###### **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 ПРАКТИЧЕСКОЙ РАБОТЫ**

**Практическая работа №5**

ВЫСОКОУРОВНЕВАЯ РАБОТА С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ

студента 2 курса, группы 23201

Сорокина Матвея Павловича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.С. Матвеев

Новосибирск 2024

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ** 3](#_Toc180448182)

[**ЗАДАНИЕ** 3](#_Toc180448183)

[**ОПИСАНИЕ РАБОТЫ** 4](#_Toc180448184)

[**ЗАКЛЮЧЕНИЕ** 8](#_Toc180448185)

[**ПРИЛОЖЕНИЯ** 9](#_Toc180448186)

[**Приложение 1:Исходный код программы *lab5\_EVM.cpp*** 9](#_Toc180448187)

[**Приложение 2: пример результата работы программы** 10](#_Toc180448188)

# **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**

Полный листинг программы и пример ее импользования предоставлены. (см. Приложения 1,2)

В *OpenCV* существует множество модулей. Каждый из них отвечает за определенную область или подход к обработке изображений. Перед тем как использовать любой из модулей, нужно сначала подключить заголовочные файлы, в которых объявлено содержимое каждого отдельного модуля.

В моей программе используются:

* ***core*** - в данном модуле определены базовые структуры, вычисления (математические функции, генерация псевдослучайных чисел, DFT, DCT, ввод/вывод в XML и т.п.)
* ***highgui*** - содержит функции для отображения, загрузки и сохранения изображений и видео в окне (*GUI* функции).

|  |
| --- |
| cv::imshow("Original image", frame);  cv::imshow("Redacted image", redacted\_frame);  ...  cv::destroyAllWindows(); |

* ***imgproc*** - отвечает за обработку изображений (размытие).

|  |
| --- |
| cv::blur(frame, redacted\_frame, cv::Size(5, 5)); |

Также включены два заголовочных файла:

* ***iostream*** - заголовочный файл с классами, функциями и переменными для организации ввода-вывода. Используется для записи в стандартный поток вывода *stdout*.
* ***chrono*** - часть стандартной библиотеки, предназначена для работы с временем и временными интервалами. Она предоставляет средства для точного измерения времени, выполнения операций с временными отрезками.

|  |
| --- |
| auto start = std::chrono::high\_resolution\_clock::now();  …  auto end = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> elapsed = end - start;  double fps = frame\_count / elapsed.count(); |

Рассмотрим функцию *main*:

1. Открытие видеопотока с камеры (0 – первая подключенная камера).

***cv::VideoCapture*** — это класс в *OpenCV*, который отвечает за захват видеопотока с камеры или чтение видеофайла.

Также проверяем удалось ли открыть видеопоток с камеры. Для этого используется метод ***isOpened()****,* который возвращает *true*, если камера была успешно открыта, и *false* - если произошла ошибка при её открытии.

|  |
| --- |
| cv::VideoCapture capture(0);  if (!capture.isOpened()) {  std::cerr << "Error! Couldn't open camera" << std::endl;  return 0;  } |

1. Создание двух переменных класса *cv::Mat* – матрицы данных изображений, счетчика кадров и переменной, хранящей текущее время (хранит начальное время).

|  |
| --- |
| cv::Mat frame, redacted\_frame;  int frame\_count = 0;  auto start = std::chrono::high\_resolution\_clock::now(); |

1. Находясь в цикле считываем кадр объектом класса cv::VideoCapture, который используется для захвата видеопотока с камеры (или из файла). Захваченный кадр сохраняется в переменной ***frame.***

После захвата кадра проверяется, пуст ли кадр с помощью метода ***empty().*** Этот метод возвращает ***true***, если изображение в объекте ***frame*** не было загружено (например, из-за ошибки захвата кадра или завершения видеопотока).

Если операция успешна, то в ***frame*** будет содержаться изображение (кадр).

|  |
| --- |
| capture >> frame;  if (frame.empty()) {  std::cerr << "Error! Cannot capture the frame" << std::endl;  break;  } |

1. Размытие изображения ***frame***, создавая его размытую версию в ***redacted\_frame***. Размер ядра размытия — 5x5, значит, каждый пиксель на выходе будет усреднён на основе 5x5 соседних пикселей вокруг него.

|  |
| --- |
| cv::blur(frame, redacted\_frame, cv::Size(5, 5)); |

1. Применяем *colormap* к изображению ***frame***, добавляя ему «костяную» цветовую палитру и записываем результат в ***redacted\_frame.***

|  |
| --- |
| cv::applyColorMap(frame, redacted\_frame, cv::ColormapTypes::COLORMAP\_BONE); |

1. Инкрементация переменной ***frame\_count,*** которая отслеживает количество кадров, которые были обработаны с начала работы программы.

|  |
| --- |
| frame\_count++; |

1. Создается переменная ***current***, которая хранит текущее время, разница между текущим временем (***current***) и временем старта (***start***) вычисляется и записывается в инициализированную переменную ***elapsed*** типа ***std::chrono::duration<double>.***

Вычисляется количество кадров в секунду - деление количества кадров на прошедшее время и записывается в инициализированную переменную ***fps***.

Функция ***cv::putText*** используется для отображения текста на изображении, где:

* **frame** - неизмененное изображение, на которое будет наложен текст.
* **fps\_text** - строка с FPS, которую нужно отобразить.
* **cv::Point(10, 30)** - координаты верхнего левого угла текста на изображении (в пикселях). В данном случае текст будет отображаться в левом верхнем углу.
* **cv::FONT\_HERSHEY\_SIMPLEX** - шрифт, используемый для отображения текста.
* **1.0** - размер текста (масштаб шрифта).
* **cv::Scalar(0, 255, 0)** - цвет текста (зелёный).
* **2** - толщина линий, которыми будет нарисован текст.

|  |
| --- |
| frame\_count++;  auto current = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> elapsed = current - start;  double fps = frame\_count / elapsed.count();  std::string fps\_text = "FPS: " + std::to\_string(fps);  cv::putText(frame, fps\_text, cv::Point(10, 30), cv::FONT\_HERSHEY\_SIMPLEX, 1.0, cv::Scalar(0, 255, 0), 2); |

1. Отображение двух изображений в отдельных окнах функцией ***cv::imshow***

|  |
| --- |
| cv::imshow("Original image", frame);  cv::imshow("Redacted image", redacted\_frame); |

1. Проверка нажатия клавиши «Esc» для выхода из цикла *while*

***cv::waitKey*** - функция, которая ждёт нажатие клавиши на протяжении указанного времени (в миллисекундах).   
27 — это ASCII-код клавиши «Esc».

|  |
| --- |
| char c = (char)cv::waitKey(33);  if (c == 27) break; |

1. Вывод на экран:

* количества обработанных кадров,
* время, затраченное на выполнение программы,
* количество кадров в секунду.

|  |
| --- |
| auto end = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> elapsed = end - start;  double fps = frame\_count / elapsed.count();  std::cout << "Frames counted: " << frame\_count << std::endl;  std::cout << "Elapsed time: " << elapsed.count() << " seconds" << std::endl;  std::cout << "Amount of frames per second (FPS): " << fps << std::endl; |

A black background with white text

Description automatically generated

1. Закрытие видеопотока:

* Метод ***release()*** освобождает все ресурсы, связанные с объектом ***capture.*** В нашем случае освобождается камера, которая использовалась в реальном времени.
* Метод ***cv::destroyAllWindows()*** освобождает ресурсы, связанные с графическим интерфейсом (*HighHUI* окна).

|  |
| --- |
| capture.release();  cv::destroyAllWindows(); |

1. Завершение функции main возвратом 0 и завершение программы.

|  |
| --- |
| return 0; |

# **ЗАКЛЮЧЕНИЕ**

В ходе данной лабораторной работы я познакомился с библиотекой алгоритмов компьютерного зрения OpenCV а также освоил работу с встроенной камерой, изображениями, полученными с камеры, вывод обработанного изображения на экран и работу с окнами.

# **ПРИЛОЖЕНИЯ**

# **Приложение 1:Исходный код программы *lab5\_EVM.cpp***

|  |
| --- |
| #include <opencv2/opencv.hpp>  //#include <opencv2/core.hpp> // basic building blocks  //#include <opencv2/imgcodecs.hpp> // reading/writing functions (imshow)  //#include <opencv2/highgui.hpp> // GUI functions (namedWindow; destroyAllWindows)  //#include <opencv2/imgproc.hpp> // image processing (blur)  #include <iostream> // for console input/output  #include <chrono> // for time measurement  int main() {  // std::cout << "OpenCV version: " << CV\_VERSION << std::endl;  // открытие видеопотока с камеры (0 – первая подключенная камера)  cv::VideoCapture capture(0);  if (!capture.isOpened()) {  std::cerr << "Error! Couldn't open camera" << std::endl;  return 0;  }  cv::Mat frame, redacted\_frame; // переменные для хранения каров  int frame\_count = 0; // счетчик кадров  auto start = std::chrono::high\_resolution\_clock::now(); // начальное время  while (true) {  // считывание очередного кадра  capture >> frame;  if (frame.empty()) {  std::cerr << "Error! Cannot capture the frame" << std::endl;  break;  }  // редактирование изображения - сглаживание  cv::blur(frame, redacted\_frame, cv::Size(5, 5));  // черно-белый эффект  cv::applyColorMap(frame, redacted\_frame, cv::ColormapTypes::COLORMAP\_BONE);  frame\_count++;  auto current = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> elapsed = current - start;  double fps = frame\_count / elapsed.count();  std::string fps\_text = "FPS: " + std::to\_string(fps);  cv::putText(frame, fps\_text, cv::Point(10, 30), cv::FONT\_HERSHEY\_SIMPLEX, 1.0, cv::Scalar(0, 255, 0), 2);  cv::imshow("Original image", frame);  cv::imshow("Redacted image", redacted\_frame);  // проверка нажатия Esc для выхода  char c = (char)cv::waitKey(33); // 33 мс ожидаем нажатие клавиши  if (c == 27) break;  }  // вывод продолжительности работы программы  auto end = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> elapsed = end - start;  double fps = frame\_count / elapsed.count();  std::cout << "Frames counted: " << frame\_count << std::endl;  std::cout << "Elapsed time: " << elapsed.count() << " seconds" << std::endl;  std::cout << "Amount of frames per second (FPS): " << fps << std::endl;  // закрытие видеопотока  capture.release();  cv::destroyAllWindows();  return 0;  }  // запуск программы  // g++ main.cpp `pkg-config --cflags --libs opencv4`  // проверил работу камеры с помощью команд:  // cheese  // guvcview  // v4l2-ctl --list-devices  // https://docs.opencv.org/4.x/dd/d6e/tutorial\_windows\_visual\_studio\_opencv.html |

# **Приложение 2: пример результата работы программы**

A couple of people sitting in a room

Description automatically generated