МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«Высокоуровневая работа с периферийными устройствами»

Студента 2 курса, 21211 группы

Петрова Сергея Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Антон Юрьевич Кудинов

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ

- *Ознакомиться с программированием периферийных устройств на примере ввода данных с Web-камеры с использованием библиотеки OpenCV;*

ЗАДАНИЕ

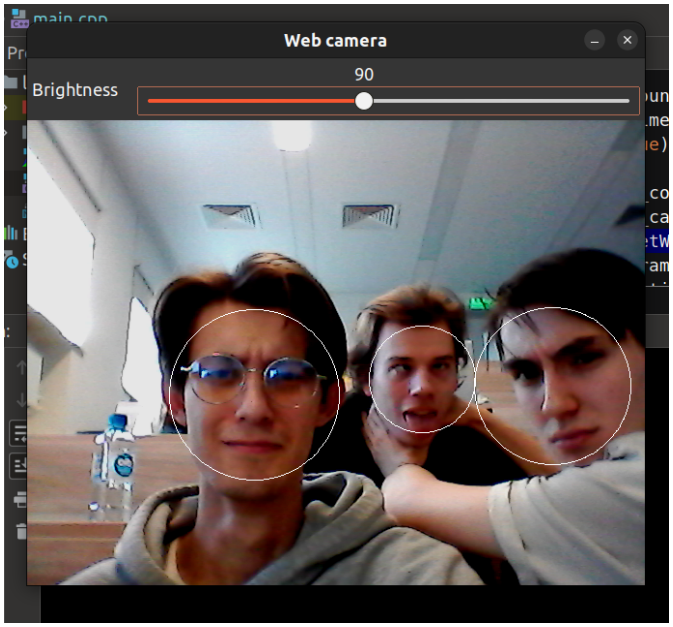
1. *Реализовать программу с использованием OpenCV, которая получает поток видеоданных с камеры и выводит его на экран.*
2. *Выполнить произвольное преобразование изображения.*
3. *Измерить количество кадров, обрабатываемое программой в секунду. Оценить долю времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных, получаемых с камеры.*
4. *Составить отчет по лабораторной работе. Отчет должен содержать следующее:*
    - *Титульный лист.*
    - *Цель лабораторной работы.*
    - *Полный компилируемый листинг реализованной программы и команды для ее компиляции.*
    - *Оценку скорости обработки видео (кадров в секунду) и долю времени, затрачиваемого процессором на обработку (ввод, показ) видеоданных.*
    - *Вывод по результатам лабораторной работы.*

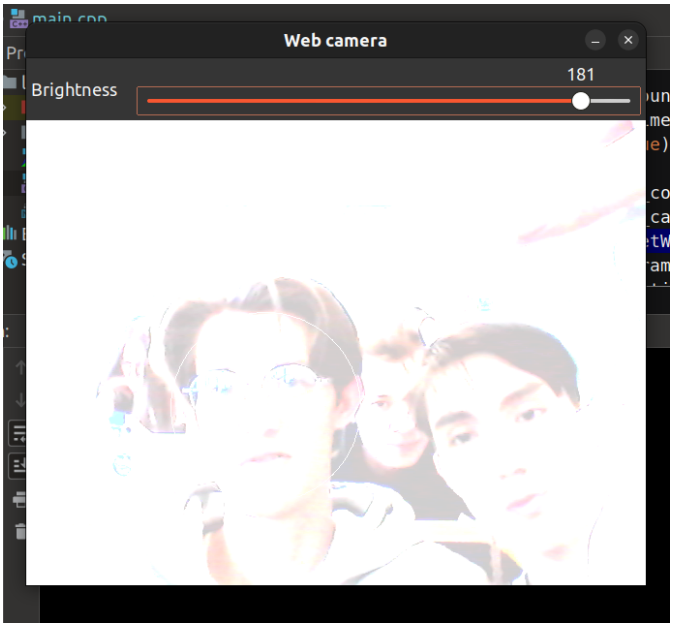
ОПИСАНИЕ РАБОТЫ

Пошаговое описание выполненной работы

1. *Реализовал программу с использованием OpenCV, которая получает поток видеоданных с Web-камеры и выводит его на экран;*
2. *Добавил обнаружение лиц (для этого добавил классификатор Хаара, используемый для обнаружения человеческих лиц, config/haarcascade_frontalface_alt2.xml из библиотеки OpenCV) и ползунок регулировки яркости изображения;*
3. *Измерил количество кадров, обрабатываемых программой, в секунду;*
4. *Оценил долю времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных, получаемых с камеры.*

Результат измерений






```
FPS: 11.5632
Input time: 1.2126%
Process time: 98.3788%
Output time: 0.408594%
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы:

- *Ознакомился с программированием периферийных устройств на примере ввода данных с Web-камеры с использованием библиотеки OpenCV;*
- *Реализовать программу с использованием OpenCV, которая получает поток видеоданных с камеры и выводит его на экран с функцией обнаружения лиц и ползунком регулировки яркости изображения;*
- *Измерил количество кадров, обрабатываемое программой в секунду.*
- *Оценил долю времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных, получаемых с камеры.*

ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ НА C++)

src/main.cpp

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

#define ESC 27

using namespace std;
using namespace cv;

string xml_path =
"/home/acer/NSU_Computer_And_Peripherals/lab5/config/haarcascade_frontalfac
e_alt2.xml";
string window_name = "Web camera";
string trackbar_name = "Brightness";
Mat frame;

// detect and highlight faces
void DetectAndHighlightFaces(CascadeClassifier face_cascade);

// change brightness
void ChangeBrightness();

// print information about fps and times
void PrintInfo(int fps_counter);

// class for measuring time
class Clock
{
public:
    void Start()
    {
        clock_gettime(CLOCK_MONOTONIC_RAW, &start_);
    }

    void Finish()
    {
        clock_gettime(CLOCK_MONOTONIC_RAW, &finish_);
        total_time_ = (double)finish_.tv_sec - (double)start_.tv_sec +
            1e-9 * ((double)finish_.tv_nsec - (double)start_.tv_nsec);
    }

    double GetTotalTime() const
    {
        return total_time_;
    }
private:
    timespec start_ = { 0, 0 };
    timespec finish_ = { 0, 0 };
    double total_time_ = 0;
} input_time, process_time, output_time, program_time;

int main()
{
    // Open a capturing device
    VideoCapture video_capture(0);
    if (!video_capture.isOpened())
```

```
{
    cerr << "VideoCapture error" << endl;
    return EXIT_FAILURE;
}

// Load face cascade
CascadeClassifier face_cascade(xml_path);
if (face_cascade.empty())
{
    cerr << "CascadeClassifier error" << endl;
    return EXIT_FAILURE;
}

// create window
namedWindow(window_name);

// create trackbar
createTrackbar(trackbar_name,
               window_name,
               nullptr,
               200);
setTrackbarPos(trackbar_name,
               window_name,
               100);

int fps_counter = 0;
program_time.Start();
while (true)
{
    ++fps_counter;
    input_time.Start();

    // get frame
    video_capture >> frame;

    // check that frame is empty and window is closed
    if (getWindowProperty(window_name, WND_PROP_AUTOSIZE) == -1 ||
        frame.empty()) break;

    input_time.Finish();

    process_time.Start();

    // mirror the frame horizontally
    flip(frame,
         frame,
         1);

    ChangeBrightness();

    DetectAndHighlightFaces(face_cascade);
    process_time.Finish();

    output_time.Start();
    // display frame in window
    imshow(window_name,
           frame);
    output_time.Finish();
```

```
        // wait for pressed key ESC
        if (waitKey(1) == ESC) break;
    }
    program_time.Finish();

    PrintInfo(fps_counter);

    return 0;
}

void DetectAndHighlightFaces(CascadeClassifier face_cascade)
{
    // detect faces in frame
    std::vector<Rect> faces;
    face_cascade.detectMultiScale(frame,
                                  faces);

    // draw ellipse around faces
    for (auto & face : faces)
    {
        Point center(int(face.x + face.width * 0.5),
                     int(face.y + face.height * 0.5));
        ellipse(frame,
                center,
                Size(int(face.width * 0.5), int(face.height * 0.5)),
                0,
                0,
                360,
                Scalar(255, 255, 255));
    }
}

void ChangeBrightness()
{
    // get brightness value from trackbar
    int brightness = getTrackbarPos(trackbar_name,
                                    window_name);

    // change frame brightness
    frame.convertTo(frame,
                    -1,
                    1,
                    double(brightness - 100) * 255 / 100);
}

void PrintInfo(int fps_counter)
{
    double time = input_time.GetTotalTime() + process_time.GetTotalTime() +
output_time.GetTotalTime();
    cout << "FPS: " << fps_counter / program_time.GetTotalTime() << endl;
    cout << "Input time: " << 100.0 * input_time.GetTotalTime() / time <<
"%" << endl;
    cout << "Process time: " << 100.0 * process_time.GetTotalTime() / time
<< "%" << endl;
    cout << "Output time: " << 100.0 * output_time.GetTotalTime() / time <<
"%" << endl;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16.3)

project(lab5 CXX)

find_package(OpenCV REQUIRED)

add_executable(lab5 src/main.cpp)
target_include_directories(lab5 PUBLIC ${OpenCV_INCLUDE_DIRS})
target_link_libraries(lab5 PUBLIC ${OpenCV_LIBS})
```