Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(УНИВЕРСИТЕТ ИТМО)

Факультет «Систем управления и робототехники»

ОТЧЕТ
О ЛАБОРАТОРНОЙ РАБОТЕ №2

По дисциплине «Техническое зрение»
на тему:
«Геометрические преобразования изображений»

Студенты:
Гуров Михаил Алексеевич 408510 R3243
Зыкин Леонид Витальевич 470912 R3335
Куликов Илья Вячеславович 470122 R3243

Преподаватель:
Шаветов Сергей Васильевич

г. Санкт-Петербург
2025

Цель работы:
Освоение основных видов отображений и использование геометрических преобразований для решения задач пространственной коррекции изображений.

Теоретическое обоснование применяемых методов и функций геометрических преобразований:

Геометрические преобразования изображений используются для изменения их структуры без потери информации. Основные виды преобразований включают:

1. **Сдвиг (cv.warpAffine)** – перемещение изображения на заданное расстояние в пикселях без изменения его формы.
2. **Масштабирование (cv.resize, cv.warpAffine)** – изменение размеров изображения с возможностью интерполяции для сохранения качества.
3. **Поворот (cv.getRotationMatrix2D, cv.warpAffine)** – вращение изображения относительно заданной точки.
4. **Скос (аффинные преобразования, cv.getAffineTransform)** – линейное изменение формы изображения, изменяя углы без искажения параллельных линий.
5. **Перспективные преобразования (cv.getPerspectiveTransform, cv.warpPerspective)** – изменение точки обзора с сохранением перспективы.
6. **Дисторсия и её коррекция** – устранение оптических искажений (бочкообразных и подушкообразных) с помощью радиальных функций.
7. **Автоматическая склейка (cv.Stitcher())** – объединение изображений с помощью поиска ключевых точек и определения гомографии.

Все операции выполняются с помощью функций OpenCV, которые используют **матричные преобразования** и **интерполяционные методы**, обеспечивая точность обработки изображений.

Ход выполнения работы:
Исходные изображения:

Рисунок 1 - изображение для всех заданий

Листинги программных реализаций:

Введение

Импортируем необходимые библиотеки (cv2, numpy, math) и подключеним функции ShowImages из модуля pa_utils для отображения изображений. Далее загружается изображение по указанному пути (fn = "images/cat.png"). Проверяем успешность загрузки: если изображение не загружено, выводится сообщение об ошибке. Если загрузка прошла успешно, изображение отображается с помощью ShowImages.

```
import cv2
import cv2 as cv
import numpy
import numpy as np
import math
from pa_utils import ShowImages

fn = "images/cat.png"
I = cv.imread(fn, cv.IMREAD_COLOR)
if not isinstance(I, np.ndarray) or I.data == None:
  print("Error reading file \"{}\"".format(fn))
else:
  ShowImages([("Source image", I)])
```

Задание 1. Простейшие геометрические преобразования.

Линейные трансформации

Сдвиг изображения

В данном задании выполняется сдвиг изображения на заданное расстояние по горизонтали и вертикали. Для этого создается матрица сдвига MM, в которой параметры Tx=50 и Ty=30 задают смещение на 50 пикселей вправо и 30 пикселей вниз. Затем, с помощью функции cv.warpAffine() применяется аффинное преобразование, осуществляющее данное перемещение. Итоговое изображение выводится на экран с помощью ShowImages().

```
# Определяем матрицу сдвига (Tx=50, Ty=30 - сдвиг на 50 пикселей вправо и 30 вниз)
h, w = I.shape[:2]
M = np.float32([[1, 0, 50], [0, 1, 30]])
shifted_image = cv.warpAffine(I, M, (w, h))

# Отображаем сдвинутое изображение
ShowImages([("Сдвинутое изображение", shifted_image)])
```

Отражение изображения

В данном задании выполняется зеркальное отражение изображения. Используется функция cv.flip(), где параметр определяет направление отражения:

- cv.flip(I, 1) – отражение по горизонтали (зеркально слева направо).
- cv.flip(I, 0) – отражение по вертикали (зеркально сверху вниз).
- cv.flip(I, -1) – отражение одновременно по горизонтали и вертикали. Результаты отображаются с помощью ShowImages().

```
flipped_horizontally = cv.flip(I, 1)
flipped_vertically = cv.flip(I, 0)
flipped_both = cv.flip(I, -1)

ShowImages([
    ("Отражение по горизонтали", flipped_horizontally),
    ("Отражение по вертикали", flipped_vertically),
    ("Отражение по обеим осям", flipped_both)
])
```

Однородное масштабирование изображения

В данном задании выполняется однородное масштабирование изображения с коэффициентом 1.5 с использованием аффинного преобразования. Создаётся матрица преобразования, в которой коэффициент масштабирования применяется к обеим осям. Затем функция cv.warpAffine() выполняет трансформацию, изменяя размеры изображения в соответствии с заданным масштабом. Итоговое изображение выводится на экран.

```
scale_factor = 1.5  # Коэффициент масштабирования
```

```
new_w = int(I.shape[1] * scale_factor)
new_h = int(I.shape[0] * scale_factor)

M = np.float32([[scale_factor, 0, 0], [0, scale_factor, 0]])

scaled_image = cv.warpAffine(I, M, (new_w, new_h))

ShowImages([("Масштабированное изображение", scaled_image)])
```

Поворот изображения

В данном задании выполняется поворот изображения на 45 градусов по часовой стрелке вокруг центра нижнего левого квартала. Реализованы два метода:

1. Ручное создание матрицы поворота
    - Создаются три матрицы: первая сдвигает изображение так, чтобы центр вращения совпадал с началом координат, вторая выполняет сам поворот, третья возвращает изображение обратно.
    - Итоговая матрица получается путем перемножения этих трёх матриц.
    - Функция cv.warpAffine() применяет преобразование к изображению.
2. Использование cv.getRotationMatrix2D()
    - Встроенная функция OpenCV автоматически создаёт матрицу поворота вокруг заданного центра.
    - cv.warpAffine() применяет преобразование.

Оба метода дают одинаковый результат, что подтверждает корректность ручного вычисления матрицы.

```
angle = 45  # Угол поворота в градусах

# Определяем центр поворота (нижний левый квартал)
center_x = int(I.shape[1] * 1/4)
center_y = int(I.shape[0] * 3/4)
center = (center_x, center_y)

# 1. Создание матрицы вручную с тремя матрицами (сдвиг -> поворот -> обратный сдвиг)
M_translate1 = np.array([[1, 0, -center_x], [0, 1, -center_y], [0, 0, 1]])
M_rotate = np.array([
    [math.cos(math.radians(-angle)), -math.sin(math.radians(-angle)), 0],
    [math.sin(math.radians(-angle)), math.cos(math.radians(-angle)), 0],
    [0, 0, 1]
])
```

```
M_translate2 = np.array([[1, 0, center_x], [0, 1, center_y], [0, 0, 1]])

# Итоговая матрица
M_manual = M_translate2 @ M_rotate @ M_translate1
M_manual = M_manual[:2, :]  # Преобразуем в 2x3 для warpAffine

# 2. Использование getRotationMatrix2D
M_opencv = cv.getRotationMatrix2D(center, angle, 1.0)

# Применение обеих матриц
rotated_manual = cv.warpAffine(I, M_manual, (I.shape[1], I.shape[0]))
rotated_opencv = cv.warpAffine(I, M_opencv, (I.shape[1], I.shape[0]))

# Отображаем результаты
ShowImages([
    ("Поворот вручную", rotated_manual),
    ("Поворот через getRotationMatrix2D", rotated_opencv)
])
```

Обратное аффинное преобразование.

В данном задании выполняется обратное аффинное преобразование изображения. Для этого используются две группы контрольных точек: Pdst (точки искажённого изображения) и Psrc (исходные точки). С помощью cv.getAffineTransform() вычисляется обратная матрица трансформации, которая применяется к изображению с использованием cv.warpAffine(). В результате изображение приводится к исходной форме, однако некоторые области могут оказаться чёрными из-за потери данных при обратном преобразовании.

```
Psrc = np.float32([[50, 300], [150, 200], [50, 50]])
Pdst = np.float32([[50, 200], [250, 200], [50, 100]])
T = cv.getAffineTransform(Psrc, Pdst)
Iaffine = cv.warpAffine(I, T, I.shape[:2][::-1])

# Draw triangles on source and transformed images
Icopy = I.copy()
Iaffine_copy = Iaffine.copy()
cv.polylines(Icopy, [Psrc.astype(np.int32)], True, (0, 0, 0), 1)
cv.polylines(Iaffine_copy, [Pdst.astype(np.int32)], True, (0, 0, 0), 1)

# Display it
ShowImages([("Source image", Icopy), ("Affine mapping", Iaffine_copy)], 2)
```

Скос изображения.

В данном задании выполняется трансформация скоса (bevel transformation) изображения с коэффициентом сдвига s=0.3; для этого создаётся матрица преобразования, в которой параметр s отвечает за горизонтальное смещение

строк изображения в зависимости от их высоты. Затем с помощью функции cv.warpAffine() применяется аффинное преобразование, изменяющее форму изображения, создавая эффект наклона. Поскольку изображение становится шире, рассчитываются новые размеры, чтобы избежать обрезки. Итоговое изображение выводится на экран.

```
s = 0.3  # Коэффициент сдвига

# Создаем матрицу преобразования для скоса
M = np.float32([[1, s, 0], [0, 1, 0]])

# Определяем новые размеры изображения после трансформации
new_w = int(I.shape[1] + s * I.shape[0])
new_h = I.shape[0]

# Применяем аффинное преобразование (warpAffine)
skewed_image = cv.warpAffine(I, M, (new_w, new_h))

ShowImages([("Скошенное изображение", skewed_image)])
```

Кусочно-линейное отображение

В этом задании выполняется частично-линейное масштабирование, при котором правая половина изображения уменьшается вдвое. Для этого создаётся аффинная матрица преобразования, изменяющая X-координаты пикселей в правой части изображения. Затем с помощью cv.warpAffine() это преобразование применяется только к правой половине, в то время как левая часть остаётся неизменной. В результате правая сторона изображения сжимается, сохраняя общую структуру изображения.

```
def piecewise_linear_mapping(img, scale_factor):
    rows, cols = img.shape[:2]
    t = np.float32([[scale_factor, 0, 0], [0, 1, 0]])
    img[:, int(cols / 2):] = cv.warpAffine(img[:, int(cols / 2):], t, (cols -
int(cols / 2), rows))
    return img

piecewiselinear2 = piecewise_linear_mapping(I.copy(), 2)  # Увеличение в 2 раза
piecewiselinear05 = piecewise_linear_mapping(I.copy(), 0.5)  # Уменьшение в 2
раза
```

Нелинейные трансформации

Проекционное отображение

В этом задании выполняется обратное проекционное преобразование, восстанавливающее исходное изображение из его искажённой версии. Для этого используются две группы контрольных точек: Pdst (точки искажённого изображения) и Psrc (исходные точки). Функция cv.getPerspectiveTransform() вычисляет матрицу обратного преобразования, которая затем применяется к изображению с помощью cv.warpPerspective(). В результате изображение приближается к исходному виду, однако некоторые области могут остаться чёрными из-за потери данных при обратном преобразовании.

```
def projection(img, arr):
    rows, cols = img.shape[:2]
    t = np.float32(arr)  # Преобразуем массив в float32
    return cv.warpPerspective(img, t, (cols, rows))

# Применяем перспективное преобразование
T = [[1.1, 0.35, 0], [0.2, 1.1, 0], [0.00075, 0.0005, 1]]
Ipersp = projection(I, T)
```

Полиномиальное отображение

В данном задании выполняется **полиномиальное преобразование координат** пикселей изображения.
Для каждой точки вычисляются новые координаты по заданным полиномиальным формулам, где:

- **Горизонтальная координата** остаётся неизменной
- **Вертикальная координата** модифицируется с нелинейными искажениями

После вычисления новых координат они ограничиваются в пределах изображения с помощью np.clip(), а затем применяется cv.remap() для выполнения трансформации. В результате изображение приобретает нелинейные искажения, заметные в вертикальном направлении.

Синусоидальное искажение

В данном задании выполняется синусоидальное искажение вдоль оси OyOy.
Для каждой точки вычисляется новая вертикальная координата.
Функция `cv.remap()` применяет эти новые координаты, создавая волнообразное смещение вдоль вертикальной оси.

```
def sinusoidal_distortion_oy(img):
    rows, cols = img.shape[:2]
    x, y = np.meshgrid(np.arange(cols), np.arange(rows))
    y = y + 20 * np.sin(2 * np.pi * x / 90)  # Амплитуда 20, период 90 пикселей
    y = np.clip(y, 0, rows - 1).astype(np.float32)
    Isin_y = cv.remap(img, x.astype(np.float32), y,
interpolation=cv.INTER_LINEAR)
```

```
    return Isin_y

I_sin_y = sinusoidal_distortion_oy(I)
```

Задание 2. Коррекция дисторсии

Бочкообразная дисторсия.

В этом задании выполняется бочкообразная дисторсия изображения. Сначала координаты нормализуются относительно центра, затем переводятся в полярную систему. Затем координаты преобразуются обратно в декартовую систему, и корректированное изображение формируется с помощью cv.remap(), создавая эффект бочкообразного искажения.

```
x, y = np.meshgrid(np.arange(I.shape[1]), np.arange(I.shape[0]))

mid = I.shape[1] / 2.0, I.shape[0] / 2.0
x, y = x - mid[0], y - mid[1]

r, theta = cv.cartToPolar(x / mid[0], y / mid[1])
b0 = 1
F3 = 0.1
F5 = 0.12
r = b0 * r + F3 * r ** 3 + F5 * r ** 5
u, v = cv.polarToCart(r, theta)
u = u * mid[0] + mid[0]
v = v * mid[1] + mid[1]

Ibarrel = cv.remap(I, u.astype(np.float32), v.astype(np.float32),
cv.INTER_LINEAR)
```

Подушкообразная дисторсия

В этом задании выполняется **подушкообразная дисторсия** изображения. Координаты пикселей сначала нормализуются относительно центра и переводятся в полярную систему. Затем радиус изменяется так, что края изображения **втягиваются внутрь**, создавая характерный эффект. После этого координаты преобразуются обратно в декартовую систему, и изображение корректируется с помощью cv.remap().

```
x, y = np.meshgrid(np.arange(I.shape[1]), np.arange(I.shape[0]))

mid = I.shape[1] / 2.0, I.shape[0] / 2.0
x, y = x - mid[0], y - mid[1]

r, theta = cv.cartToPolar(x / mid[0], y / mid[1])
b0 = 1.2
F3 = -0.15
```

```
F5 = 0
r = b0 * r + F3 * r ** 3 + F5 * r ** 5
u, v = cv.polarToCart(r, theta)
u = u * mid[0] + mid[0]
v = v * mid[1] + mid[1]

Ipincushion = cv.remap(I, u.astype(np.float32), v.astype(np.float32),
cv.INTER_LINEAR)
```

Коррекция бочкообразной дисторсии

В этом задании выполняется коррекция бочкообразной дисторсии. Сначала координаты пикселей нормализуются относительно центра изображения и переводятся в полярную систему. Затем к радиусу применяется обратное искажение, компенсирующее эффект выпуклого искажения краёв. После этого координаты преобразуются обратно в декартовую систему, и скорректированное изображение формируется с помощью cv.remap().

```
def fix_pincushion_distortion(image):
    height, width = image.shape[:2]
    x_grid, y_grid = np.meshgrid(np.arange(width), np.arange(height))

    x_center = width / 2.0
    y_center = height / 2.0

    x_grid -= x_center
    y_grid -= y_center

    radius, angle = cv.cartToPolar(x_grid / x_center, y_grid / y_center)

    correction_factor_1 = 1 / 1.3  # Инверсный множитель
    correction_factor_2 = 0.2  # Инверсный коэффициент r^2
    correction_factor_3 = -0.05  # Инверсный коэффициент r^4

    # Исправление радиуса
    radius_corrected = correction_factor_1 * radius + correction_factor_2 *
radius**3 + correction_factor_3 * radius**5

    # Возвращаемся в декартовы координаты
    corrected_x, corrected_y = cv.polarToCart(radius_corrected, angle)
    corrected_x = corrected_x * x_center + x_center
    corrected_y = corrected_y * y_center + y_center

    # Применяем исправление с помощью remap
    corrected_image = cv.remap(image, corrected_x.astype(np.float32),
corrected_y.astype(np.float32), cv.INTER_LINEAR)

    return corrected_image
```

```
# Применяем коррекцию подушкообразной дисторсии
I_fixed_pincushion = fix_pincushion_distortion(I)
```

Коррекция подушкообразной дисторсии

Этот код сначала создаёт подушкообразную дисторсию, при которой края изображения втягиваются внутрь, а затем выполняет коррекцию. Деформация и её обратное преобразование выполняются через полярные координаты с использованием cv.remap().

```
def correct_pincushion(image):
    height, width = image.shape[:2]
    x_grid, y_grid = np.meshgrid(np.arange(width, dtype=np.float32),
np.arange(height, dtype=np.float32))

    x_center = width / 2.0
    y_center = height / 2.0

    x_grid -= x_center
    y_grid -= y_center

    r, theta = cv.cartToPolar(x_grid / x_center, y_grid / y_center)

    # Обратная коррекция (инверсия искажения)
    r_corrected = (r + 0.2 * r**2 - 0.05 * r**4) / 1.3

    u, v = cv.polarToCart(r_corrected, theta)
    u = u * x_center + x_center
    v = v * y_center + y_center

    corrected_img = cv.remap(image, u, v, interpolation=cv.INTER_LINEAR)

    return corrected_img

I_distorted = distort_pincushion(I)
I_corrected = correct_pincushion(I_distorted)
```

Задание 3. Склейка изображений

Склейка изображения

Этот код **разделяет изображение пополам**, затем **склеивает его обратно** с помощью np.vstack(). После этого выполняется **проверка идентичности** полученного изображения с исходным с использованием cv.absdiff(), который вычисляет разницу между пикселями. Если разница равна нулю по всем пикселям, изображения считаются **полностью совпадающими**.

```
def split_and_stitch(image):
```

```
    height = image.shape[0]

    # Разделяем изображение на две равные части
    Itop = image[:height // 2, :, :]
    Ibottom = image[height // 2:, :, :]

    # Склеиваем обратно
    stitched_image = np.vstack((Itop, Ibottom))

    return stitched_image

I = cv.imread("images/cat.png")
I_stitched = split_and_stitch(I)
ShowImages([
    ("Оригинальное изображение", I),
    ("Склеенное изображение", I_stitched)
])

difference = cv.absdiff(I, I_stitched)
if np.all(difference == 0):
    print("Изображения полностью совпадают!")
else:
    print("Изображения отличаются.")
```

Результирующие изображения:

Рисунок 2 - Задание 1, сдвиг изображения

Рисунок 3 - Задание 1, отражение изображения

Рисунок 4 - Задание 1, поворот изображения

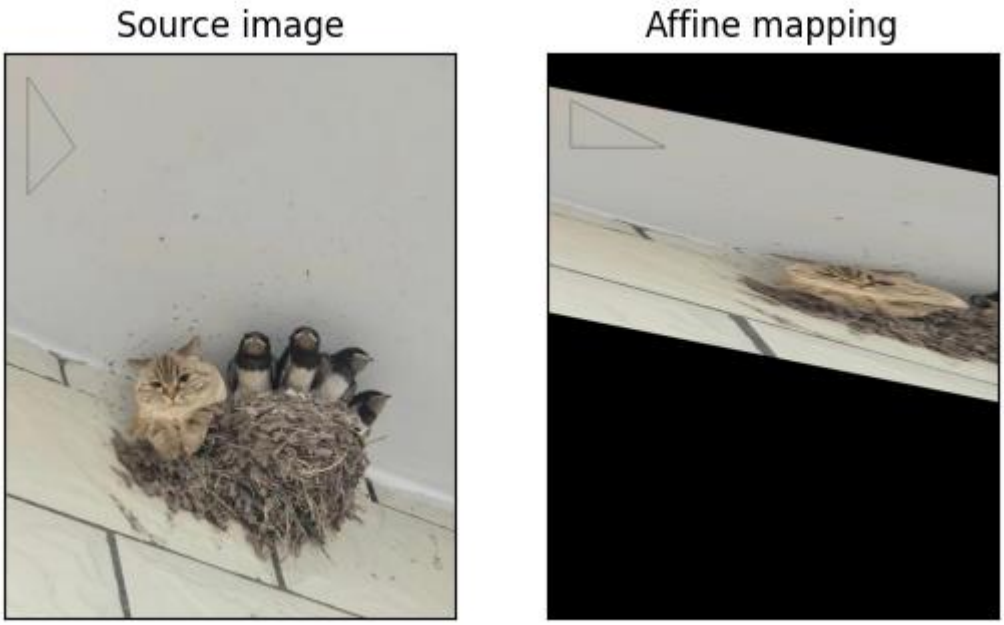

Рисунок 5 - Задание 1, обратное аффинное преобразование

Рисунок 6 - Задание 1, скос изображения

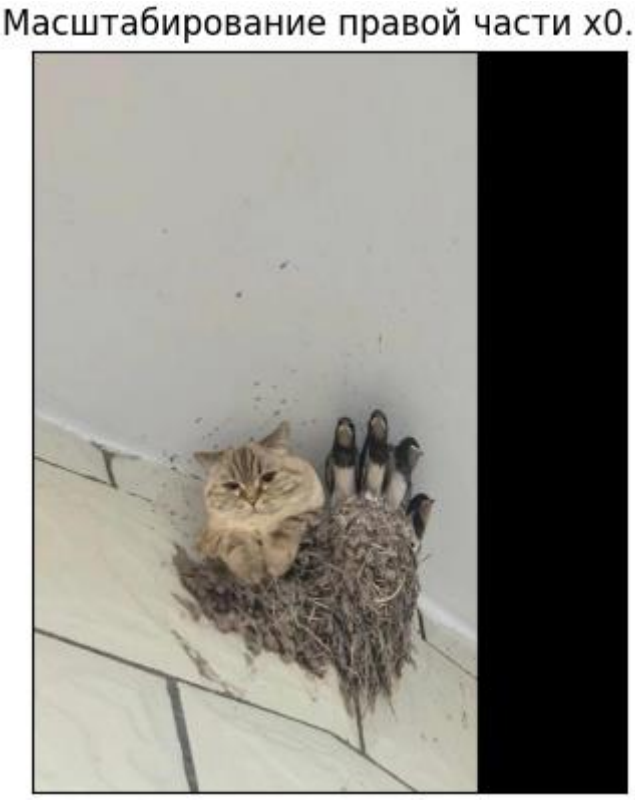

Рисунок 7 - Задание 1, кусочно-линейное отображение

Рисунок 8 - Задание 2, проекционное отображение

Рисунок 9 - Задание 1, полиномиальное отображение

Рисунок 10 - Задание 2, синусоидальное отображение

Рисунок 11 - Задание 2, бочкообразная дисторсия

Рисунок 12 - Задание 2, подушкообразная дисторсия

Рисунок 13 - Задание 2, коррекция бочкообразной дисторсии

Рисунок 14 - Задание 2, коррекция подушкообразной дисторсии

Рисунок 15 - Задание 3, склейка изображения

Выводы о проделанной работе:

В ходе работы были изучены и реализованы различные геометрические преобразования изображений, включая сдвиг, масштабирование, поворот, скос, перспективные преобразования и коррекцию дисторсий. Также была выполнена склейка изображений, сначала путём разбиения изображения на части, а затем их объединения. В результате освоены ключевые техники обработки изображений в OpenCV, а также методы корректировки и анализа результатов.