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, 24203 группы

Анисимова Льва Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
доцент, кандидат технических наук
А. Ю. Власенко

Новосибирск 2025

СОДЕРЖАНИЕ

ЦЕЛЬ

Создать интерактивное приложение для обработки видеопотока с веб-камеры в реальном времени с применением различных фильтров и измерением производительности, а также приложение для сканирования и вывода информации об USB-устройствах, подключённых к компьютеру.

ЗАДАНИЕ

Задание 1:

Реализовать приложение, которое позволяет:

- Захватывать видео с веб-камеры
- Применять фильтры к кадрам
- Измерять производительность на каждом этапе обработки
- Отображать результаты в реальном времени

Задание 2:

Реализовать программу, которая:

- Инициализирует libusb библиотеку
- Сканирует все подключённые USB-устройства
- Для каждого устройства выводит:
  - Класс устройства
  - Vendor ID (VID)
  - Product ID (PID)
  - Серийный номер
- Освобождает ресурсы после работы

Составить отчет по лабораторной работе.

ОПИСАНИЕ РАБОТЫ

Задание 1

Архитектура системы захвата и обработки видеопотока

Приложение реализует конвейерную архитектуру обработки видеопотока, состоящую из трёх последовательных этапов: захват, обработка и отображение. Объект `cv::VideoCapture` инициализируется с параметром 0, что обозначает захват видео с основной веб-камеры системы. На этапе инициализации устанавливается разрешение видеопотока 1920×1200 пикселей. Окно отображения создаётся с использованием `cv::namedWindow()` и привязывается обработчик мыши через `cv::setMouseCallback()`, который обеспечивает интерактивное управление выбором фильтра через события `cv::EVENT_LBUTTONDOWN` и `cv::EVENT_RBUTTONDOWN`.

На этапе захвата видеопотока используется оператор перегрузки `cap>>frame`, который вызывает метод `VideoCapture::operator>>()` и асинхронно читает очередной кадр из буфера видеоустройства в объект `cv::Mat frame`. Если кадр успешно прочитан, возвращается ненулевое значение; если буфер пуст или достигнут конец видеофайла, функция возвращает нулевое значение, что проверяется условием `if (frame.empty())`. Захватанный кадр затем подвергается геометрическому преобразованию с использованием `cv::flip(frame, flipped, 1)`, где третий параметр 1 указывает на горизонтальное отражение, которое необходимо для визуально корректного отображения веб-камеры. Если пиксельные размеры кадра отличаются от установленного разрешения 1920×1200, применяется функция `cv::resize(flipped, flipped, cv::Size(1920, 1200))`, которая использует билинейную интерполяцию по умолчанию для масштабирования пиксельных данных.

Измерение производительности конвейера обработки

Система производительности построена на использовании высокоточных таймеров `std::chrono::high_resolution_clock`, которые позволяют измерять интервалы времени с микросекундной точностью. Для каждого этапа конвейера создаются две контрольные точки: `t_input_start` и `t_input_end` для измерения времени захвата, `t_proc_start` и `t_proc_end` для времени обработки кадра, и `t_display_start` и `t_display_end` для времени отображения. Длительность каждого этапа вычисляется через функцию `std::chrono::duration<double, std::milli>()`, которая вычисляет разницу между контрольными точками в миллисекундах. Для получения процентного соотношения нагрузки вычисляется общее время обработки как сумма `inputTime + processTime`, а затем каждый компонент делится на общее время с умножением на 100: `inputPercent = (inputTime / totalTime) * 100`. Это позволяет визуально оценить, какой этап конвейера является узким местом в системе обработки.

Система фильтров и их реализация

Фильтры реализованы в функции `applyFilter(const cv::Mat& frame, int type)`, которая использует переключатель `switch` для выбора одного из трёх режимов обработки. Фильтр типа 0 (Original) просто возвращает клонированный кадр без изменений. Фильтр типа 1 (Edges) применяет цепочку трансформаций: сначала цветовое пространство преобразуется из BGR в grayscale через `cv::cvtColor(result, result, cv::COLOR_BGR2GRAY)`, затем применяется оператор границ Canny через `cv::Canny(result, result, 100, 200)` с пороговыми значениями 100 и 200, которые определяют чувствительность обнаружения границ, и наконец результат преобразуется обратно в трёхканальное BGR пространство для единообразного отображения. Фильтр типа 2 (Sepia) реализует более сложную пиксельную обработку, используя двойной цикл по координатам (y, x) кадра и обращение к каждому пикселю через `result.at<cv::Vec3b>(y, x)`. Для каждого пикселя вычисляется яркость в grayscale через формулу `gray = (int)(0.2126 * r + 0.7152 * g + 0.0722 * b)`, а затем применяется синусоидальная модуляция множителей цвета: `redMult = 1.2f + 0.5f * sin(frameCount * 0.1f), greenMult = 1.0f + 0.2f * sin(frameCount * 0.1f + 2.09f), blueMult = 0.8f + 0.2f * sin(frameCount * 0.1f + 4.19f)`. Фазовые сдвиги 2.09 и 4.19 радиан обеспечивают несинхронное колебание цветовых каналов, создавая эффект пульсирующей анимации. Функция `cv::saturate_cast<uchar>()` обеспечивает насыщение значений пикселей в диапазон .

Режим сетки и интерактивное управление

В режиме отображения сетки кадр разделяется на три равные колонки с помощью вычисления `width = grid.cols / 3`, где каждая колонка имеет ширину 640 пиксели. Между колонками отрисовываются вертикальные линии белого цвета через функцию `cv::line(grid, cv::Point(i * width, 0), cv::Point(i * width, grid.rows), cv::Scalar(255, 255, 255), 3)` с толщиной 3 пиксела. Для каждой колонки извлекается прямоугольный регион через конструктор `cv::Rect(x1, 0, width, grid.rows)`, который клонируется в отдельный буфер `cell`, обрабатывается соответствующим фильтром, а результат копируется обратно в исходное изображение через функцию `copyTo()`. Интерактивное управление реализуется через обработчик мыши `mouseCallback()`, который вычисляет номер колонки на основе координаты X клика: `int col = x / (currentFrame.cols / 3)`, и устанавливает глобальную переменную `selectedFilter` в значение от 0 до 2. При клике правой кнопкой мыши `selectedFilter` устанавливается в -1, переводя приложение обратно в режим сетки.

Отображение информации и управление циклом

Информация выводится на кадр с использованием функции `cv::putText()`, которая растеризует текст в растровое пиксельное представление и накладывает его на кадр. Параметры функции включают начальную координату `cv::Point(20, 50)`, шрифт `cv::FONT_HERSHEY_SIMPLEX`, масштаб 1.2, цвет в формате BGR `cv::Scalar(0, 255, 0)` (зелёный) и толщину линии 2. Временное разрешение отображения FPS вычисляется один раз в секунду путём сравнения текущего системного времени `time(nullptr)` с сохранённым `prevTime` и обнуления счётчика кадров `frameCount`. Основной цикл обработки управляется функцией `cv::waitKey(1)`, которая блокирует выполнение программы на 1 миллисекунду и возвращает код клавиши, если она была нажата; проверка `if ((char)cv::waitKey(1) == 27)` позволяет выйти из цикла при нажатии клавиши ESC (код 27). После завершения цикла ресурсы освобождаются через `cap.release()` и `cv::destroyAllWindows()`.

Задание 2

Инициализация и управление контекстом libusb

Библиотека libusb работает на основе контекста, который управляет всеми операциями работы с USB-устройствами. Контекст инициализируется через функцию `libusb_init(libusb_context **ctx)`, которая выделяет внутренние структуры данных и инициализирует драйверы USB-устройств операционной системы. При успешной инициализации функция возвращает 0, а адрес контекста сохраняется в переменную `ctx`. Контекст содержит информацию обо всех подключённых USB-устройствах, доступных через опрос USB-хаба операционной системы. После завершения работы контекст обязательно освобождается через `libusb_exit(ctx)`, которая закрывает соединение с драйверами и освобождает выделенную память.

Получение и обработка списка USB-устройств

Получение полного списка подключённых USB-устройств выполняется функцией `libusb_get_device_list(libusb_context *ctx, libusb_device ***devs)`, которая возвращает количество найденных устройств в виде целого числа. Указатель `devs` становится массивом указателей на структуры `libusb_device`, каждая из которых представляет физическое USB-устройство в системе. Возвращаемое значение сохраняется в переменную `cnt`, которая затем используется в цикле для итерации по всем устройствам: `for (ssize_t i = 0; i < cnt; i++)`. Каждый элемент массива `devs[i]` содержит опубликованную информацию об устройстве, которая была получена от USB-хаба при подключении устройства. После завершения обработки всех устройств память списка освобождается через `libusb_free_device_list(devs, 1)`, где второй параметр 1 указывает на необходимость уменьшения счётчика ссылок для каждого устройства в списке.

Чтение дескриптора устройства и извлечение базовой информации

Для каждого устройства в списке читается его дескриптор через функцию `libusb_get_device_descriptor(libusb_device *dev, struct libusb_device_descriptor *desc)`, которая заполняет структуру `desc` информацией об устройстве, включая классификацию, идентификаторы производителя и изделия, и индексы строковых дескрипторов. Структура `libusb_device_descriptor` содержит поле `bDeviceClass`, которое кодирует тип устройства в один байт: например, 0x09 обозначает USB Hub, 0x00 означает, что класс определён на уровне интерфейса, 0xFF обозначает специфичный для производителя класс. Поля `idVendor` и `idProduct` хранят 16-битные беззнаковые целые числа, которые однозначно идентифицируют производителя и конкретную модель устройства. Эти данные считываются из конфигурационных дескрипторов устройства при его подключении и кэшируются в памяти USB-контроллера, поэтому они доступны без открытия устройства. Форматирование вывода выполняется с использованием манипуляторов потока C++: `hex` переводит числовой вывод в шестнадцатеричный формат, `setfill('0')` заполняет пробелы нулями, `setw(4)` устанавливает ширину поля в 4 символа для VID и PID, `dec` возвращает десятичный формат.

Получение строковых дескрипторов и серийного номера

Для получения текстовой информации об устройстве, такой как серийный номер, необходимо открыть дескриптор устройства через функцию `libusb_open(libusb_device *dev, libusb_device_handle **handle)`, которая устанавливает соединение с USB-устройством и возвращает дескриптор в переменную `handle`. Успешное открытие проверяется условием `r == 0 && handle != NULL`, где `r` содержит код результата (0 означает успех, отрицательное значение означает ошибку). Хотя дескриптор содержит поле `iSerialNumber`, которое хранит индекс строкового дескриптора в диапазоне , сама строка находится в памяти устройства и должна быть получена через функцию `libusb_get_string_descriptor_ascii(libusb_device_handle *dev, uint8_t desc_index, unsigned char *data, int length)`. Эта функция запрашивает у устройства строку с индексом `desc.iSerialNumber` и преобразует её из внутреннего Unicode-представления USB в ASCII-текст, который сохраняется в буфер `data`. После получения всей необходимой информации дескриптор устройства закрывается через `libusb_close(handle)`.

Структура программного потока обработки устройств

Основной поток обработки представляет собой последовательный конвейер: инициализация libusb контекста → получение списка устройств → итерация по каждому устройству → чтение базовых дескрипторов (класс, VID, PID) → вывод информации без открытия устройства → открытие устройства для получения серийного номера → закрытие дескриптора устройства → освобождение памяти списка → завершение libusb контекста. Обработка ошибок выполняется через проверку возвращаемых кодов функций libusb: положительные значения обычно обозначают успех, ноль означает операцию без полезной нагрузки, отрицательные значения содержат коды ошибок, такие как `LIBUSB_ERROR_NO_DEVICE (-4)` при отключении устройства или `LIBUSB_ERROR_PERMISSION (-3)` при недостаточных привилегиях доступа. Такая архитектура позволяет эффективно сканировать все подключённые USB-устройства с минимальной задержкой, так как базовая информация (класс, VID, PID) читается из кэша контроллера без физического обращения к устройству.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторных работ были реализованы два приложения, демонстрирующие работу с различными аспектами взаимодействия с периферийными устройствами: обработку видеопотока с веб-камеры с использованием библиотеки OpenCV и сканирование USB-устройств через библиотеку libusb.

Приложение 1. *Задание 1. Код приложения на C++*

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

int selectedFilter = -1;
int frameCount = 0;
cv::Mat currentFrame;

double inputTime = 0, processTime = 0, displayTime = 0;
double inputPercent = 0, processPercent = 0;

cv::Mat applyFilter(const cv::Mat& frame, int type)
{
    cv::Mat result = frame.clone();

    switch (type)
    {
    case 0: // Original
        return result;

    case 1: // Edges
        {
            cv::cvtColor(result, result, cv::COLOR_BGR2GRAY);
            cv::Canny(result, result, 100, 200);
            cv::cvtColor(result, result, cv::COLOR_GRAY2BGR);
            return result;
        }

    case 2: // Sepia
        {
            for (int y = 0; y < result.rows; y++)
            {
                for (int x = 0; x < result.cols; x++)
                {
                    cv::Vec3b pixel = result.at<cv::Vec3b>(y,
x);
                    int b = pixel[0], g = pixel[1], r =
pixel[2];
                    int gray = (int)(0.2126 * r + 0.7152 * g +
0.0722 * b);

                    // Изменение цвета по синусоиде
                    float t = frameCount * 0.1f;
                    float redMult = 1.2f + 0.5f * sin(t);
                    float greenMult = 1.0f + 0.2f * sin(t +
2.09f);
                    float blueMult = 0.8f + 0.2f * sin(t +
4.19f);

                    result.at<cv::Vec3b>(y, x) = cv::Vec3b(
                        cv::saturate_cast<uchar>(gray *
blueMult),
```

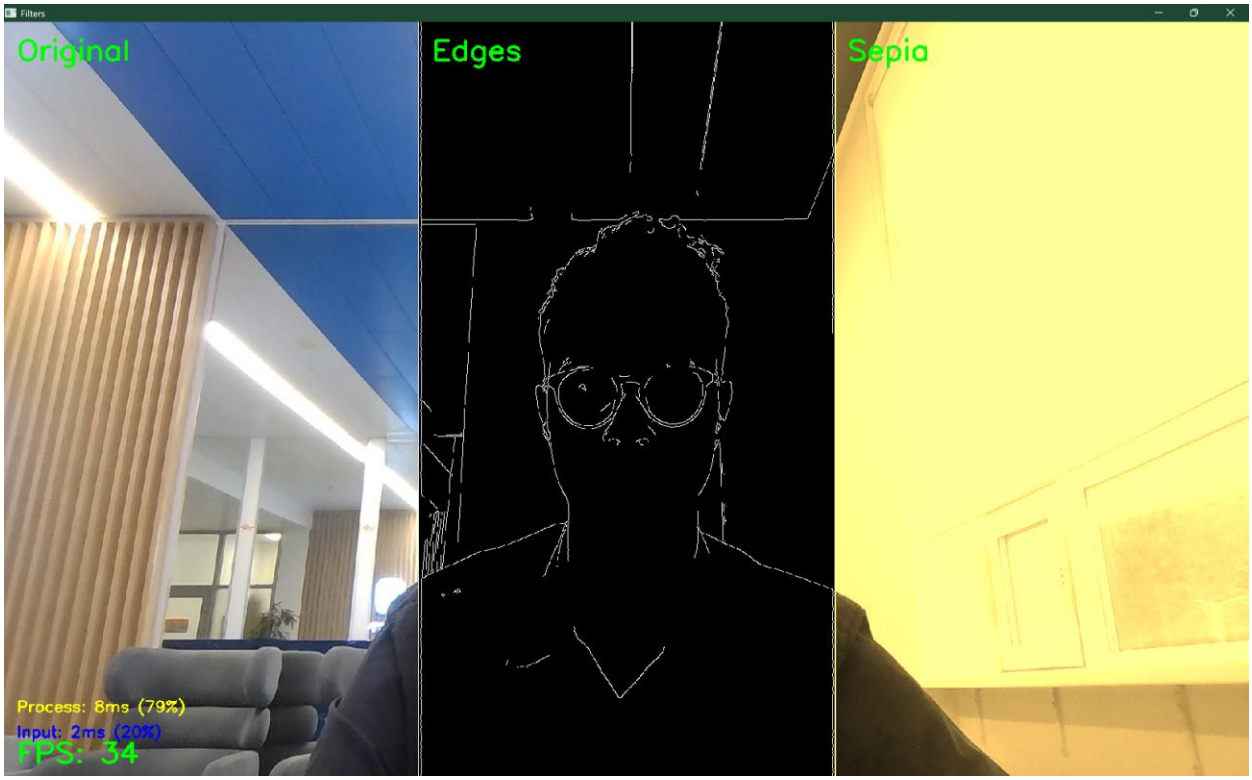
```
                        cv::saturate_cast<uchar>(gray *
greenMult),
                        cv::saturate_cast<uchar>(gray * redMult)
                    );
                }
            }

            frameCount++;
            return result;
        }

    default: return result;
    }
}

void mouseCallback(int event, int x, int, int, void*)
{
    if (event == cv::EVENT_LBUTTONDOWN)
    {
        int col = x / (currentFrame.cols / 3);
        if (col < 3) selectedFilter = col;
    }
    if (event == cv::EVENT_RBUTTONDOWN)
    {
        selectedFilter = -1;
    }
}

int main()
{
    cv::VideoCapture cap(0);
    if (!cap.isOpened())
    {
        return -1;
    }

    cap.set(cv::CAP_PROP_FRAME_WIDTH, 1920);
    cap.set(cv::CAP_PROP_FRAME_HEIGHT, 1200);

    std::vector<std::string> names = {"Original", "Edges",
"Sepia"};

    cv::namedWindow("Filters", cv::WINDOW_NORMAL);
    cv::resizeWindow("Filters", 1920, 1200);
    cv::setMouseCallback("Filters", mouseCallback);

    cv::Mat frame, flipped;

    double fps = 0;
    time_t prevTime = time(nullptr);

    while (true)
    {
```

```
        auto t_input_start =
std::chrono::high_resolution_clock::now();
        cap >> frame;
        auto t_input_end =
std::chrono::high_resolution_clock::now();

        if (frame.empty())
        {
            break;
        }

        auto t_proc_start =
std::chrono::high_resolution_clock::now();

        // Flipping increases time x3.8 times
        cv::flip(frame, flipped, 1);

        // Resize does the same
        if (flipped.cols != 1920 || flipped.rows != 1200)
        {
            cv::resize(flipped, flipped, cv::Size(1920, 1200));
        }

        // Clear memory copying needs much system time
        currentFrame = flipped.clone();
        frameCount++;

        if (time(nullptr) - prevTime >= 1)
        {
            fps = frameCount;
            frameCount = 0;
            prevTime = time(nullptr);
        }

        auto t_proc_end =
std::chrono::high_resolution_clock::now();
        inputTime = std::chrono::duration<double,
std::milli>(t_input_end - t_input_start).count();
        processTime = std::chrono::duration<double,
std::milli>(t_proc_end - t_proc_start).count();
        double totalTime = inputTime + processTime;
        if (totalTime > 0)
        {
            inputPercent = (inputTime / totalTime) * 100;
            processPercent = (processTime / totalTime) * 100;
        }

        if (selectedFilter >= 0 && selectedFilter < 3)
        {
            cv::Mat filtered = applyFilter(flipped,
selectedFilter);
            cv::putText(filtered, "FPS: " +
std::to_string((int)fps),
```

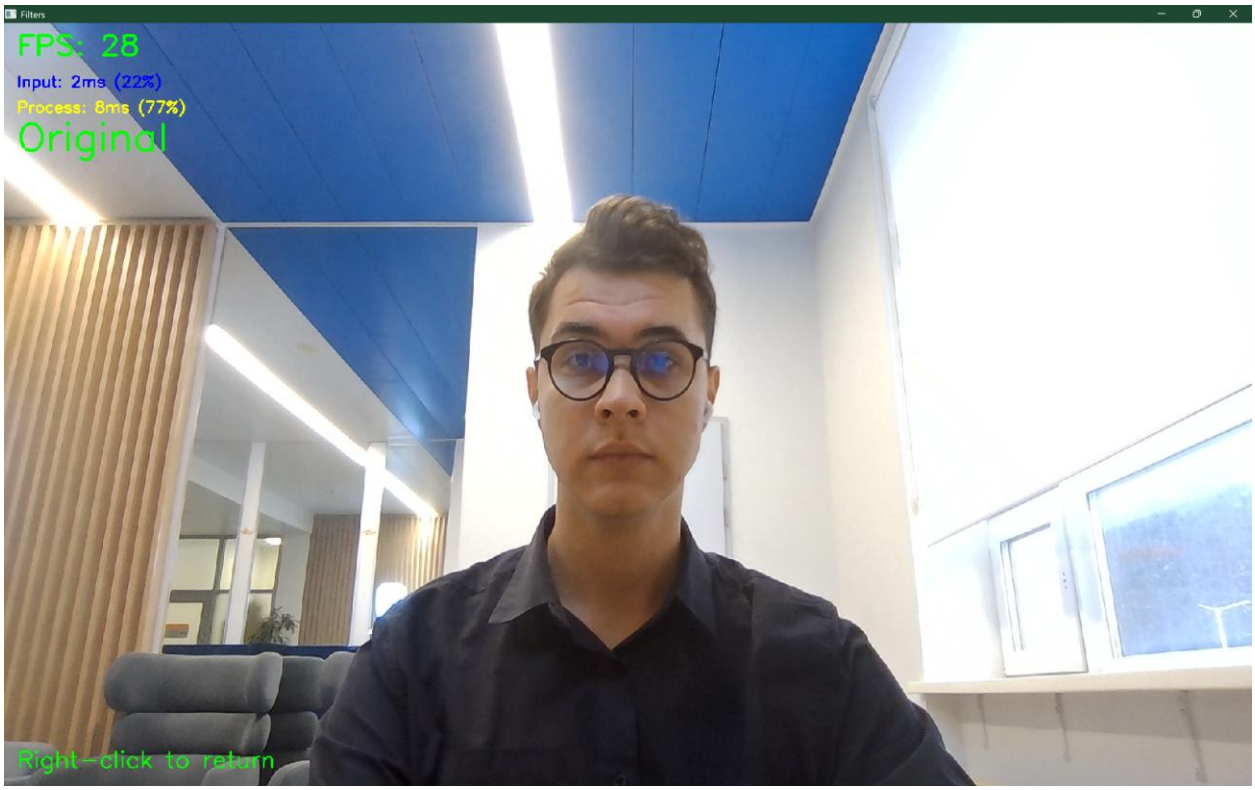
```
                        cv::Point(20, 50),
cv::FONT_HERSHEY_SIMPLEX, 1.5,
                        cv::Scalar(0, 255, 0), 3);
            cv::putText(filtered, "Input: " +
std::to_string((int)inputTime) + "ms (" +
std::to_string((int)inputPercent) + "%)",
                        cv::Point(20, 100),
cv::FONT_HERSHEY_SIMPLEX, 0.8,
                        cv::Scalar(255, 0, 0), 2);
            cv::putText(filtered, "Process: " +
std::to_string((int)processTime) + "ms (" +
std::to_string((int)processPercent) + "%)",
                        cv::Point(20, 140),
cv::FONT_HERSHEY_SIMPLEX, 0.8,
                        cv::Scalar(0, 255, 255), 2);
            cv::putText(filtered, names[selectedFilter],
                        cv::Point(20, 200),
cv::FONT_HERSHEY_SIMPLEX, 2,
                        cv::Scalar(0, 255, 0), 3);
            cv::putText(filtered, "Right-click to return",
                        cv::Point(20, filtered.rows - 30),
                        cv::FONT_HERSHEY_SIMPLEX, 1.2,
cv::Scalar(0, 255, 0), 2);

            auto t_display_start =
std::chrono::high_resolution_clock::now();
            cv::imshow("Filters", filtered);
            auto t_display_end =
std::chrono::high_resolution_clock::now();
            displayTime = std::chrono::duration<double,
std::milli>(t_display_end - t_display_start).count();
        }
        else
        {
            cv::Mat grid = flipped.clone();
            int width = grid.cols / 3;

            // Vertical lines
            for (int i = 1; i < 3; i++)
            {
                cv::line(grid, cv::Point(i * width, 0),
                         cv::Point(i * width, grid.rows),
                         cv::Scalar(255, 255, 255), 3);
            }

            // Apply filters to each column
            for (int i = 0; i < 3; i++)
            {
                int x1 = i * width;
                cv::Mat cell = grid(cv::Rect(x1, 0, width,
grid.rows)).clone();
                applyFilter(cell, i).copyTo(grid(cv::Rect(x1, 0,
width, grid.rows)));
```

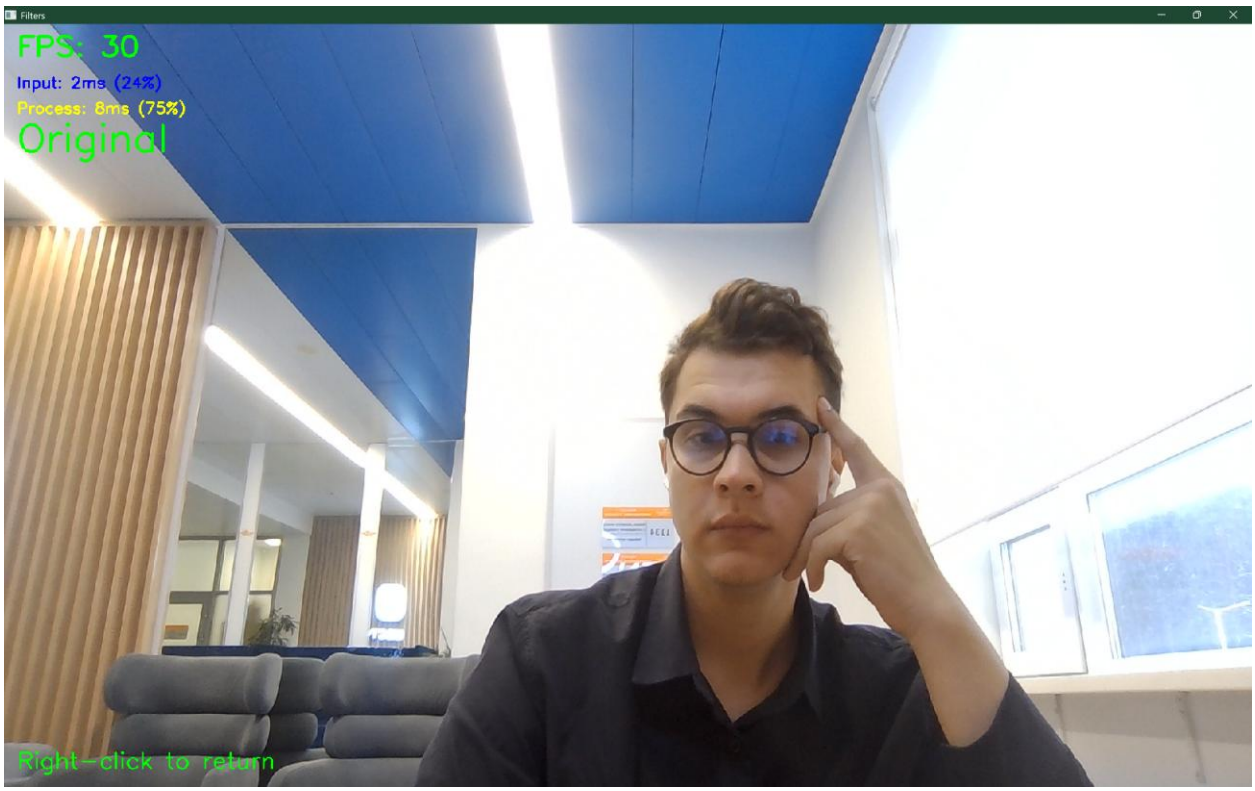
```
                cv::putText(grid, names[i], cv::Point(x1 + 20,
60),
                            cv::FONT_HERSHEY_SIMPLEX, 1.5,
                            cv::Scalar(0, 255, 0), 3);
            }

            cv::putText(grid, "FPS: " +
std::to_string((int)fps),
                        cv::Point(20, grid.rows - 30),
                        cv::FONT_HERSHEY_SIMPLEX, 1.5,
cv::Scalar(0, 255, 0), 3);
            cv::putText(grid, "Input: " +
std::to_string((int)inputTime) + "ms (" +
std::to_string((int)inputPercent) + "%)",
                        cv::Point(20, grid.rows - 70),
                        cv::FONT_HERSHEY_SIMPLEX, 0.8,
cv::Scalar(255, 0, 0), 2);
            cv::putText(grid, "Process: " +
std::to_string((int)processTime) + "ms (" +
std::to_string((int)processPercent) + "%)",
                        cv::Point(20, grid.rows - 110),
                        cv::FONT_HERSHEY_SIMPLEX, 0.8,
cv::Scalar(0, 255, 255), 2);

            auto t_display_start =
std::chrono::high_resolution_clock::now();
            cv::imshow("Filters", grid);
            auto t_display_end =
std::chrono::high_resolution_clock::now();
            displayTime = std::chrono::duration<double,
std::milli>(t_display_end - t_display_start).count();
        }

        if ((char)cv::waitKey(1) == 27) {
            break;
        }
    }

    cap.release();
    cv::destroyAllWindows();
    return 0;
}
```

Приложение 2. *Работа приложения*

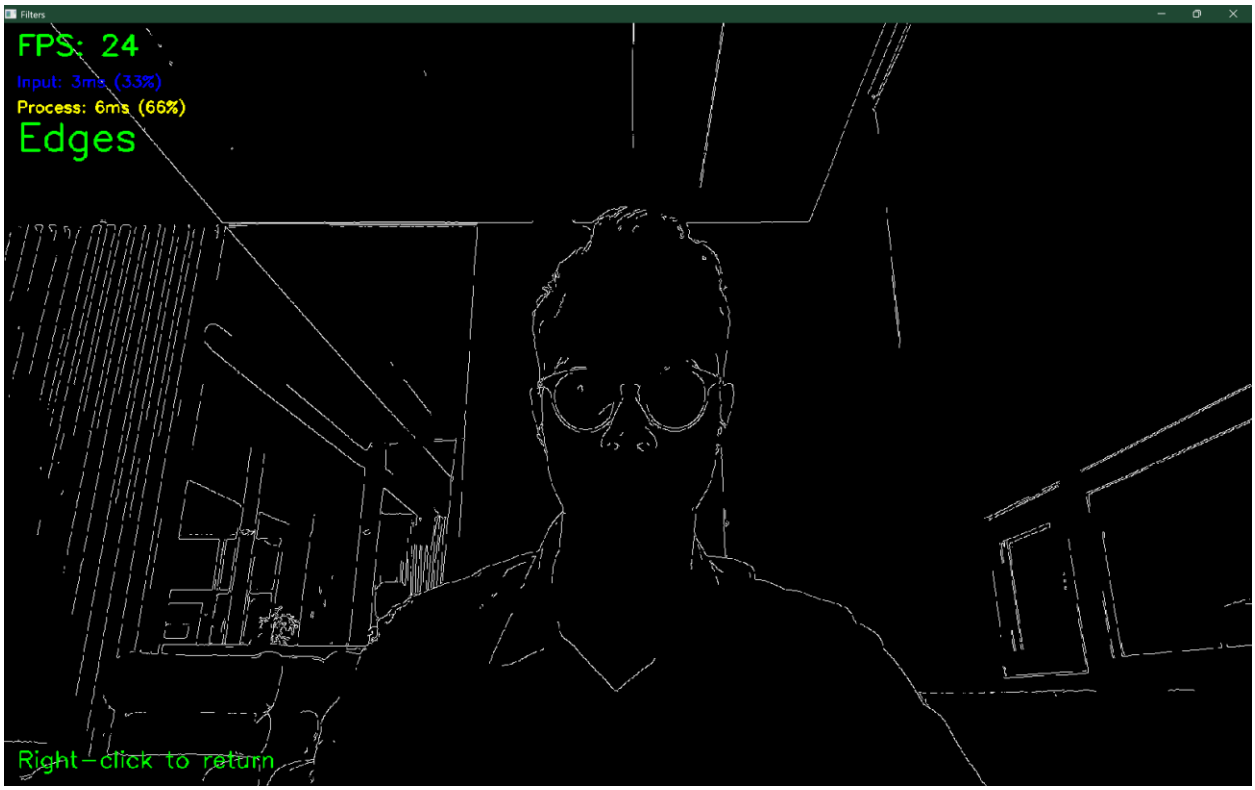


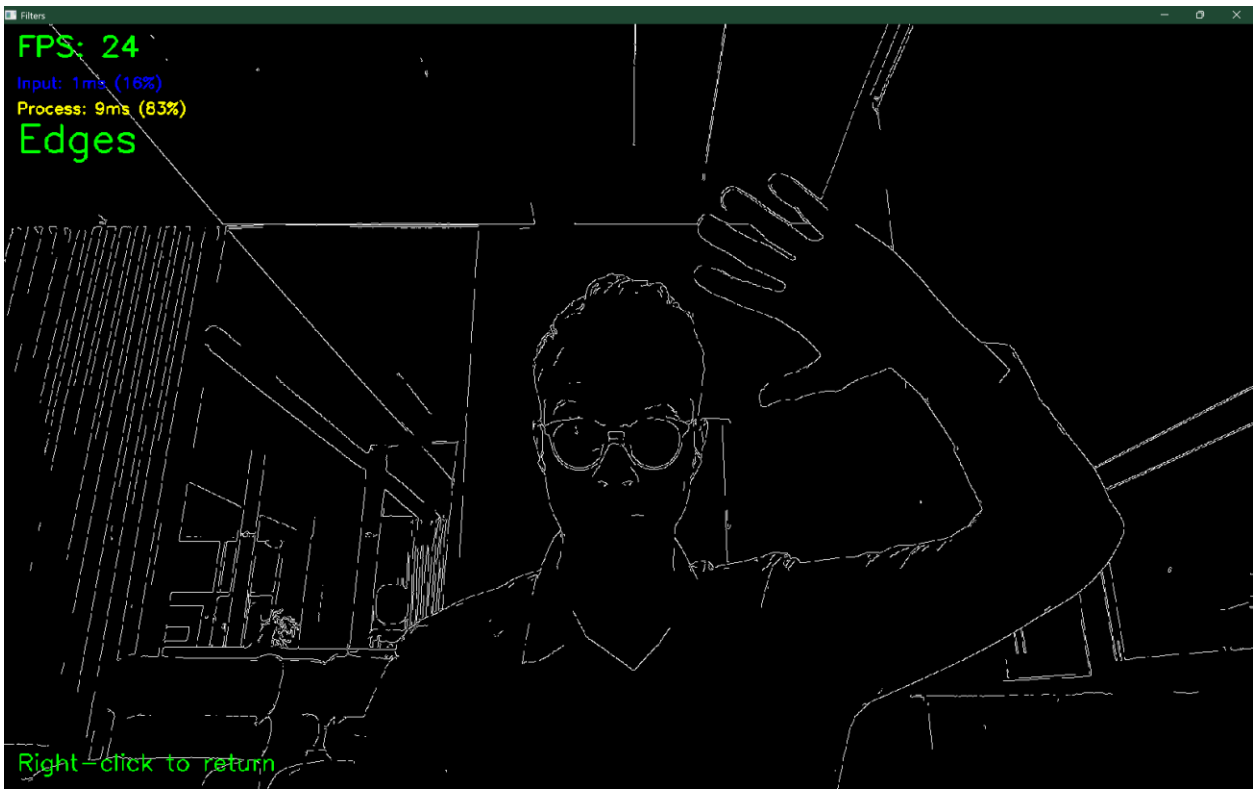
Приложение 3. *Вывод с камеры (Original)*



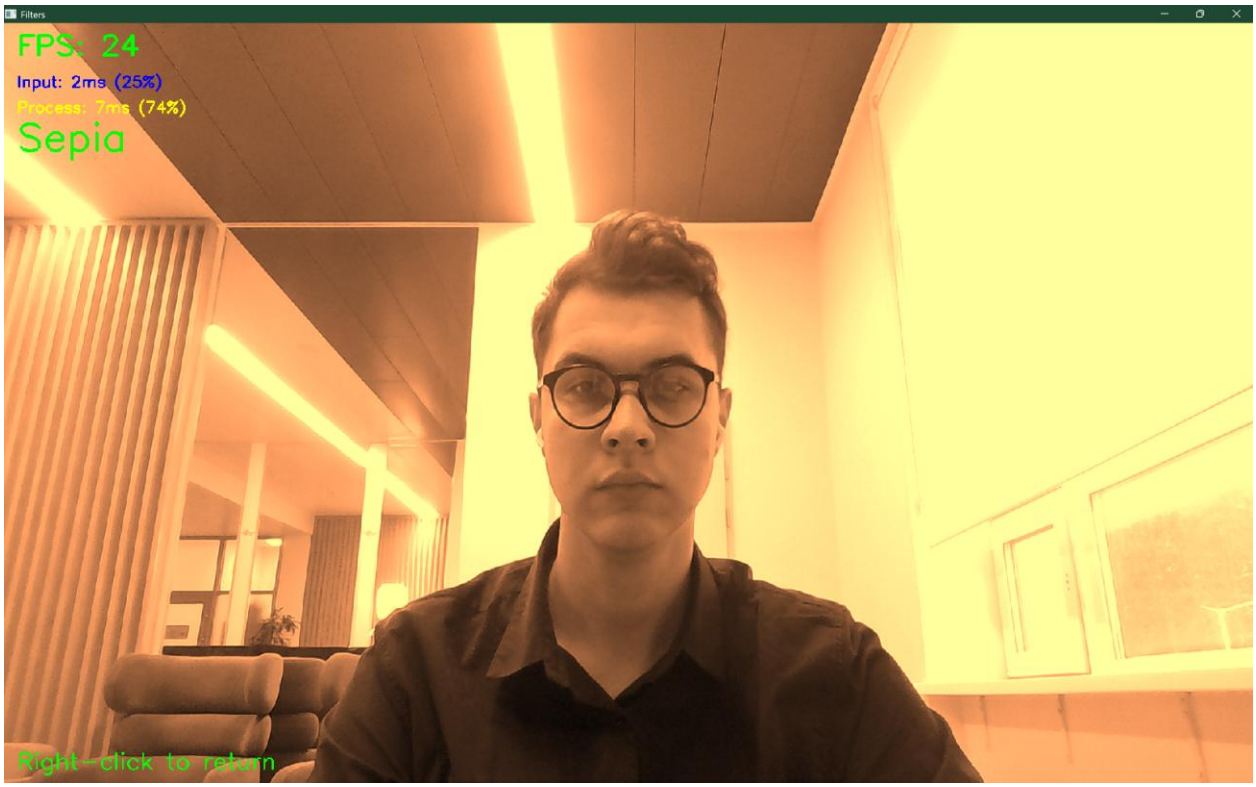


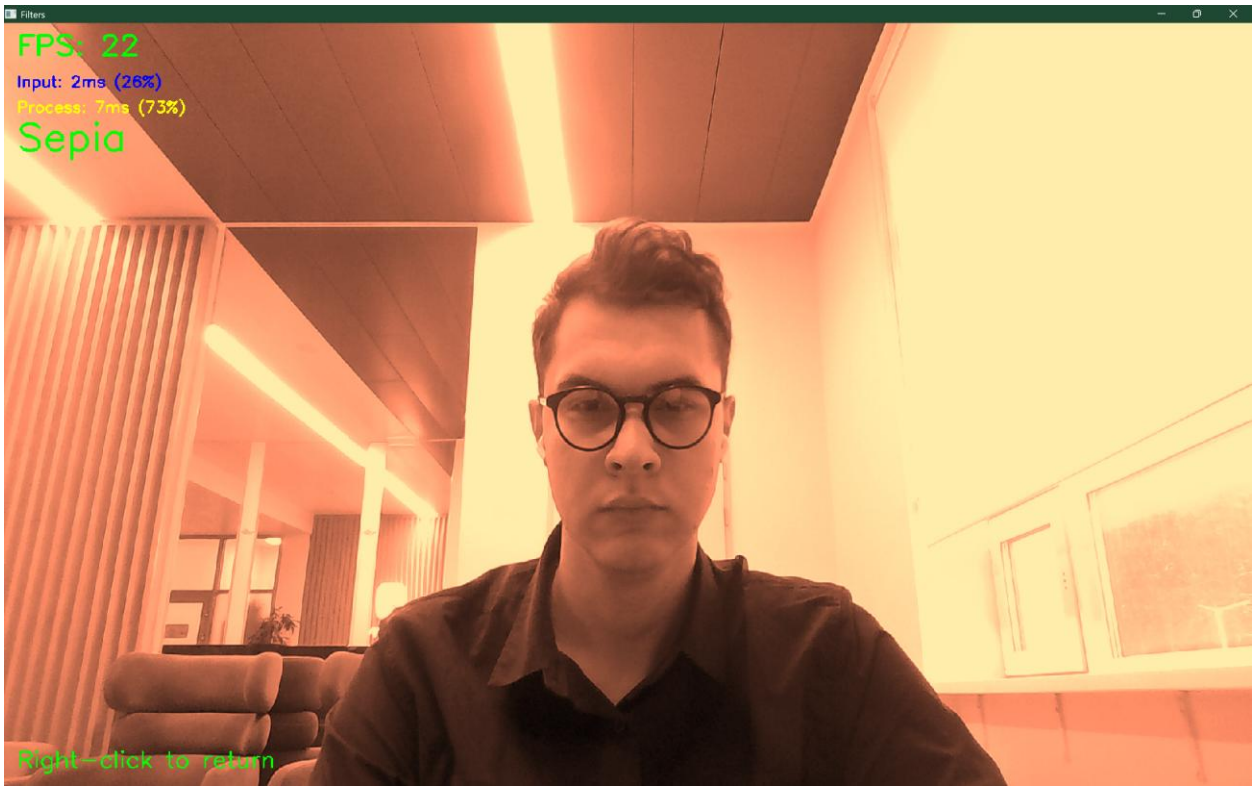
Приложение 4. *Вывод с камеры (Edge)*





Приложение 5. *Вывод с камеры (Sepia)*





Приложение 6. *Задание 2. Код программы на C++*

```
#include <iostream>
#include <libusb.h>
#include <iomanip>

using namespace std;

const char* getDeviceClass(uint8_t class_code) {
    switch(class_code) {
        case 0x00: return "Device";
        case 0x01: return "Audio";
        case 0x02: return "Communications";
        case 0x03: return "HID (Human Interface Device)";
        case 0x05: return "Physical";
        case 0x06: return "Image";
        case 0x07: return "Printer";
        case 0x08: return "Mass Storage";
        case 0x09: return "Hub";
        case 0x0A: return "CDC-Data";
        case 0x0B: return "Smart Card";
        case 0x0D: return "Content Security";
        case 0x0E: return "Video";
        case 0x0F: return "Personal Healthcare";
        case 0x10: return "Audio/Video";
        case 0xEF: return "Miscellaneous";
        case 0xFE: return "Application Specific";
        case 0xFF: return "Vendor Specific";
        default: return "Unknown";
    }
}

string getSerialNumber(libusb_device_handle *dev, uint8_t
serial_index) {
    if (serial_index == 0) {
        return "N/A";
    }

    unsigned char serial_number[256];
    int ret = libusb_get_string_descriptor_ascii(dev, serial_index,
serial_number, sizeof(serial_number));

    if (ret < 0) {
        return "Error reading serial";
    }

    return string(reinterpret_cast<char*>(serial_number));
}

int main() {
    libusb_device **devs = NULL;
    libusb_context *ctx = NULL;
    int r = libusb_init(&ctx);

    if (r < 0) {
        cerr << "Error initializing libusb: " << libusb_error_name(r)
<< endl;
```

```
        return -1;
    }

    // Логирование
    libusb_set_option(ctx, LIBUSB_OPTION_LOG_LEVEL,
LIBUSB_LOG_LEVEL_WARNING);

    // Список всех USB-устройств
    ssize_t cnt = libusb_get_device_list(ctx, &devs);

    if (cnt < 0) {
        cerr << "Error getting device list: " <<
libusb_error_name(cnt) << endl;
        libusb_exit(ctx);
        return -1;
    }

    cout << "Found " << cnt << " USB device(s)" << endl;
    cout << "======================================" << endl;
    cout << endl;

    for (ssize_t i = 0; i < cnt; i++) {
        libusb_device *dev = devs[i];
        libusb_device_descriptor desc{};

        // Дескриптор
        r = libusb_get_device_descriptor(dev, &desc);
        if (r < 0) {
            cerr << "Error getting device descriptor: " <<
libusb_error_name(r) << endl;
            continue;
        }

        cout << "Device " << (int)i + 1 << ":" << endl;

        // Вывод класса
        cout << " -- Class: " << getDeviceClass(desc.bDeviceClass)
             << " (0x" << hex << setfill('0') << setw(2) <<
(int)desc.bDeviceClass << ")" << dec << endl;

        // Вывод VID и PID
        cout << " ---- Vendor ID: 0x" << hex << setfill('0') <<
setw(4) << desc.idVendor << dec << endl;
        cout << " ------ Product ID: 0x" << hex << setfill('0') <<
setw(4) << desc.idProduct << dec << endl;

        libusb_device_handle *handle = NULL;
        r = libusb_open(dev, &handle);

        if (r == 0 && handle != NULL) {
            // Получение и вывод серийного номера
            string serial = getSerialNumber(handle,
desc.iSerialNumber);
            cout << " -------- Serial Number: " << serial << endl;

            libusb_close(handle);
        } else {
            cout << "  Serial Number: Could not open device (may
```

```
require root/sudo)" << endl;
        }

        cout << endl;
    }

    libusb_free_device_list(devs, 1);
    libusb_exit(ctx);

    cout << "Done!" << endl;
    return 0;
}
```

Приложение 7. *Результат работы программы*

```
Found 12 USB device(s)
======================================

Device 1:
 -- Class: Miscellaneous (0xef)
 ---- Vendor ID: 0x046d
 ------ Product ID: 0x0825
 -------- Serial Number: 95410D90

Device 2:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0002
 -------- Serial Number: 0000:00:1d.7

Device 3:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0001
 -------- Serial Number: 0000:00:1d.2

Device 4:
 -- Class: Device (0x00)
 ---- Vendor ID: 0x0458
 ------ Product ID: 0x003a
 -------- Serial Number: N/A

Device 5:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0001
 -------- Serial Number: 0000:00:1d.1

Device 6:
 -- Class: Device (0x00)
 ---- Vendor ID: 0x1c4f
 ------ Product ID: 0x0026
 -------- Serial Number: N/A

Device 7:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0001
```

```
 -------- Serial Number: 0000:00:1d.0

Device 8:
 -- Class: Device (0x00)
 ---- Vendor ID: 0x0bda
 ------ Product ID: 0x0181
 -------- Serial Number: 20060413092100000

Device 9:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0002
 -------- Serial Number: 0000:00:1a.7

Device 10:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0001
 -------- Serial Number: 0000:00:1a.2

Device 11:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0001
 -------- Serial Number: 0000:00:1a.1

Device 12:
 -- Class: Hub (0x09)
 ---- Vendor ID: 0x1d6b
 ------ Product ID: 0x0001
 -------- Serial Number: 0000:00:1a.0

Done!
```

Приложение 8. *Задание 1. CMakeLists.txt*

```
cmake_minimum_required(VERSION 3.15)
project(lab3)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

set(OpenCV_DIR
"D:/NSU/MyLabs/EVMandPU/evm_lab_works/lab3/opencv/build
-static")
set(OpenCV_STATIC ON)
set(BUILD_SHARED_LIBS OFF)

find_package(OpenCV REQUIRED COMPONENTS world)

add_executable(lab3 src/main.cpp)

target_include_directories(lab3                    PRIVATE
${OpenCV_INCLUDE_DIRS})
target_link_libraries(lab3 PRIVATE ${OpenCV_LIBS})

if(MSVC)
    set_property(TARGET lab3 PROPERTY
            MSVC_RUNTIME_LIBRARY
"MultiThreaded$<$<CONFIG:Debug>:Debug>")

    target_link_libraries(lab3 PRIVATE
            ws2_32
            comctl32
            gdi32
            ole32
            oleaut32
            uuid
            vfw32
    )
endif()

if(MINGW)
    target_link_options(lab3 PRIVATE
            "-static"
            "-static-libgcc"
            "-static-libstdc++"
    )
endif()
```

```
if(APPLE)
    target_link_libraries(lab3 PRIVATE
            "-framework IOKit"
            "-framework Cocoa"
            "-framework OpenGL"
    )
endif()
```

Приложение 9. *Задание 2. CMakeLists.txt*

```
cmake_minimum_required(VERSION 3.10)
project(USBDevices)

# Используем PkgConfig для поиска libusb
find_package(PkgConfig REQUIRED)
pkg_check_modules(libusb    REQUIRED    IMPORTED_TARGET
libusb-1.0)

# Создание исполняемого файла
add_executable(usb_devices usb_devices.cpp)

# Связывание с libusb через PkgConfig
target_link_libraries(usb_devices                PRIVATE
PkgConfig::libusb)

# Установка стандарта C++
set_property(TARGET  usb_devices  PROPERTY  CXX_STANDARD
11)
```