## Задачи этой лабораторной(fps) нужны были для оценки времени обработки изображений.

## Где используется OpenCV(open source computer vision library)

Библиотека OpenCV предоставляет высокоуровневый интерфейс для разработки прикладных программ в области машинного зрения.

В ходе выполнения лабораторной работы были изучены основные группы функций библиотеки OpenCV:

1. работа с растровыми изображениями,
2. работа с видеоданными,
3. интерфейс для работы с камерами,
4. машинное обучение с помощью нейронных сетей и других моделей,
5. реализация упрощенного оконного интерфейса,
6. работа с векторами, матрицами и др.

**Основные группы функций OpenCV:**

1. **Работа с растровыми изображениями и их форматами**:
   * **Загрузка и сохранение изображений**: OpenCV поддерживает множество форматов изображений, таких как JPEG, PNG, BMP, TIFF и другие. Функции imread и imwrite позволяют загружать и сохранять изображения.
   * **Обработка изображений**: Библиотека предоставляет широкий набор функций для обработки изображений, включая фильтрацию, морфологические операции, преобразования цветового пространства и другие.
   * **Геометрические преобразования**: OpenCV поддерживает операции поворота, масштабирования, сдвига и другие геометрические преобразования изображений.
2. **Работа с видеоданными и их форматами**:
   * **Захват видео**: OpenCV позволяет захватывать видео с камер и видеофайлов с помощью класса VideoCapture.
   * **Обработка видео**: Библиотека предоставляет функции для обработки видеопотока, включая детектирование движения, отслеживание объектов и другие.
   * **Сохранение видео**: OpenCV позволяет записывать обработанное видео в файл с помощью класса VideoWriter.
3. **Интерфейс для работы с камерами**:
   * **Поддержка различных камер**: OpenCV поддерживает работу с различными типами камер, включая веб-камеры, IP-камеры и другие.
   * **Настройка параметров камеры**: Библиотека позволяет настраивать параметры камеры, такие как фокусное расстояние, яркость, контрастность и другие.
4. **Машинное обучение с помощью нейронных сетей и других моделей**:
   * **Поддержка нейронных сетей**: OpenCV включает модуль dnn (Deep Neural Networks), который позволяет загружать и использовать предварительно обученные нейронные сети для задач классификации, детектирования объектов и других.
   * **Другие модели машинного обучения**: OpenCV также поддерживает другие модели машинного обучения, такие как SVM (Support Vector Machines), K-Means и другие.
5. **Реализация упрощенного оконного интерфейса**:
   * **Отображение изображений и видео**: OpenCV предоставляет функции для отображения изображений и видео в окнах, такие как imshow и namedWindow.
   * **Обработка событий**: Библиотека позволяет обрабатывать события, такие как нажатие клавиш и движение мыши, с помощью функции waitKey и других.
6. **Операции над векторами и матрицами**:
   * **Матричные операции**: OpenCV предоставляет мощные инструменты для работы с матрицами, включая базовые операции (сложение, умножение, транспонирование) и более сложные (SVD, QR-разложение и другие).
   * **Векторные операции**: Библиотека также поддерживает операции над векторами, такие как скалярное произведение, нормализация и другие.
7. **Другие функции**:
   * **Геометрические преобразования**: OpenCV поддерживает различные геометрические преобразования, такие как преобразование Хафа, детектирование линий и окружностей.
   * **Обработка сигналов**: Библиотека включает функции для обработки сигналов, такие как фильтрация и преобразование Фурье.
   * **Работа с графическими примитивами**: OpenCV позволяет рисовать графические примитивы (линии, окружности, прямоугольники) на изображениях.

Чтобы программа умела смотреть на изображение не как на набор пикселей, а как на набор известных ей объектов, используют **машинное обучение.**

**Как OpenCV воспринимает изображения**

**Если изображение в оттенках серого**, то массив этот двумерный. Каждый пиксель в изображении представлен в виде числа 0 до 255, где 0 — чёрный, 255 — белый, а всё остальное — оттенки серого между ними.

Каждая пиксельная строка объединена в одномерный массив. Например, [255, 255, 77, 77, 77, 255, 255]. Это строка из семи пикселей: двух белых, трёх серых и ещё двух белых.

Каждая такая пиксельная строка объединена в массив второго уровня. Например:

**Если изображение цветное**, то потребуется трёхмерный массив. Цвет каждого пикселя описывает не одно, а сразу три числа от 0 до 255 — в соответствии с [моделью RGB](https://skillbox.ru/media/design/rgb-cmyk-lab/#stk-2?utm_source=media&utm_medium=link&utm_campaign=all_all_media_links_links_articles_all_all_skillbox). Только красный и синий цвет переставлены местами, так что первое число отвечает за синий цвет, второе — за зелёный, третье — за красный: получается BGR.

Вот примеры пикселей разного цвета на языке OpenCV:

* [0, 0, 0] — чёрный.
* [255, 255, 255] — белый.
* [255, 0, 0] — синий.
* [0, 255, 0] — зелёный.
* [0, 0, 255] — красный.
* [0, 255, 255] — жёлтый.
* [203, 192, 255] — розовый.

Как и в случае с изображениями в оттенках серого, массив следующего уровня объединяет строки пикселей, а верхнеуровневый массив — всё изображение.

**Изменение цвета пикселей**

OpenCV позволяет обращаться к пикселям по их записи в NumPy-массиве. Так мы можем менять изображение.

Для начала **получим доступ к пикселю**. Для этого обратимся к нему по координатам. Сначала указываем координаты ширины, потом — высоты. Например:

blue, green, red = image[10, 350]

print(f'RGB-код пикселя: {red}, {green}, {blue}')

>>> RGB-код пикселя: 249, 7, 6

Теперь **изменим цвет пикселя**. Для этого обратитесь к нему по координатам и сделайте его синим. Помните, что синий и красный переставлены местами:

image[0, 0] = [255, 0, 0]

Если открыть изменённое изображение, то в правом верхнем углу вы увидите чёрную точку размером с один пиксель.

**Как с помощью OpenCV получить изображение с видеокамеры и вывести его на экран?**

* 1. Создается поток ввода видеоданных с первой камеры (нумерация начинается с нуля).
* videocapture – класс, который представляет собой поток ввода видеоданных (камера – номер, номер камеры на устройстве, путь к какому-то видео(откроет видео и будет входной поток видео). Класс для захвата видео из видеофайлов, последовательностей изображений или камер.

Класс VideoCapture в библиотеке OpenCV (Open Source Computer Vision Library) является абстракцией реального устройства или источника видеопотока. Это может быть:

1. **Веб-камера**: VideoCapture может быть использован для захвата видео с подключенной веб-камеры.
2. **Видеофайл**: VideoCapture также может читать видеопоток из файла на диске.
3. **IP-камера**: Если у вас есть IP-камера, которая транслирует видео по сети, VideoCapture может подключиться к ней через URL.
4. **Другой источник видео**: В некоторых случаях VideoCapture может работать с другими источниками видео, такими как потоковое видео через сеть.

Таким образом, VideoCapture абстрагирует сложность работы с различными источниками видео, предоставляя унифицированный интерфейс для захвата кадров видеопотока.

2. Запускается циклическая обработка потока видеоданных. while(1) {

3. На каждой итерации извлекается очередной кадр (изображение) из открытого потока ввода видеоданных.

Класс Mat в OpenCV — это основной класс для работы с изображениями и матрицами данных. Он представляет собой многомерную матрицу, которая может хранить данные различных типов (например, 8-битные целые числа, 32-битные числа с плавающей точкой и т.д.). Mat является одним из ключевых классов в OpenCV и используется практически во всех операциях с изображениями и матрицами.

### Основные характеристики класса Mat

1. **Многомерная матрица**: Mat может представлять как двумерные матрицы (например, изображения), так и многомерные массивы данных.
2. **Автоматическое управление памятью**: OpenCV автоматически управляет памятью, выделенной для объектов Mat. Это означает, что память будет освобождена, когда объект Mat выходит из области видимости.
3. **Различные типы данных**: Mat может хранить данные различных типов, таких как CV\_8U (8-битные целые числа без знака), CV\_32F (32-битные числа с плавающей точкой) и т.д.
4. **Разреженные матрицы**: Mat также поддерживает разреженные матрицы, которые хранят только ненулевые элементы.

Mat – хранит изображение(однокональный – чернобелое, трехканальный – цветные, четырехканальный). Матрица, в которой каждое значение – пиксель, пиксель состоит из кол-во значений равной кол-ву каналов и каждый канал кодируется 1,2, 4 байт. В зависимости от того, что возвращает video capture

Mat frame;

capture.read(frame); // захватывает текущий кадр с веб-камеры и сохраняет его в переменной frame

4. Текущий кадр выводится в окно с именем imshow("camera1", frame);

**5.**Нажатие клавиши пользователем

if (waitKey(10) >= 0)

break;

Используется в цикле обработки видео или изображений для проверки нажатия клавиши пользователем.

Это условие проверяет, была ли нажата какая-либо клавиша в течение 10 миллисекунд.

* + Если клавиша была нажата, waitKey(10) вернет код нажатой клавиши, который будет больше или равен 0.
  + Если клавиша не была нажата, waitKey(10) вернет -1.

break;

Оператор break используется для выхода из текущего цикла. В данном контексте, если условие if (waitKey(10) >= 0) истинно (т.е. была нажата какая-либо клавиша), выполнение цикла прерывается, и программа переходит к следующему блоку кода после цикла.

1. Какие способы преобразования изображений можно использовать, применяя OpenCV.

Библиотека OpenCV предоставляет набор функций, позволяющих выполнять различные преобразования изображений, например, сглаживание, морфологические преобразования (сужение и расширение), пороговые преобразования и другие.

**Преобразование цветового пространства:**

* **RGB в HSV**:

Mat hsv\_frame;

cvtColor(frame, hsv\_frame, COLOR\_BGR2HSV);

**К отчету:**

### 1. Работа с растровыми изображениями

**Работа с растровыми изображениями** включает в себя множество операций, таких как загрузка, сохранение, отображение, преобразование и обработка изображений.

* **Отображение изображения**:

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", image);

waitKey(0);

* + namedWindow создает окно для отображения изображения.
  + imshow отображает изображение в окне.
  + waitKey ожидает нажатия клавиши.
* **Преобразование цветового пространства**:

Mat gray\_image;

cvtColor(image, gray\_image, COLOR\_BGR2GRAY);

* + cvtColor преобразует изображение из одного цветового пространства в другое (например, из BGR в оттенки серого).

### 2. Работа с видеоданными

**Работа с видеоданными** включает в себя захват, обработку и отображение видеопотока.

### 3. Интерфейс для работы с камерами

**Интерфейс для работы с камерами** позволяет настраивать параметры камеры и получать данные с камеры.

#### Основные функции:

* **Настройка параметров камеры**:

cap.set(CAP\_PROP\_FRAME\_WIDTH, 640);

cap.set(CAP\_PROP\_FRAME\_HEIGHT, 480);

* + cap.set настраивает параметры камеры, такие как ширина и высота кадра.
* **Получение параметров камеры**:

double width = cap.get(CAP\_PROP\_FRAME\_WIDTH);

double height = cap.get(CAP\_PROP\_FRAME\_HEIGHT);

* + cap.get получает текущие параметры камеры.

### 4. Реализация упрощенного оконного интерфейса

**Реализация упрощенного оконного интерфейса** позволяет создавать окна для отображения изображений и видео, а также обрабатывать события, такие как нажатие клавиш и движение мыши.

#### Основные функции:

* **Создание окна**:

namedWindow("Window", WINDOW\_AUTOSIZE);

* + namedWindow создает окно с указанным именем.
* **Отображение изображения в окне**:

imshow("Window", image);

* + imshow отображает изображение в окне.
* **Обработка событий**:

while (true) {

imshow("Window", image);

if (waitKey(30) >= 0) break;

}

* + waitKey ожидает нажатия клавиши и возвращает код нажатой клавиши.

Функция cvtColor в OpenCV используется для преобразования изображения из одного цветового пространства в другое. В вашем примере cvtColor(frame, modified\_frame, COLOR\_BGR2HSV); выполняет преобразование изображения из цветового пространства BGR (Blue-Green-Red) в цветовое пространство HSV (Hue-Saturation-Value).

**HSV** – Данное отображение создано для упрощения представление цвета человеком. Люди в случае описания цвета редко пользуются компонентами цвета, обычно они применяют такие слова как оттенок(**HUE**), насыщенность(**SATURATION**) и интенсивность(**VALUE**) цвета.

### Цветовые пространства

#### 1. BGR (Blue-Green-Red)

* **BGR** — это стандартное цветовое пространство, используемое в OpenCV по умолчанию.
* Каждый пиксель изображения представлен тремя компонентами: синий (B), зеленый (G) и красный (R).
* Значения каждого компонента находятся в диапазоне от 0 до 255.

#### 2. HSV (Hue-Saturation-Value)

* **HSV** — это альтернативное цветовое пространство, которое часто используется в задачах компьютерного зрения, таких как сегментация цветов и отслеживание объектов по цвету.
* **Hue (Оттенок)**: Представляет цвет и изменяется от 0 до 179 (в OpenCV).
* **Saturation (Насыщенность)**: Определяет "чистоту" цвета и изменяется от 0 до 255.
* **Value (Яркость)**: Определяет яркость цвета и изменяется от 0 до 255.

### Преобразование BGR в HSV

При преобразовании из BGR в HSV каждый пиксель изображения пересчитывается из трех компонентов (B, G, R) в три новых компонента (H, S, V). Это преобразование позволяет более удобно работать с цветом, особенно в задачах, где важна сегментация по цвету.

* **HSV** — это цветовая модель, которая позволяет работать с цветом и яркостью изображения.
* **Негатив** — это преобразование, при котором цвета изображения инвертируются.
* **RGB**: Широко используется в компьютерной графике, веб-разработке и многих других областях. Формат RGB является стандартным для большинства графических программ и библиотек.
* **BGR**: Используется в библиотеке OpenCV по историческим причинам. OpenCV изначально была написана на C++, и в C++ массивы часто хранятся в порядке, обратном тому, что используется в других языках и библиотеках.

Морфологические преобразования в OpenCV предоставляют мощный инструмент для обработки изображений, позволяя изменять форму и структуру объектов на изображении. Они полезны для различных задач, таких как удаление шума, выделение границ, заполнение дыр и т.д.

Морфологические преобразования — это операции, которые изменяют форму и структуру объектов на бинарных или полутоновых изображениях. Они основаны на двух основных операциях: эрозии и дилатации.

**Основные морфологические операции**

1. **Эрозия (Erosion)**
   * **Описание**: Эрозия "съедает" границы объектов, уменьшая их размер.
   * **Применение**: Используется для удаления мелких объектов или шума, который меньше структурного элемента.
   * **Формула**:

Erosion(I)=min⁡(x,y)∈BI(x,y)Erosion(*I*)=min(*x*,*y*)∈*B*​*I*(*x*,*y*)  
где B*B* — структурный элемент.

1. **Дилатация (Dilation)**
   * **Описание**: Дилатация увеличивает размеры объектов, "расширяя" их границы.
   * **Применение**: Используется для заполнения маленьких дыр и углублений в объектах.
   * **Формула**:

Dilation(I)=max⁡(x,y)∈BI(x,y)Dilation(*I*)=max(*x*,*y*)∈*B*​*I*(*x*,*y*)  
где B*B* — структурный элемент.

**Комбинированные морфологические операции**

1. **Открытие (Opening)**
   * **Описание**: Открытие — это последовательное применение эрозии, за которой следует дилатация.
   * **Применение**: Используется для удаления мелких объектов и шума, сохраняя при этом форму и размер основных объектов.
   * **Формула**:

Opening(I)=Dilation(Erosion(I))Opening(*I*)=Dilation(Erosion(*I*))

1. **Закрытие (Closing)**
   * **Описание**: Закрытие — это последовательное применение дилатации, за которой следует эрозия.
   * **Применение**: Используется для заполнения маленьких дыр и трещин в объектах, сохраняя при этом их общую форму.
   * **Формула**:

Closing(I)=Erosion(Dilation(I))Closing(*I*)=Erosion(Dilation(*I*))

**Специальные морфологические операции**

1. **MORPH\_TOPHAT**
   * **Описание**: Разница между исходным изображением и результатом его открытия.
   * **Применение**: Выделяет светлые области на темном фоне, которые меньше структурного элемента.
   * **Формула**:

TopHat(I)=I−Opening(I)TopHat(*I*)=*I*−Opening(*I*)

1. **MORPH\_BLACKHAT**
   * **Описание**: Разница между результатом закрытия изображения и самим изображением.
   * **Применение**: Выделяет темные области на светлом фоне, которые меньше структурного элемента.
   * **Формула**:

BlackHat(I)=Closing(I)−I

Структурный элемент (Structuring Element) — это матрица, которая определяет область, в которой будут выполняться морфологические операции (эрозия, дилатация, открытие, закрытие и т.д.). Структурный элемент определяет форму и размер области, которая будет использоваться для сравнения с пикселями изображения.

**Основные характеристики структурного элемента**

1. **Форма**: Структурный элемент может иметь различные формы, такие как прямоугольник, эллипс или крест. Форма определяет, как будет проходить сравнение с пикселями изображения.
2. **Размер**: Размер структурного элемента определяет область, в которой будет происходить сравнение. Чем больше размер, тем больше область, которая будет учитываться при выполнении операций.
3. **Центральный пиксель**: Структурный элемент имеет центральный пиксель, который определяет текущую позицию на изображении, где выполняется операция.

**Создание структурного элемента в OpenCV**

В OpenCV структурный элемент создается с помощью функции getStructuringElement. Эта функция принимает два параметра:

1. **Форма**: Определяет форму структурного элемента. Возможные значения:
   * MORPH\_RECT: Прямоугольник.
   * MORPH\_ELLIPSE: Эллипс.
   * MORPH\_CROSS: Крест.
2. **Размер**: Определяет размер структурного элемента в виде объекта Size.

Пример создания структурного элемента:

Mat kernel = getStructuringElement(MORPH\_RECT, Size(5, 5));

Этот код создает прямоугольный структурный элемент размером 5x5 пикселей.

**Применение структурного элемента**

Структурный элемент используется в различных морфологических операциях. Например, при эрозии структурный элемент перемещается по изображению, и для каждого положения центрального пикселя структурного элемента вычисляется минимальное значение в области, определяемой структурным элементом.

**FPS(**Frames Per Second)

Библиотека <ctime> предоставляет функции для работы со временем в C++. В коде используется функция clock(), которая возвращает количество тактов процессора, прошедших с начала работы программы. Это значение можно использовать для измерения времени выполнения различных частей кода.

#### Как считается время

1. **Захват кадра**:

clock\_t c\_start = clock();

Mat frame;

capture.read(frame); // захватывает текущий кадр с веб-камеры и сохраняет его в переменной frame

inputTime += (clock() - c\_start); //время, затраченное на захват кадра

* + c\_start сохраняет текущее значение тактов процессора перед захватом кадра.
  + После захвата кадра вычисляется разница между текущим значением тактов и c\_start, что дает время, затраченное на захват кадра.
  + Это время добавляется к переменной inputTime.

1. **Обработка кадра**:

c\_start = clock();

Mat modified\_frame;

cvtColor(frame, modified\_frame, COLOR\_BGR2HSV);

// Применение MORPH\_TOPHAT

Mat tophat;

morphologyEx(frame, tophat, MORPH\_TOPHAT, kernel);

// Применение MORPH\_BLACKHAT

Mat blackhat;

morphologyEx(frame, blackhat, MORPH\_BLACKHAT, kernel);

procTime += (clock() - c\_start); //время, затраченное на обработку кадра

* + c\_start обновляется перед началом обработки кадра.
  + После обработки кадра вычисляется разница между текущим значением тактов и c\_start, что дает время, затраченное на обработку кадра.
  + Это время добавляется к переменной procTime.

1. **Отображение кадра**:

c\_start = clock();

imshow("camera1", frame); //отображает обработанный кадр в окне "camera1"

imshow("camera2", modified\_frame); //отображает обработанный кадр в окне "camera2"

imshow("MORPH\_TOPHAT", tophat); //отображает результат MORPH\_TOPHAT

imshow("MORPH\_BLACKHAT", blackhat); //отображает результат MORPH\_BLACKHAT

outputTime += (clock() - c\_start); //время, затраченное на отображение кадра

* + c\_start обновляется перед началом отображения кадра.
  + После отображения кадра вычисляется разница между текущим значением тактов и c\_start, что дает время, затраченное на отображение кадра.
  + Это время добавляется к переменной outputTime.

#### Расчет FPS

FPS (Frames Per Second) — это количество кадров, которые программа может обработать и отобразить за одну секунду. В вашем коде FPS рассчитывается следующим образом:

totalTime = clock() - totalTime;

cout << "frame per second: " << ((frameCount) / (totalTime / 1000.0)) << endl;

* totalTime вычисляется как разница между текущим значением тактов и начальным значением totalTime.
* Затем это значение преобразуется в миллисекунды (totalTime / 1000.0).
* FPS рассчитывается как отношение количества обработанных кадров (frameCount) к общему времени в секундах.

#### Расчет процентов времени

Для расчета процентов времени, затраченного на чтение, обработку и отображение кадров, используется следующий код:

totalTime = inputTime + procTime + outputTime;

double procent = totalTime / 100.0;

cout << "Reading time: " << inputTime / procent << "%" << endl;

cout << "Processing time : " << procTime / procent << "%" << endl;

cout << "The time of the withdrawal: " << outputTime / procent << "%" << endl;

* totalTime суммирует время, затраченное на чтение, обработку и отображение кадров.
* procent вычисляется как totalTime / 100.0, что дает базовое значение для расчета процентов.
* Проценты времени вычисляются как отношение времени на каждый этап к procent.

Изначально fps(кол-во тактов в секунду) выставлялись так, чтобы было примерно равно 30?

totalTime измеряется в тактах процессора, возвращаемых функцией clock(). Для того чтобы преобразовать это значение в секунды, необходимо разделить его на количество тактов в секунду, которое определяется константой CLOCKS\_PER\_SEC. Однако, в вашем коде используется другой подход: значение totalTime делится на 1000.0, чтобы преобразовать его в миллисекунды. Давайте разберем, почему это делается и какой результат это дает.

**Почему переводим в миллисекунды**

1. **Понимание единиц измерения**:
   * clock() возвращает количество тактов процессора, прошедших с начала работы программы.
   * CLOCKS\_PER\_SEC — это количество тактов в секунду. На большинстве систем CLOCKS\_PER\_SEC равно 1 000 000 (миллион тактов в секунду).
2. **Преобразование в секунды**:
   * Для преобразования тактов в секунды, нужно разделить значение, возвращаемое clock(), на CLOCKS\_PER\_SEC.
   * Однако, в вашем коде используется деление на 1000.0, чтобы получить время в миллисекундах. Это делается для удобства, так как миллисекунды — более привычная единица измерения для таких задач.

**Расчет FPS**

В вашем коде FPS рассчитывается следующим образом:

totalTime = clock() - totalTime;

cout << "frame per second: " << ((frameCount) / (totalTime / 1000.0)) << endl;

* totalTime — это общее время выполнения цикла в тактах.
* totalTime / 1000.0 преобразует это время в миллисекунды.
* Затем это значение преобразуется в секунды, разделив на 1000.0.
* FPS рассчитывается как отношение количества обработанных кадров (frameCount) к общему времени в секундах.