### *Дедлайн: 27/10/25 12:00*

### Лабораторная работа №1. Применение фильтров для обработки изображений

#### Ход выполнения

1. **Медианный фильтр**. С помощью метода cv2.medianBlur()реализуйте медианный фильтр.

import cv2

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

filtered\_frame = cv2.medianBlur(frame, 5)

cv2.imshow('Original', frame)

cv2.imshow('Filtered', filtered\_frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

1. **Размытие.** **Выберите изображение в формате JPEG, на котором присутствуют четко различимые объекты.** Примените к исходному изображению Гауссово размытие. Выберите размер ядра размытия, основываясь на характеристиках изображения и желаемом уровне размытия.

blurred\_image = cv2.GaussianBlur(image, (5, 5), 0)

1. **Повышение резкости. Для исходного изображения протестируйте и выберите наиболее оптимальный, на ваш взгляд, способ повышения резкости.**

**3.1 Примените к исходному изображению свертку с ядром**

kernel = np.array([[0, -1, 0],

[-1, 5, -1],

[0, -1, 0]])

sharpened = cv2.filter2D(img, -1, kernel)

**3.2 В качестве альтернативного метода используйте “Маску нерезкости”. Данный метод состоит из двух этапов:**

**- размытие изображения**

**- вычитание размытого изображения из оригинала.**

blurred = cv2.GaussianBlur(image, (5, 5), 0)

sharpened = cv2.addWeighted(image, 1.5, blurred, -0.5, 0)

**Функция** cv2.addWeighted() **используется для комбинирования двух изображений с заданными весами, Здесь** 1.5**,** -0.5 **– веса изображений,** 0 **– смещение (не применяется в данном случае). Отрицательное значение** -0.5 **указывает на то, что размытие будет вычитаться из оригинального изображения.**

1. **Выделение границ.** Примените оператор Собеля к исходному изображению. В OpenCV оператор Собеля реализован с помощью функции cv2.Sobel(), которая возвращает изображение, содержащее градиенты по указанным направлениям X и Y в виде 2D массива, где значения представляют собой величину градиента

***Синтаксис***

cv2.Sobel(src, ddepth, dx, dy, ksize=3, scale=1, delta=0, borderType=cv2.BORDER\_DEFAULT)

#### Параметры

**src -** входное изображение, к которому будет применяться оператор Собеля. Обычно это изображение в градациях серого (одноканальное), но может быть и цветным (в этом случае будет применяться к каждому каналу отдельно).

**ddepth** - тип данных для хранения результата, например:

cv2.CV\_8U: 8-битное беззнаковое целое.

cv2.CV\_16S: 16-битное знаковое целое.

cv2.CV\_64F: 64-битное число с плавающей запятой (рекомендуется для получения более точных значений градиента)

**dx** - указывает, будет ли вычисляться градиент по X (1) или нет (0)

**dy** - указывает, будет ли вычисляться градиент по Y (1) или нет (0)

**ksize** - размер ядра, используемого для вычисления градиента. Должен быть нечетным числом (например, 1, 3, 5, 7). Чем больше размер ядра, тем более сглаженные границы будут получены.

**scale** (необязательный) - коэффициент, который умножается на результат. Используется для изменения масштаба выходных значений.

**delta** (необязательный) - значение, которое добавляется к результату.

**borderType** (необязательный) - метод обработки границ изображения: cv2.BORDER\_DEFAULT, cv2.BORDER\_CONSTANT, cv2.BORDER\_REPLICATE и др. Определяет, как обрабатываются пиксели за пределами изображения.

***Пример***

edges = cv2.Sobel(image, cv2.CV\_64F, 1, 1, ksize=5)

Важно: Для дальнейшего комбинирования с лругими изображениями понадобится преобразование в 8-битный формат (от 0 до 255), который является стандартным для отображения изображений в OpenCV.

edges = cv2.convertScaleAbs(edges)

1. **Собственный фильтр**. С помощью метода cv2.filter2D() создайте реализуйте собственный сверточный фильтр.
2. **Комбинирование результатов**. С помощью метода cv2.addWeighted() скомбинируйте размытое изображения, выделенные границы и изображение с повышенной резкостью. Экспериментируйте с весовыми коэффициентами и последовательностью комбинирования, чтобы добиться оригинального визуального эффекта.

combined = cv2.addWeighted(blurred, 0.5, edges, 0.5, 0)   
combined = cv2.addWeighted(combined, 0.5, sharpened, 0.5, 0)

1. **Отображение результатов**. Итогом выполнения программного кода должны быть исходное изображение, все промежуточные результаты, а также результат комбинирования. Используйте библиотеку matplotlib для отображения оригинального изображения и всех промежуточных результатов

def show\_images(original, blurred, edges, sharpened, combined):

plt.figure(figsize=(12, 10))

plt.subplot(2, 3, 1)

plt.title('Оригинальное изображение')

plt.imshow(cv2.cvtColor(original, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.subplot(2, 3, 2)

plt.title('Размытие по Гауссу)')

plt.imshow(cv2.cvtColor(blurred, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.subplot(2, 3, 3)

plt.title('Выделение границ')

plt.imshow(cv2.cvtColor(edges, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.subplot(2, 3, 4)

plt.title('Повышение резкости')

plt.imshow(cv2.cvtColor(sharpened, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.subplot(2, 3, 5)

plt.title('Комбинация изображений')

plt.imshow(cv2.cvtColor(combined, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.tight\_layout()

plt.show()

show\_images(image, blurred, edges, sharpened, combined)

***Пример отображения результатов***



### Лабораторная работа №2. Выделение края с помощью фильтра Canny

### С помощью фильтра Канни преобразовать фотографию лица в “графический рисунок”

import cv2

image = cv2.imread('img/coins.jpg', 0)

edges = cv2.Canny(image, 100, 200)

cv2.imshow('Original Image', image)

cv2.imshow('Canny Edges', edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

### Лабораторная работа №3. Применение каскада Хаара для детекции лиц

#### Ход выполнения

### 1. Реализуйте элемент интерфейса для взаимодействия социального робота с пользователем. При появлении человека в кадре робот должен выполнять следующие действия:

### Проверить, улыбается ли пользователь, и открыты ли у него глаза.

### Если пользователь не улыбается, робот должен вывести на экран фразу “Улыбнись”.

### Если хотя бы один глаз пользователя закрыт, робот должен вывести на экран фразу “Открой глаза”.

### Пользователь должен видеть изображение своего лица в отдельном окне.

### Обнаруженные лицо, глаза и улыбка должны быть обведены в рамку.

### 2. Реализуйте вычисление значения FPS (Frames Per Second, количество обрабатываемых кадров в секунду). Значение FPS должно отображаться в верхнем левом углу кадра в реальном времени.

**FPS = 1 / время обработки одного кадра**

* Замерьте время начала обработки текущего кадра (с помощью time.time()).
* Вычислите time\_difference - разницу между текущим временем и временем предыдущего кадра.
* FPS определяется как 1 / time\_difference.
* Обновите время предыдущего кадра для следующего расчета.
* Выведите FPS на экран, обновляя его при обработке каждого кадра.

### В отчете приведите примеры успешного обнаружения глаз и улыбки, а также пример кадра с выводом сообщений ”Улыбнись” и “Открой глаза”.

### Также в отчете укажите полученное среднее значение FPS при выполнении программы, рассчитанное по времени работы системы.

**Отчет о выполнении лабораторных работ №1-3**

ФИО \_\_\_Лапшов Кирилл Юрьевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| **Лабораторная работа №1** |
| Исходное изображение и результаты обработки |
|  |

|  |
| --- |
| **Лабораторная работа №2** |
| Исходное изображение |
|  |
| Результат |
|  |

|  |
| --- |
| **Лабораторная работа №3** |
| Примеры результатов обработки кадров |
|  |
| Среднее значение FPS по времени работы системы |
| Среднее значение FPS за время работы: 29.48 |

https://github.com/Mraq98/LIRS.lab

*Исходный код доступен в репозитории GitHub по ранее предоставленной мной ссылке*