivo coletar fotos do rosto dos integrantes a serem reconhecidos pelo software final. Para configurá-lo, baixe o arquivo “takephotos.cpp” presente no repositório mencionado anteriormente ou crie um novo arquivo de texto e cole o seguinte código:

```
#include <opencv2/opencv.hpp>
#include <opencv2/face.hpp>
#include <fstream>
#include <filesystem>
#include <iostream>
#include <ctime>

namespace fs = std::filesystem;
using namespace cv;
using namespace std;

// Realiza o salvamento do par (Id,name) no arquivo Id_names.csv
void saveToCSV(const string &filename, const vector<pair<int, string>> &data) {
        ofstream file(filename);
        if (file.is_open()) {
        file << "id,name\n";
        for (const auto &entry : data) {
        file << entry.first << "," << entry.second << "\n";
        }
        file.close();
        }
}

// Realiza a leitura dos pares (Id,name) a partir do arquivo Id_names.csv
vector<pair<int, string>> readFromCSV(const string &filename) {
        vector<pair<int, string>> data;
        ifstream file(filename);
        if (file.is_open()) {
        string line;
        getline(file, line);
```

```
        while (getline(file, line)) {
        size_t commaPos = line.find(',');
        int id = stoi(line.substr(0, commaPos));
        string name = line.substr(commaPos + 1);
        data.emplace_back(id, name);
        }
        file.close();
        }
        return data;
}

int main() {

        // Especifica onde os arquivos serão salvos
        string csvFile = "./train/id-names.csv";
        string csvFileToRecog = "./train/Recog/Classifiers/id-names.csv";
        string facesPath = "./train/faces";

        vector<pair<int, string>> idNames;

        // Carrega ou inicializa o .csv
        if (fs::exists(csvFile)) {
        idNames = readFromCSV(csvFile);
        } else {
        saveToCSV(csvFile, idNames);
        }

        // Verifica se o caminho do diretório apontado existe
        if (!fs::exists(facesPath)) {
        // Cria a estrutura de diretorios especificada, caso ela não exista
        fs::create_directories(facesPath);
        }

        cout << "Welcome!\n\nPlease put in your ID.\n";
        cout << "If this is your first time, choose a random ID between 1-10000\n";

        int id;
        string name;
        cout << "ID: ";
        cin >> id;

        auto it = find_if(idNames.begin(), idNames.end(),
                [id](const pair<int, string> &entry) { return entry.first == id; });

        //Verifica a partir do ID se o usuário já está cadastrado
        if (it != idNames.end()) {
        name = it->second;
        cout << "Welcome Back! " << name << "!!\n";
        } else {
        cout << "Please Enter your name: ";
        cin.ignore();
        getline(cin, name);

        // Cria o diretorio para o usuario no caminho "/train/faces"
        string personDir = facesPath + "/" + to_string(id);
        fs::create_directory(personDir);

        idNames.emplace_back(id, name);
        saveToCSV(csvFile, idNames);
        saveToCSV(csvFileToRecog, idNames);
        }

        //inicia a etapa de captura. Sugere-se a captura de 10 a 30 fotos
        cout << "\nLet's capture! Press 's' to take a picture, and 'q' to quit.\n";
        // verifica
        VideoCapture camera(0);
        if (!camera.isOpened()) {
        cerr << "Error: Could not open the camera.\n";
        return -1;
        }

        CascadeClassifier faceClassifier("Classifiers/haarface.xml");
        if (faceClassifier.empty()) {
        cerr << "Error: Could not load classifier cascade.\n";
```

```
        return -1;
        }

        int photosTaken = 0;
        Mat frame, gray;

        while (true) {
        //inicia a captura de fotos segundo comando do proprio usuário
        camera >> frame;
        if (frame.empty()) break;

        //Transformação para escala de cinza
        cvtColor(frame, gray, COLOR_BGR2GRAY);
        vector<Rect> faces;
        faceClassifier.detectMultiScale(gray, faces, 1.1, 5);

        for (const auto &face : faces) {
        //Realiza o redimencionamento da foto capturada.
        rectangle(frame, face, Scalar(0, 0, 255), 2);
        Mat faceRegion = gray(face);

        if (waitKey(1) == 's' && mean(faceRegion)[0] > 50) {
        Mat resizedFace;
        resize(faceRegion, resizedFace, Size(220, 220));

        // Salva as fotos no diretório especificado
        string filename = facesPath + "/" + to_string(id) + "/face_" + to_string(photosTaken++) + ".jpg";
        imwrite(filename, resizedFace);
        cout << photosTaken << " -> Photos taken!\n";
        }
        }

        imshow("Face Capture", frame);
        if (waitKey(1) == 'q') break;
        }
        // Encerra o acesso à camera do dispositivo
        camera.release();
        destroyAllWindows();
        return 0;
}
```

Além disso, é necessário configurar um arquivo “CMakeLists.txt” e salvá-lo na mesma pasta em que criou o código anterior. Este arquivo deverá ter o seguinte código:

```
cmake_minimum_required(VERSION 2.8)
project( takephotos )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( takephotos takephotos.cpp )
target_link_libraries( takephotos ${OpenCV_LIBS} )
```

Em seguida, copie, também, para o repositório criado, a pasta “Classifiers”. Esta pasta deverá conter um arquivo XML “haarface.xml” que auxilia o código a reconhecer rostos.

Por fim, acesse, via terminal, o repositório criado e execute os comandos “cmake .” e, em seguida, “make”. Você deverá notar que um arquivo executável de nome “takephotos” foi criado nesta pasta.

Para executá-lo, certifique-se que a WebCam está conectada à máquina e utilize o comando “./takephotos”. Em seguida, informe um número inteiro como ID para o integrante a ser registrado e aperte enter. Na sequência, informe, também, o nome deste integrante e confirme novamente. ATENÇÃO: não utilize o mesmo ID para coletar dados de participantes diferentes, esta ação sobrescreverá as informações previamente salvas.

Você notará que uma janela exibindo a imagem coletada pela WebCam em tempo real. Certifique-se que o integrante a ser cadastrado está bem centralizado e não aparecem mais rostos na imagem. Aperte a tecla “s” para tirar uma foto e coletar os dados das imagens. Recomendamos que tire em torno de 10 a 30 imagens por participante. Após tirar as fotos, pressione a tecla “q” para encerrar o programa. Repita essa operação para cada integrante do grupo.

Após coletar as imagens de todos os integrantes, você deverá notar que foi criada uma pasta “faces” em que todas as fotos foram salvas. Além disso, um arquivo “id-names.csv” também foi gerado. Este arquivo será utilizado nos próximos códigos, então não o altere.

QUESTÕES:
1 - O que acontece quando o rosto do participante está muito “virado” ou “inclinado” em relação à imagem coletada pela WebCam? Por que isso acontece?
2 - Por que as imagens salvas tem resolução menor do que a imagem coletada pela WebCam?
3 - Ainda sobre as imagens coletadas, por que todas foram salvas em “preto e branco”?

Parte 3:Treinamento do modelo
Antes de iniciar o programa de reconhecimento facial, é necessário criar um modelo de “treino” das faces coletadas. Para isso, será necessário criar e executar o programa a seguir. Para isso, baixe o arquivo “train.cpp” do repositório do github ou cole em um arquivo de texto o código abaixo. Não se esqueça de salvar com a extensão de arquivo “.cpp”.

```
#include <iostream>
#include <filesystem>
#include <opencv2/opencv.hpp>
#include <opencv2/face.hpp>

namespace fs = std::filesystem;

std::vector<int> labels;
std::vector<cv::Mat> faces;

// Realiza o carregamento dos dados de treino
void createTrain(const std::string& datasetPath) {
	for (const auto& idDir : fs::directory_iterator(datasetPath)) {
	if (!fs::is_directory(idDir)) continue;

	std::string id = idDir.path().filename().string();
	for (const auto& imgFile : fs::directory_iterator(idDir.path())) {
	try {
	// Acessa a imagem e aplica escala de cinza
	cv::Mat face = cv::imread(imgFile.path().string());
	cv::cvtColor(face, face, cv::COLOR_BGR2GRAY);

	// Adiona os dados TRATADOS ao vetor de treino
	faces.push_back(face);
	labels.push_back(std::stoi(id));
	} catch (...) {
	std::cerr << "Error processing: " << imgFile.path() << std::endl;
	}
	}
	}
}

int main() {
```

```
        // instancia a classe LBPHFaceRecognizer em um objeto lbph
        cv::Ptr<cv::face::LBPHFaceRecognizer> lbph = cv::face::LBPHFaceRecognizer::create();
        lbph->setThreshold(50.0); // Set threshold

        // Realiza o carregamento dos dados de treinamento
        std::cout << "Loading training data..." << std::endl;
        createTrain("faces");

        // Verifica a existência dos dados de treinamento no diretorio /train/faces
        if (faces.empty() || labels.empty()) {
        std::cerr << "No training data found!" << std::endl;
        return -1;
        }

        std::cout << "Training Started" << std::endl;
        lbph->train(faces, labels);

        // Salva o modelo treinado no arquivo e caminho especificado
        std::string modelPath = "./Recog/TrainedLBPH.yml";
        std::string modelPathAlt = "./Recog/Classifiers/TrainedLBPH.yml";
        lbph->save(modelPath);
        lbph->save(modelPathAlt);

        std::cout << "Training Complete!" << std::endl;

        return 0;
}
```

Além deste, também é necessário criar, novamente, um arquivo de texto “CMakeLists.txt”, conforme o código abaixo:

```
cmake_minimum_required(VERSION 2.8)
project( Train )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( Train Train.cpp )
target_link_libraries( Train ${OpenCV_LIBS} )
```

Após isso, abra um novo terminal com acesso à esta pasta e execute os comandos “cmake .” e “make” em sequência. Você notará que um novo arquivo executável de nome “Train” foi criado. Para executá-lo execute no mesmo terminal o comando “./Train”.

Quando finalizado, você notará que na pasta classifiers foi gerado um novo arquivo chamado “TrainedLBPH.yml”. Este arquivo de treino também será utilizado para execução do código final de reconhecimento.

QUESTÃO:
1 - O que acontece ao alterarmos a variável referente ao “Threshold”? (Linha 36 do código “Train.cpp”).

```
lbph->setThreshold(50.0);
```

Parte 4: Reconhecimento Facial
Agora que já temos o novo modelo treinado , cole o código “recognize_face.cpp” disponível no repositório do GitHub ou, em um arquivo de texto, salve como “.cpp” os seguintes comandos:

```
#include <opencv2/opencv.hpp>
```

```
#include <opencv2/face.hpp>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/objdetect.hpp>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <sstream>

//Carrega ID e nomes a partir do arquivo csv
std::map<int, std::string> loadIdNames(con.st std::string& csvFilePath) {
	std::map<int, std::string> idNames;
	std::ifstream file(csvFilePath);
	if (!file.is_open()) {
	std::cerr << "Error opening file: " << csvFilePath << std::endl;
	return idNames;
	}

	std::string line;
	while (std::getline(file, line)) {
	std::stringstream ss(line);
	std::string idStr, name;
	if (std::getline(ss, idStr, ',') && std::getline(ss, name)) {
	try {
	int id = std::stoi(idStr);
	if (!name.empty()) {
		idNames[id] = name;
	} else {
		std::cerr << "Empty name for ID: " << id << std::endl;
	}
	} catch (const std::invalid_argument&) {
	std::cerr << "Invalid ID: " << idStr << " in line: " << line << std::endl;
	} catch (const std::out_of_range&) {
	std::cerr << "ID out of range: " << idStr << " in line: " << line << std::endl;
	}
	} else {
	std::cerr << "Malformed line: " << line << std::endl;
	}
	}
	file.close();
	return idNames;
}

int main(int argc, char* argv[]) {
	// Valida os argumentos fornecidos na chamada da classe
	if (argc < 4) {
	std::cerr << "Usage: " << argv[0]
	<< " <id-names.csv> <haar-cascade-path> <trained-model-path>" << std::endl;
	return -1;
	}

	std::string csvPath = argv[1];
	std::string haarCascadePath = argv[2];
	std::string trainedModelPath = argv[3];

	// Load ID names from CSV
	std::map<int, std::string> idNames = loadIdNames(csvPath);
	if (idNames.empty()) {
	std::cerr << "No valid ID-name pairs loaded from CSV." << std::endl;
	return -1;
	}

	// Carrega o classificador Haar
```

```
cv::CascadeClassifier faceClassifier(haarCascadePath);
if (faceClassifier.empty()) {
std::cerr << "Error loading Haar Cascade Classifier from " << haarCascadePath << std::endl;
return -1;
}

// Carrega o modelo treinado (arquivo .yml)
cv::Ptr<cv::face::LBPHFaceRecognizer> lbph = cv::face::LBPHFaceRecognizer::create();
lbph->setThreshold(500.0);
try {
lbph->read(trainedModelPath);
} catch (const cv::Exception& e) {
std::cerr << "Error loading trained model: " << e.what() << std::endl;
return -1;
}

// Acessa e executa a camera do dispositvo
cv::VideoCapture camera(0);
if (!camera.isOpened()) {
std::cerr << "Error opening camera." << std::endl;
return -1;
}

std::cout << "Press 'q' to quit." << std::endl;

cv::Mat img, grey;
while (cv::waitKey(1) != 'q') {
camera >> img;
if (img.empty()) continue;

//Conversão do frame para escala de cinza
cv::cvtColor(img, grey, cv::COLOR_BGR2GRAY);

// DETECÇÃO DE FACES
std::vector<cv::Rect> faces;
faceClassifier.detectMultiScale(grey, faces, 1.1, 4);

for (const auto& face : faces) {
cv::Mat faceRegion = grey(face);
cv::resize(faceRegion, faceRegion, cv::Size(220, 220));

int label = -1;
double trust = 0.0;
lbph->predict(faceRegion, label, trust);

auto it = idNames.find(label);
if (it != idNames.end() && trust > 0) {
const std::string& name = it->second;
//  DESENHA RETANGULO COM IDENTIFICADOR (NOME)
cv::rectangle(img, face, cv::Scalar(0, 255, 0), 2);
cv::putText(img, name, cv::Point(face.x, face.y - 10),
        cv::FONT_HERSHEY_COMPLEX, 0.8, cv::Scalar(0, 255, 0), 2);
} else {
cv::rectangle(img, face, cv::Scalar(0, 0, 255), 2);
cv::putText(img, "Unknown", cv::Point(face.x, face.y - 10),
        cv::FONT_HERSHEY_COMPLEX, 0.8, cv::Scalar(0, 0, 255), 2);
}
}

// Apresenta
cv::imshow("Recognize", img);
}

// Release the camera and destroy windows
camera.release();
cv::destroyAllWindows();
```

```
	return 0;
}
```

Como feito nas partes anteriores, crie também um arquivo “CMakeLists.txt” com os comandos:

```
cmake_minimum_required(VERSION 2.8)
project( recognize_face )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( recognize_face recognize_face.cpp )
target_link_libraries( recognize_face ${OpenCV_LIBS} )
```

No mesmo diretório criado, abra um novo terminal e execute, novamente, os comandos de “cmake .” e “make” para gerar o arquivo executável.

Por fim, para executar o programa final, no mesmo terminal, cole o comando abaixo:

```
./recognize_face ./Classifiers/id-names.csv ./Classifiers/haarface.xml ./Classifiers/TrainedLBPH.yml
```

Ao ser executado, você notará que uma nova janela com a imagem da WebCam será aberta. Quando um integrante previamente cadastrado entrar em seu campo de visão, você deverá notar que uma “caixa verde” será sobreposta ao seu rosto, exibindo, também, seu nome, conforme as imagens abaixo.

QUESTÕES:

1 - O que acontece se um integrante previamente não cadastrado tentar ser reconhecido?

2 - O que acontece se mais de uma pessoa tentar ser reconhecida, simultaneamente?

3 - O que acontece se parte do rosto estiver coberto? Por exemplo, o nariz e a boca?