

**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

<i>СОДЕРЖАНИЕ</i>	2
<i>ЦЕЛЬ</i>	3
<i>ЗАДАНИЕ</i>	3
<i>ОПИСАНИЕ РАБОТЫ</i>	4
<i>Пошаговое описание выполненной работы</i>	4
<i>Команды для компиляции</i>	4
<i>Результат измерений</i>	4
<i>ЗАКЛЮЧЕНИЕ</i>	6
<i>ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ)</i>	7
<i>src/main.cpp</i>	7
<i>CMakeLists.txt</i>	9

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 ввод, обработку и показ видеоданных.*
 - *Вывод по результатам лабораторной работы.*

ОПИСАНИЕ РАБОТЫ

Пошаговое описание выполненной работы

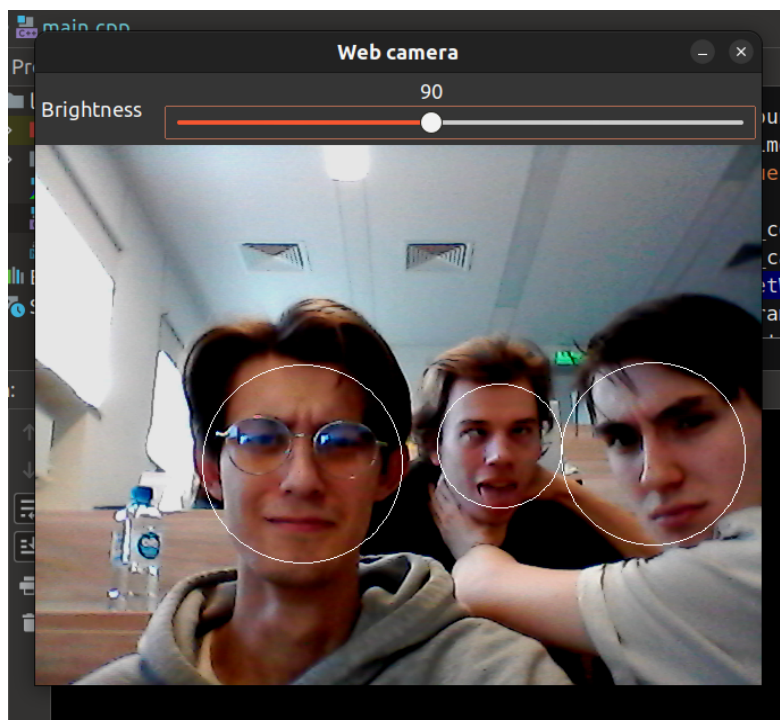
1. Реализовал программу с использованием *OpenCV*, которая получает поток видеоданных с *Web*-камеры и выводит его на экран;
2. Добавил обнаружение лиц (для этого добавил классификатор Хаара, используемый для обнаружения человеческих лиц, *config/haarcascade_frontalface_alt2.xml* из библиотеки *OpenCV*)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

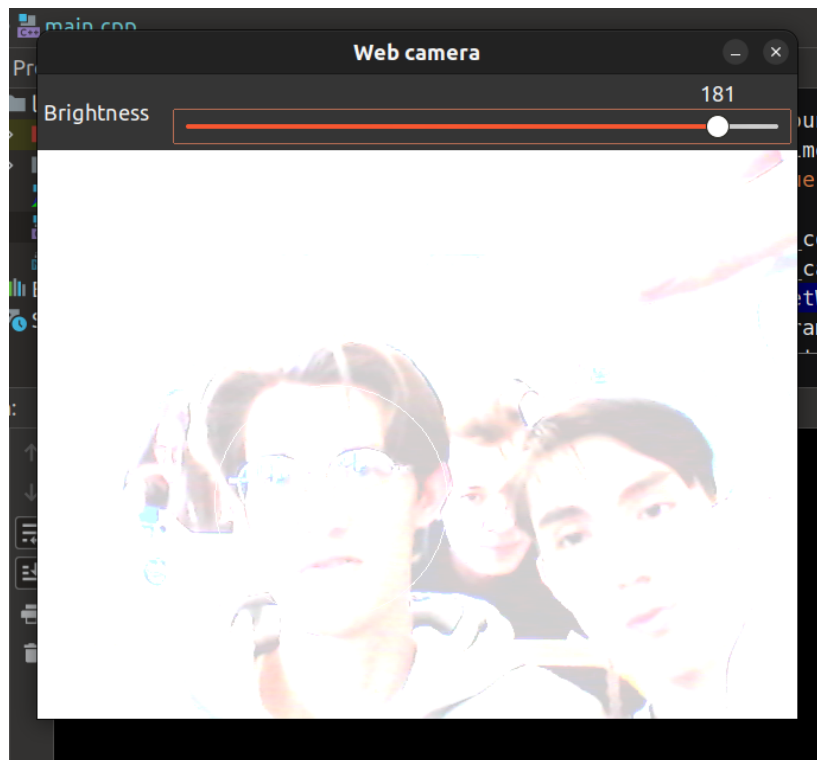
Команды для компиляции

```
evmpu@comrade:~$ cmake
-DCMAKE_BUILD_TYPE=Release
-DCMAKE_CXX_COMPILER=/usr/bin/g++
-G Ninja
-S /home/acer/NSU_Computer_And_Peripherals/lab5
-B /home/acer/NSU_Computer_And_Peripherals/lab5/bin

evmpu@comrade:~$ cmake
--build /home/acer/NSU_Computer_And_Peripherals/lab5/bin
--target lab5
```

Результат измерений





```
/home/acer/NSU_Computer_And_Peripherals/lab5/cmake-build-release/lab5  
Input time: 33.8874%  
Process time: 65.7856%  
Output time: 0.327006%
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы:

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

ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ)

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_frontalface_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
void PrintFPS();

// print information about times
void PrintTimes();

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

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

    double GetIntervalTime() const
    {
        return interval_time_;
    }
}
```

```

        double GetTotalTime() const
        {
            return total_time_;
        }
private:
    timespec start_ = { 0, 0 };
    timespec finish_ = { 0, 0 };
    double interval_time_ = 0.0;
    double total_time_ = 0.0;
} input_time, process_time, output_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);

    while (true)
    {
        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();

        PrintFPS();

        // wait for pressed key ESC
        if (waitKey(1) == ESC) break;
    }

    PrintTimes();

    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),
                3);
    }
}

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 PrintFPS()
{
    double time = input_time.GetIntervalTime() +
                  process_time.GetIntervalTime() +
                  output_time.GetIntervalTime();
    cout << "FPS: " << 1.0 / time << endl;
}

void PrintTimes()
{
    double time = input_time.GetTotalTime() +
                  process_time.GetTotalTime() +
                  output_time.GetTotalTime();
    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})

```