

Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный Исследовательский Университет ИТМО»

**ЛАБОРАТОРНАЯ РАБОТА №2
ПО ПРЕДМЕТУ «ТЕХНИЧЕСКОЕ ЗРЕНИЕ»
ПО ТЕМЕ «ГЕОМЕТРИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ ИЗОБРАЖЕНИЙ»**

Преподаватель:
Шаветов С. В.

Выполнили:
Румянцев А. А.
Овчинников П. А.
Чебаненко Д. А.

Факультет: СУиР
Группа: R3241
Поток: ТЕХ.ЗРЕНИЕ 2.1

Санкт-Петербург
2024

Содержание

1 Начало	2
1.1 Цель	2
1.2 Немного о преобразовании изображений	2
1.3 Оригинальная картинка	2
1.4 Язык программирования и необходимая библиотека	2
2 Простейшие геометрические преобразования	3
2.1 Сдвиг изображения	3
2.2 Отражение изображения	3
2.3 Однородное масштабирование изображения	4
2.4 Поворот изображения	5
2.5 Скос изображения	6
2.6 Кусочно-линейное отображение	7
2.7 Проекционное отображение	8
2.8 Полиномиальное отображение	9
2.9 Синусоидальное искажение	10
2.10 Бочкообразная дисторсия	11
2.11 Подушкообразная дисторсия	11
3 Коррекция дисторсии	13
3.1 Коррекция бочкообразной дисторсии	13
3.2 Коррекция подушкообразной дисторсии	14
4 «Склейка» изображений	15
4.1 Ручная склейка изображения	15
4.2 Автоматическая склейка изображения	16
5 Ответы на вопросы	17
5.1 Каким образом можно выполнить поворот изображения, не используя матрицу поворота?	17
5.2 Какое минимальное количество соответствующих пар точек необходимо задать на исходном и искаженном изображениях, если порядок преобразования $n = 4$?	17
5.3 После геометрического преобразования изображения могут появиться пиксели с неопределенными значениями интенсивности. С чем это связано и как решается данная проблема?	17
6 Выводы	18
6.1 Лабораторная работа 2	18

1 Начало

1.1 Цель

Цель выполнения данной работы заключается в освоении основных видов отображений и использовании геометрических преобразований для решения задач пространственной коррекции изображений

1.2 Немного о преобразовании изображений

Геометрические преобразования являются матричными преобразованиями. Множества координат пикселей преобразованного и исходного изображений связаны следующим матричным соотношением:

$$X' = XT \Rightarrow \begin{bmatrix} x' \\ y' \\ \omega' \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \omega \end{bmatrix},$$

где $X = (x, y, \omega)$ – представление точки $X = (x, y)$ на плоскости в двумерном пространстве P^2 , ω – скалярный произвольный множитель, $x = x'/\omega$, $y = y'/\omega$. Точка с декартовыми координатами (x, y) в однородных координатах примет вид $(x, y, 1)$. Тогда:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

что можно записать в виде следующей системы:

$$\begin{cases} x' = Ax + By + C \\ y' = Dx + Ey + F \end{cases}$$

1.3 Оригинальная картинка

Для всех исследуемых преобразований была выбрана следующая картинка:



Рис. 1: Оригинальная картинка

1.4 Язык программирования и необходимая библиотека

Работа выполнена на языке программирования Python с использованием библиотеки OpenCV. Следующий код позволяет считать картинку в переменную и после сохранить изображение в заданную папку:

```

1 import cv2
2
3 path = 'tech-vision/source/img1.jpg'
4 render_dir = 'tech-vision/render/'
```

```

5
6 source_img = cv2.imread(path)
7 cv2.imwrite(f'{render_dir}cats.png', source_img)

```

Листинг 1: Пример использования библиотеки OpenCV на python

На выходе получим картинку, приведенную на рисунке 1. В дальнейшем по умолчанию будут использоваться переменные path, render_dir и source_img, но для экономии места каждый раз они приводиться не будут. Также подключим библиотеку numpy для работы с матрицами преобразования

2 Простейшие геометрические преобразования

2.1 Сдвиг изображения

Сдвиг изображения – это такое преобразование изображения, при котором все точки картинки перемещаются в заданном направлении на заданное расстояние. Это преобразование можно описать матрицей преобразования координат T , в которой в случае сдвига изображения $A = E = 1$, $B = D = 0$. Ранее описанная система и матрица T примут следующий вид:

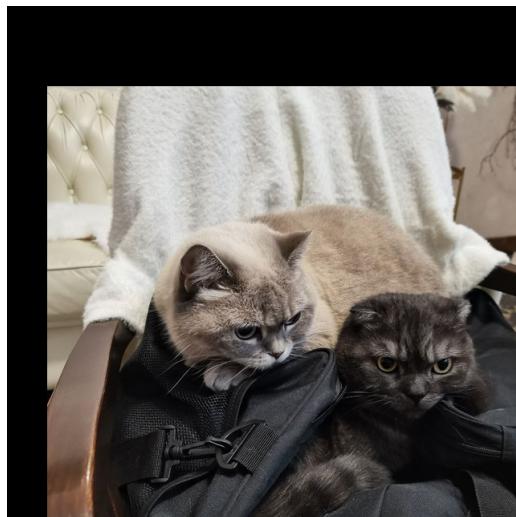
$$\begin{cases} x' = x + C \\ y' = y + D \end{cases} \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ C & D & 1 \end{bmatrix},$$

где C и D – сдвиг по осям Ox и Oy соответственно

```

1 import numpy as np
2
3 def shift_img(img, delta_x, delta_y):
4     rows, cols = img.shape[0:2]
5     t = np.float32([[1, 0, delta_x], [0, 1, delta_y]])
6     return cv2.warpAffine(img, t, (cols, rows))
7
8 shifted_img = shift_img(source_img, delta_x=50, delta_y=100)
9 cv2.imwrite(f'{render_dir}shifted_img.png', shifted_img)

```

Листинг 2: Код для сдвига изображения на 50 и 100 пикселей по осям Ox и Oy Рис. 2: Сдвиг изображения на 50 и 100 пикселей по осям Ox и Oy соответственно

Как видим, изображение съехало больше по вертикали, чем по горизонтали, значит все в порядке и оси координат не были перепутаны и написанный алгоритм выполнил свою задачу

2.2 Отражение изображения

Отражение изображения – преобразование картинки, при котором каждый горизонтальный ряд пикселей или каждый вертикальный ряд пикселей задается в обратном порядке относительно оригинального изображения. Система уравнений из пункта 1.2 и матрица преобразования координат T в случае отображения

изображения вдоль оси Ox при $A = 1$, $E = -1$, $B = C = D = F = 0$ примут вид:

$$\begin{cases} x' = x \\ y' = -y \end{cases} \Rightarrow T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Код ниже позволяет отражать изображение относительно оси Ox , если параметр `flag==true`, в противном случае относительно оси Oy . Воспользуемся им для получения двух новых картинок:

```

1 def reflect_img(img, flag: bool):
2     rows, cols = img.shape[0:2]
3     if (flag):
4         t = np.float32([[1, 0, 0], [0, -1, rows - 1]])
5     else:
6         t = np.float32([[-1, 0, rows - 1], [0, 1, 0]])
7     return cv2.warpAffine(img, t, (cols, rows))
8
9 reflected_img_ox = reflect_img(source_img, flag=True)
10 reflected_img_oy= reflect_img(source_img, flag=False)
11 cv2.imwrite(f'{render_dir}reflect_img_ox.png', reflected_img_ox)
12 cv2.imwrite(f'{render_dir}reflect_img_oy.png', reflected_img_oy)
```

Листинг 3: Код для отражения относительно оси Ox или Oy



Рис. 3: Отражение изображения относительно оси Ox



Рис. 4: Отражение изображения относительно оси Oy

2.3 Однородное масштабирование изображения

Однородное масштабирование изображения – такое матричное преобразование картинки, при котором размеры изображения меняются с помощью увеличения или уменьшения его размеров на одинаковый коэффициент по всем направлениям. Система уравнений из пункта 1.2 и матрица преобразования координат T в данном случае $A = \alpha$, $E = \beta$, $B = C = D = F = 0$ примут вид:

$$\begin{cases} x' = \alpha x, \alpha > 0 \\ y' = \beta y, \beta > 0 \end{cases} \Rightarrow T = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Если $\alpha < 1$ и $\beta < 1$, то изображение уменьшается, если $\alpha > 1$ и $\beta > 1$, то увеличивается. Если $\alpha \neq \beta$, то пропорции будут не одинаковыми по ширине и высоте

В случае использования библиотеки OpenCV из матрицы T пропадает последняя строчка. Воспользуемся следующим кодом для масштабирования изображения:

```

1 def scale_img(img, scale_x, scale_y):
2     rows, cols = img.shape[0:2]
3     t = np.float32([[scale_x, 0, 0], [0, scale_y, 0]])
4     return cv2.warpAffine(img, t, (int(cols * scale_x), int(rows * scale_y)))
```

```

6 scaled_img_bigger = scale_img(source_img, 2, 2)
7 scaled_img_smaller = scale_img(source_img, 0.5, 0.5)
8 scaled_img_wtf = scale_img(source_img, 1, 0.5)
9 cv2.imwrite(f'{render_dir}scaled_img_bigger.png', scaled_img_bigger)
10 cv2.imwrite(f'{render_dir}scaled_img_smaller.png', scaled_img_smaller)
11 cv2.imwrite(f'{render_dir}scaled_img_wtf.png', scaled_img_wtf)

```

Листинг 4: Код для масштабирования изображения



Рис. 5: Увеличенное в два раза изображение

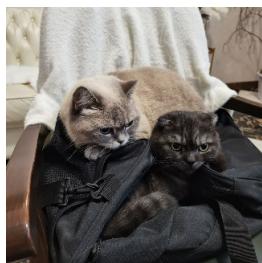


Рис. 6: Уменьшенное в два раза изображение

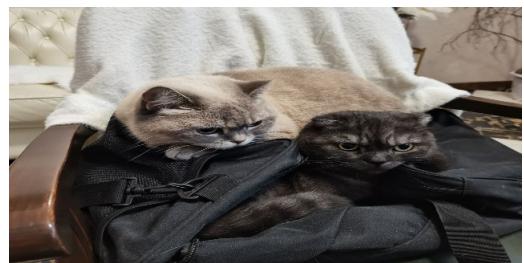


Рис. 7: Уменьшенное по высоте изображение

Кошки побольше, кошки поменьше и широкие кошки. Красота. А главное – работает

2.4 Поворот изображения

Поворот изображения – преобразование изображения, при котором картинка вращается вокруг некоторой точки или оси на заданный угол. Система уравнений пункта 1.2 и матрица T в случае поворота по часовой

стрелке $A = \cos \phi$, $B = -\sin \phi$, $D = \sin \phi$, $E = \cos \phi$, $C = F = 0$ примут вид:

$$\begin{cases} x' = x \cos \phi - y \sin \phi \\ y' = x \sin \phi + y \cos \phi \end{cases} \Rightarrow T = \begin{bmatrix} \cos \phi & \sin \phi & 0 \\ -\sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

В случае использования библиотеки OpenCV из матрицы T пропадает последняя строчка. Рассмотрим код ниже. Метод `rotate_img` поворачивает изображение относительно левого верхнего угла картинки, а метод `rotate_img_center` относительно центра:

```

1 def rotate_img(img, angle):
2     rows, cols = img.shape[0:2]
3     phi = np.radians(angle)
4     t = np.float32([[np.cos(phi), -np.sin(phi), 0], [np.sin(phi), np.cos(phi), 0]])
5     return cv2.warpAffine(img, t, (cols, rows))
6
7 def rotate_img_center(img, angle):
8     rows, cols = img.shape[0:2]
9     phi = np.radians(angle)
10    t1 = np.float32([[1, 0, -(cols - 1) / 2], [0, 1, -(rows - 1) / 2], [0, 0, 1]])
11    t2 = np.float32([[np.cos(phi), -np.sin(phi), 0], [np.sin(phi), np.cos(phi), 0], [0, 0, 1]])
12    t3 = np.float32([[1, 0, (cols - 1) / 2], [0, 1, (rows - 1) / 2], [0, 0, 1]])
13    t = np.matmul(t3, np.matmul(t2, t1))[0:2, :]
14    return cv2.warpAffine(img, t, (cols, rows))
15
16 rotated_img = rotate_img(source_img, angle=45)
17 cv2.imwrite(f'{render_dir}rotated_img.png', rotated_img)
18
19 rotated_img_center = rotate_img_center(source_img, angle=45)
20 cv2.imwrite(f'{render_dir}rotated_img_center.png', rotated_img_center)

```

Листинг 5: Код для поворота изображения



Рис. 8: Поворот изображения относительно левого угла на 45°



Рис. 9: Поворот изображения относительно центра на 45°

2.5 Скос изображения

Скос изображения – такое преобразование картинки, при котором происходит смещение всех точек вдоль одной из осей на заданное расстояние, причем стороны остаются параллельными, но углы между ними меняются. Система уравнений пункта 1.2 и матрица T в данном случае $A = E = 1$, $B = s$, $C = D = F = 0$ примут вид:

$$\begin{cases} x' = x + sy \\ y' = y \end{cases} \Rightarrow T = \begin{bmatrix} 1 & 0 & 0 \\ s & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Скос изображения с помощью кода на python с $s = 0.3$:

```

1 def bevel_img(img, arg):
2     rows, cols = img.shape[0:2]
3     t = np.float32([[1, arg, 0], [0, 1, 0]])
4     return cv2.warpAffine(img, t, (cols, rows))
5
6 beveled_img = bevel_img(source_img, 0.3)
7 cv2.imwrite(f'{render_dir}beveled_img.png', beveled_img)

```

Листинг 6: Код для скоса изображения



Рис. 10: Скос изображения

2.6 Кусочно-линейное отображение

Кусочно-линейное отображение – это отображение, при котором картинка разбивается на части, а после к каждой части применяются различные линейные преобразования

Следующий код растягивает правую часть изображения вдоль оси Ox , а левую оставляет без изменений

```

1 def piecewise_linear_mapping(img, arg):
2     rows, cols = img.shape[0:2]
3     t = np.float32([[arg, 0, 0], [0, 1, 0]])
4     img[:, int(cols / 2):, :] = cv2.warpAffine(img[:, int(cols / 2):, :], t, (cols - int(cols / 2), rows))
5     return img
6
7 piecewiselinear2 = piecewise_linear_mapping(source_img, 2)
8 piecewiselinear05 = piecewise_linear_mapping(source_img, 0.5)
9 cv2.imwrite(f'{render_dir}piecewiselinear2.png', piecewiselinear2)
10 cv2.imwrite(f'{render_dir}piecewiselinear05.png', piecewiselinear05)

```

Листинг 7: Код для изменения правой части изображения по оси Ox



Рис. 11: Растижение правой части изображения вдоль оси Ox в два раза



Рис. 12: Сужение правой части изображения вдоль оси Ox в два раза

2.7 Проекционное отображение

Проекционное отображение – это отображение, при котором прямые линии остаются прямыми линиями, однако геометрия фигуры может быть нарушена, так как данное отображение в общем случае не сохраняет параллельности линий. Однако три точки, лежащие на одной прямой (коллинеарные), после преобразования остаются на одной прямой. Проекционное отображение может быть как параллельным (изменение масштаба), так и проективными (изменение геометрии фигуры). Система и матрица преобразования T в таком случае примут вид:

$$\begin{cases} x' = \frac{Ax + By + C}{Gx + Hy + I} \\ y' = \frac{Dx + Ey + F}{Gx + Hy + I} \end{cases} \Rightarrow T = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & 1 \end{bmatrix}$$

Пример кода, где $A = 1.1$, $B = 0.35$, $C = F = 0$, $D = 0.2$, $E = 1.1$, $G = 0.00075$, $H = 0.0005$, $I = 1$

```

1 def projection(img, arr):
2     rows, cols = img.shape[0:2]
3     t = np.float32(arr)
4     return cv2.warpPerspective(img, t, (cols, rows))
5
6 projected_img = projection(source_img, [[1.1, 0.35, 0], [0.2, 1.1, 0], [0.00075, 0.0005, 1]])
7 cv2.imwrite(f'{render_dir}projected_img.png', projected_img)

```

Листинг 8: Код для проективного отображения

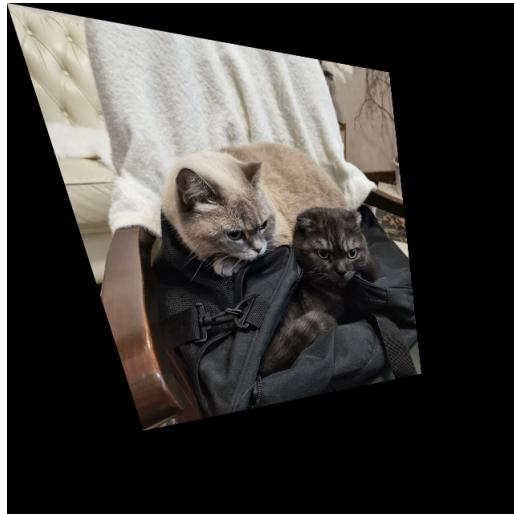


Рис. 13: Проективное отображение

2.8 Полиномиальное отображение

Полиномиальное отображение – название говорит само за себя – это отображение исходного изображения с помощью полиномов. Матрица T будет содержать коэффициенты полиномов соответствующих порядков для координат x и y . В случае полиномиального преобразования второго порядка система пункта 1.2 примет вид:

$$\begin{cases} x' = a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2 \\ y' = b_1 + b_2x + b_3y + b_4x^2 + b_5xy + b_6y^2 \end{cases},$$

где x, y – координаты точек в одной системе координат, x', y' – координаты этих точек в другой системе координат, $a_1 \dots a_6, b_1 \dots b_6$ – коэффициенты преобразования

Результирующее образованное кодом ниже изображение не будет содержать интерполированных пикселей:

```

1 def polynomial(img, t):
2     rows, cols = img.shape[0:2]
3     t = np.array(t)
4     I_polynomial = np.zeros(img.shape, img.dtype)
5
6     x, y = np.meshgrid(np.arange(cols),
7                         np.arange(rows))
8     xnew = np.round(t[0, 0] + x * t[1, 0] +
9                      y * t[2, 0] + x * x * t[3, 0] +
10                     x * y * t[4, 0] +
11                     y * y * t[5, 0]).astype(np.float32)
12     ynew = np.round(t[0, 1] + x * t[1, 1] +
13                      y * t[2, 1] + x * x * t[3, 1] +
14                     x * y * t[4, 1] +
15                     y * y * t[5, 1]).astype(np.float32)
16     mask = np.logical_and(
17         np.logical_and(xnew >= 0, xnew < cols),
18         np.logical_and(ynew >= 0, ynew < rows))
19     if img.ndim == 2:
20
21         I_polynomial[ynew[mask].astype(int),
22
23             xnew[mask].astype(int)] = \
24             img[y[mask], x[mask]]
25     else:
26         I_polynomial[ynew[mask].astype(int),
27             xnew[mask].astype(int), :] = \
28             img[y[mask], x[mask], :]
29     return I_polynomial
30
31 T = [[0, 0], [1, 0], [0, 1], [0.00001, 0], [0.002, 0], [0.001, 0]]
32 m = polynomial(source_img, T)
33 cv2.imwrite(f'{render_dir}polynomial.png', m)

```

Листинг 9: Код для полиномиального отображения

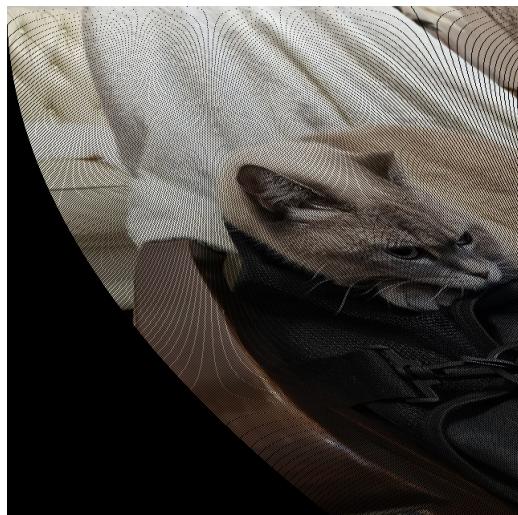


Рис. 14: Полиномиальное отображение

2.9 Синусоидальное искажение

Синусоидальное искажение – гармоническое искажение картинки, при котором пиксели перемещаются вверх и вниз или влево и вправо в зависимости от позиции на изображении, что создает эффект волнистости или искривления

```

1 def sinusoidal(img):
2     rows, cols = img.shape[0:2]
3     u, v = np.meshgrid(np.arange(cols), np.arange(rows))
4     u = u + 20 * np.sin(2 * np.pi * v / 90)
5     img_sinusoid = cv2.remap(img, u.astype(np.float32), v.astype(np.float32), cv2.INTER_LINEAR
6     )
7     return img_sinusoid
8
9 sinusoid = sinusoidal(source_img)
cv2.imwrite(f'{render_dir}sinusoid.png', sinusoid)

```

Листинг 10: Код для синусоидального искажения

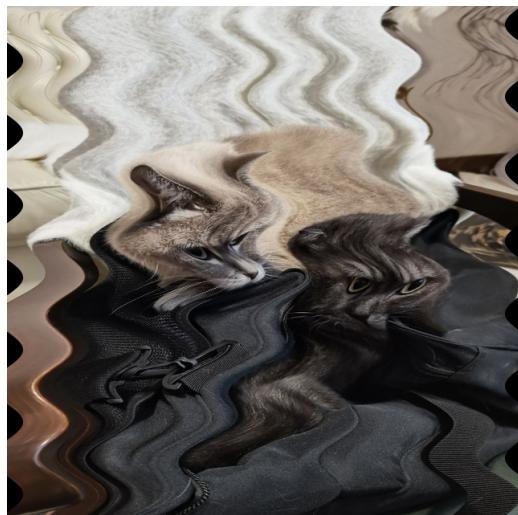


Рис. 15: Синусоидальное отображение

2.10 Бочкообразная дисторсия

Бочкообразная дисторсия – отрицательная дисторсия – оптическое искажение картинки, при котором объекты, находящиеся ближе к краям, становятся сжатыми в направлении центра. Пусть $\mathbf{r} = (x, y)$ – вектор, задающий две координаты в плоскости, расположенной перпендикулярно оптической оси. Все лучи, вышедшие из этой точки и прошедшие через оптическую систему, попадут в точку изображения с координатами \mathbf{R} , которые в общем виде определяются по формуле:

$$\mathbf{R} = b_0 \mathbf{r} + F_3 r^2 \mathbf{r} + F_5 r^4 \mathbf{r} + \dots,$$

где r – длина вектора \mathbf{r} ; $F_i, i = 3, 5, \dots, n$ – коэффициенты дисторсии n -ого порядка, которые вносят наибольший вклад в искажение формы изображения

```

1 def barrel_distortion(img):
2     rows, cols = img.shape[0:2]
3     xi, yi = np.meshgrid(np.arange(cols),
4                           np.arange(rows))
5     xmjd = cols / 2.0
6     ymid = rows / 2.0
7     xi = xi - xmjd
8     yi = yi - ymid
9     r, theta = cv2.cartToPolar(xi / xmjd, yi / ymid)
10    F3 = 0.1
11    F5 = 0.12
12    r = r + F3 * r ** 3 + F5 * r ** 5
13    u, v = cv2.polarToCart(r, theta)
14    u = u * xmjd + xmjd
15    v = v * ymid + ymid
16    I_barrel = \
17        cv2.remap(img, u.astype(np.float32),
18                   v.astype(np.float32), cv2.INTER_LINEAR)
19    return I_barrel
20
21 barreled = barrel_distortion(source_img)
22 cv2.imwrite(f'{render_dir}barrel.png', barreled)

```

Листинг 11: Код для бочкообразной дисторсии



Рис. 16: Бочкообразная дисторсия

2.11 Подушкообразная дисторсия

Подушкообразная дисторсия – положительная дисторсия – оптическое искажение картинки, при котором объекты, находящиеся ближе к центру изображения, кажутся уплощенными, а объекты, находящиеся ближе к краям, кажутся выпуклыми или "надутыми". Формулы такие же, как в пункте 2.10

```

1 def pincushion_distortion(img):
2     rows, cols = img.shape[0:2]
3     xi, yi = np.meshgrid(np.arange(cols),
4                           np.arange(rows))
5     xmid = cols / 2.0
6     ymid = rows / 2.0
7     xi = xi - xmid
8     yi = yi - ymid
9     r, theta = cv2.cartToPolar(xi / xmid, yi / ymid)
10    F3 = -0.1
11    F5 = -0.12
12    r = r + F3 * r ** 3 + F5 * r ** 5
13    u, v = cv2.polarToCart(r, theta)
14    u = u * xmid + xmid
15    v = v * ymid + ymid
16    I_barrel = \
17        cv2.remap(img, u.astype(np.float32),
18                  v.astype(np.float32), cv2.INTER_LINEAR)
19
20
21 p = pincushion_distortion(source_img)
22 cv2.imwrite(f'{render_dir}pil_distort.png', p)

```

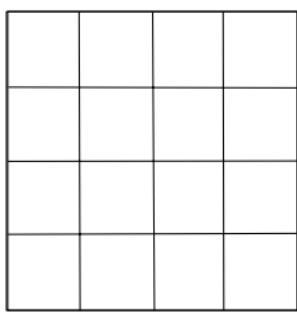
Листинг 12: Код для подушкообразной дисторсии



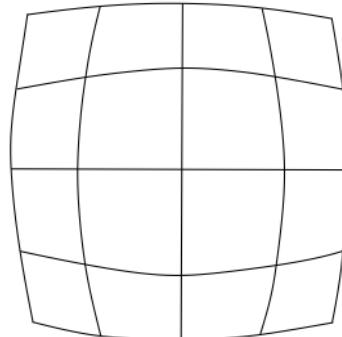
Рис. 17: Подушкообразная дисторсия

3 Коррекция дисторсии

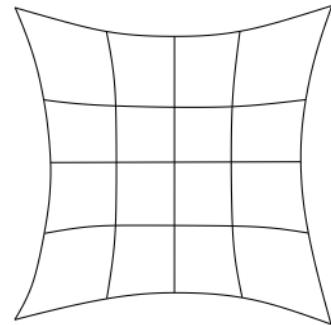
Для коррекции дисторсии используется подход для проективного отображения. Используется изображение регулярной сетки и его искаженное изображение, находятся пары точек на этих изображениях и вычисляются коэффициенты корректирующего преобразования. Возьмем два искаженных изображения и его оригинал:



(a) Оригинальное изображение



(b) Бочкообразная дисторсия



(c) Подушкообразная дисторсия

Рис. 18: Изображения для последующей коррекции и сравнения

3.1 Коррекция бочкообразной дисторсии

Попробуем починить бочку подушкой этой программой:

```

1 def undistort_barrel(img):
2     rows, cols = img.shape[0:2]
3     xi, yi = np.meshgrid(np.arange(cols), np.arange(rows))
4     xmid = cols / 2.0
5     ymid = rows / 2.0
6     xi = xi - xmid
7     yi = yi - ymid
8     r, theta = cv2.cartToPolar(xi / xmid, yi / ymid)
9
10    F3 = 0.1
11    F5 = 0.06
12
13    r = r - F3 * r ** 3 - F5 * r ** 5 # Apply the inverse distortion
14    u, v = cv2.polarToCart(r, theta)
15    u = u * xmid + xmid
16    v = v * ymid + ymid
17
18    I_undistorted = cv2.remap(img, u.astype(np.float32), v.astype(np.float32), cv2.
19                                INTER_LINEAR,
20                                borderMode=cv2.BORDER_CONSTANT, borderValue=(255, 255, 255))
21
22    return I_undistorted
23
24 barreled = barrel_distortion(source_img)
25 undistort_bar = undistort_barrel(barreled)
26 cv2.imwrite(f'{render_dir}undistort_bar.png', undistort_bar)

```

Листинг 13: Код для коррекции бочкообразной дисторсии

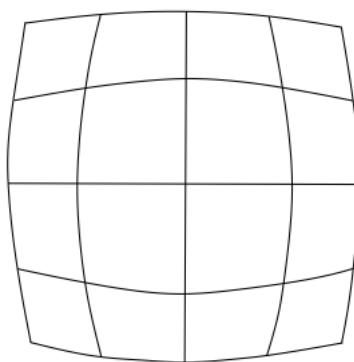


Рис. 19: Бочкообразная дисторсия

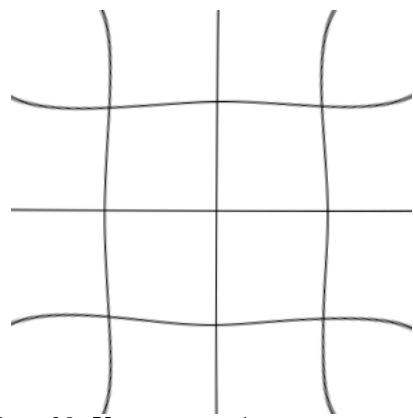


Рис. 20: Коррекция бочки подушкой

Потеряли контур изображения, линии получились извилистые, однако мы получили что-то отдаленно напоминающее оригинал

3.2 Коррекция подушкообразной дисторсии

Сыграем в бочку, подушку и оригинальное изображение следующим кодом и узнаем, бьет ли бочка подушку:

```

1 def undistort_pincushion(img):
2     rows, cols = img.shape[0:2]
3     xi, yi = np.meshgrid(np.arange(cols),
4                           np.arange(rows))
5     xmid = cols / 2.0
6     ymid = rows / 2.0
7     xi = xi - xmid
8     yi = yi - ymid
9     r, theta = cv2.cartToPolar(xi / xmid, yi / ymid)
10    F3 = 0.5
11    F5 = 1
12    r = r + F3 * r ** 3 + F5 * r ** 5
13    u, v = cv2.polarToCart(r, theta)
14    u = u * xmid + xmid
15    v = v * ymid + ymid
16    I_barrel =
17        cv2.remap(img, u.astype(np.float32),
18                  v.astype(np.float32), cv2.INTER_LINEAR)
19    return I_barrel
20
21 p = pincushion_distortion(source_img)
22 undistort_pin = undistort_pincushion(p)
23 cv2.imwrite(f'{render_dir}undistort_pin.png', undistort_pin)

```

Листинг 14: Код для коррекции подушкообразной дисторсии

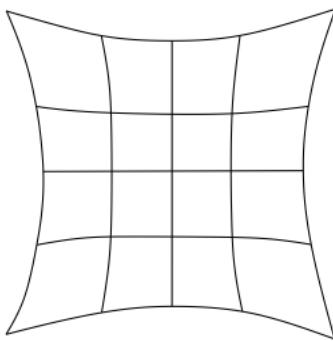


Рис. 21: Подушкообразная дисторсия

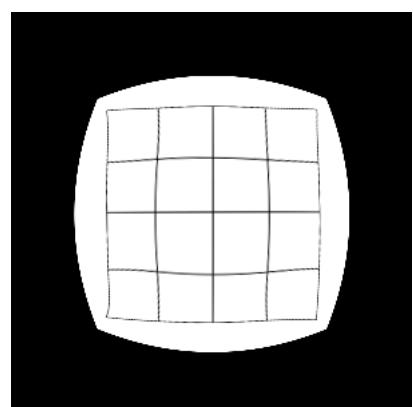


Рис. 22: Коррекция подушки бочкой

Изображение стало куда хуже выглядеть – потерялось качество и размер, но мы добились похожести на оригинал хоть в каком то виде

4 «Склейка» изображений

«Склейка» изображений – объединение двух или более изображений в единое целое, причем системы координат склеиваемых изображений могут отличаться из-за разного ракурса съемки, изменения положения камеры или движения самого объекта. Однако необходимо, чтобы оба изображения имели области перекрытия, то есть на них присутствовали одинаковые объекты. Основной задачей обработки таких изображений является введение их в общую систему координат. Для пересчета координат в случае аффинного отображения система 1.2 примет вид:

$$\begin{cases} x' = a_1 + a_2x + a_3y \\ y' = b_1 + b_2x + b_3 \end{cases}$$

Поэтому необходимо найти лишь по 3 коэффициента преобразования по каждой координате:

$$\begin{aligned} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} &= \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}^{-1} \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \\ \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} &= \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}^{-1} \begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \end{bmatrix} \end{aligned}$$

Для этого на обоих изображениях следует выбрать соответствующие пары точек (три пары в случае аффинного искажения и не менее шести пар в случае полиномиального искажения). Будем рассматривать простой случай «склейки» двух неискаженных изображений, имеющих одинаковую ширину. Необходимо склеить их по вертикали, то есть добавить к первому второе снизу. Однако, граница склейки неизвестна. Этую задачу можно реализовать при помощи корреляционного подхода

4.1 Ручная склейка изображения

Разделим кошечку пополам по горизонтали:



Рис. 23: Верхняя часть изображения



Рис. 24: Нижняя часть изображения

```

1 def join_img(topPart, botPart):
2     templ_size = 10
3     templ = topPart[- templ_size:, :, :]
4     res = cv2.matchTemplate(botPart, templ,
5         cv2.TM_CCOEFF )
6     min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
7     result_img = np.zeros(( topPart . shape [0] + botPart . shape [0] - max_loc [1]
8                           - templ_size , topPart . shape [1] , topPart . shape [2]) , dtype
9                           = np . uint8 )
10    result_img[0: topPart.shape[0], :, :] = topPart
11    result_img[topPart.shape[0]:, :, :] = botPart [ max_loc [1] + templ_size : , : , :]
12
13 f_part = cv2.imread('tech-vision/source/top.jpg', cv2.IMREAD_COLOR)
14 s_part = cv2.imread('tech-vision/source/bot.jpg', cv2.IMREAD_COLOR)
15
16 ans = join_img(f_part, s_part)
17 cv2.imwrite(f'{render_dir}join.png', ans)

```

Листинг 15: Код для склейки верхней части изображения с нижней



Рис. 25: Оригинальное изображение



Рис. 26: Кошачье воссоединение

Почти идеально – в месте соединения некоторые детали потерялись

4.2 Автоматическая склейка изображения

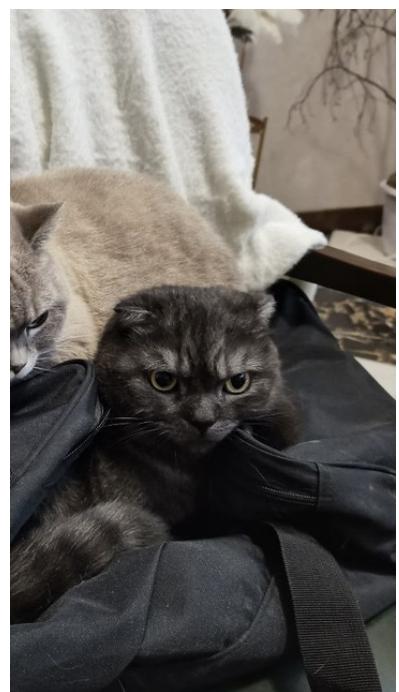
Всё то же самое, только теперь будем использовать класс Stitcher библиотеки OpenCV. Разделим кошек по вертикали на три части:



(a) Левая часть



(b) Центральная часть



(c) Правая часть

Рис. 27: Три части оригинального изображения

```

1 def auto_join_img(imgs:list):
2     stitcher = cv2.Stitcher.create(cv2.Stitcher_PANORAMA)
3     status, I_stitch = stitcher.stitch(imgs)
4     return status, I_stitch
5
6 I_3 = cv2.imread('tech-vision/source/I1.jpg', cv2.IMREAD_COLOR)
7 I_2 = cv2.imread('tech-vision/source/I2.jpg', cv2.IMREAD_COLOR)
8 I_1 = cv2.imread('tech-vision/source/I3.jpg', cv2.IMREAD_COLOR)
9 status, I_stitch = auto_join_img([I_1, I_2, I_3])
10 cv2.imwrite(f'{render_dir}auto_join.png', I_stitch)

```

Листинг 16: Код для автоматической склейки изображений



Рис. 28: Оригинальное изображение



Рис. 29: Автоматическая склейка кошек

5 Ответы на вопросы

5.1 Каким образом можно выполнить поворот изображения, не используя матрицу поворота?

Самый простой способ – воспользоваться методом `rotate(image, par)` из библиотеки OpenCV. `image` – изображение, которое необходимо повернуть, а `par` – поворот на сколько и в какую сторону. Например, в `par` можно передать `cv2.ROTATE_90_CLOCKWISE`, `cv2.ROTATE_90_COUNTERCLOCKWISE`, `cv2.ROTATE_180`, чтобы повернуть на 90° по часовой стрелке, против и на 180° соответственно

5.2 Какое минимальное количество соответствующих пар точек необходимо задать на исходном и искаженном изображениях, если порядок преобразования $n = 4$?

Мы можем рассчитать это по следующей формуле:

$$t_{min} = \frac{((n+1))(n+2)}{2} = \frac{((4+1))(4+2)}{2} = \frac{5 \cdot 6}{2} = 15$$

Таким образом, необходимо передать минимум 15 пар точек

5.3 После геометрического преобразования изображения могут появиться пиксели с неопределенными значениями интенсивности. С чем это связано и как решается данная проблема?

Такое происходит при вращении изображения не на 90°. Есть несколько методов уточнения неопределенных пикселей:

1. Прямой метод

- (a) Для каждого пикселя вычисляются новые координаты
- (b) Округляются до ближайших целых значений
- (c) Неопределенным пикселям нового изображения присваивается яркость ближайшего после геометрического преобразования пикселя с неокругленными координатами

2. Обратный метод

- (a) Координаты каждого пикселя преобразованного изображения подвергаются обратному преобразованию в систему координат исходного изображения
- (b) Координаты округляются до ближайших целых значений
- (c) Берется яркость пикселя с такими координатами

6 Выводы

6.1 Лабораторная работа 2

В ходе проделанной работы мы познакомились с различными матричными преобразованиями изображений, начиная с простых эффектов, продолжая коррекцией дисторсий и заканчивая склеиванием картинок. Мы расширили свои познания о работе с библиотекой OpenCV и языком программирования python, а также поразмыслили над интересными вопросами и дали на них ответы