**Практическая работа № 3**

**Преобразования изображений**

**Цель работы**: Преобразования изображений и некоторые практические методы их вычисления.

**1. Деформирование изображений**

Применение матрицы аффинного преобразования Н к блокам изображения называется деформированием (или аффинным деформированием). Эта операция часто применяется в машинной графике, а также в некоторых алгоритмах компьютерного зрения. Ее легко выполнить с помощью пакета ndinage, входящего в библиотеку SciPy.

Команда

transformed im = ndimage.affine\_transform(im,A,b, size)

преобразует блок изображения im, применяя к нему линейное преобразование A и вектор параллельного переноса b, как описано выше. Необязательный аргумент size позволяет задать размер выходного изображения.

По умолчанию результирующее изображение имеет такой же размер, как исходное. Чтобы посмотреть, как работает эта функция, выполните следующие команды:

from scipy import ndimage

im = array(Image.open("v1.jpg").convert ("L"))

H = array([[1.4,0.05,-100], [0.05,1.5,-100], [0,0,1]])

im2 = ndimage.affine\_transform(im, H[:2,:2], (H[0,2],H[1,2]))

figure()

gray()

imshow(im2)

show()

В результате получается изображение, показанное справа на рис. 1. Как видите, вместо отсутствующих пикселей в результирующее изображение добавлены нули.



Рис. 1. Пример деформирования изображения с помощью аффинного преобразования: справа показано изображение, получающееся в результате применения к левому функции ndimage.affine\_transform()

**Изображение внутри изображения**

В качестве простого примера аффинного деформирования рассмотрим размещение изображений или их частей внутри другого изображения выравниванием по определенным областям или опорным точкам.

Добавьте функцию image\_in\_image() в файл warp.ry. Эта функция принимает два изображения и координаты углов области внутри второго изображения, куда следует поместить первое изображение.

import homography

def image\_in\_image(im1, im2, tp):

""" Поместить im1 в im2 с помощью аффинного преобразования, так чтобы углы были как можно ближе к tp. Углы tp выражены в однородных координатах и перечисляются против часовой стрелки, начиная с левого верхнего. """

# точки деформируемого изображения

m,n = im1.shape[:2]

fp = array([[0,m,m,0],[0,0,n,n],[1,1,1,1]])

# вычислить и применить аффинное преобразование

H = homography.Haffine\_from\_points(tp, fp)

im1\_t=ndimage.affine\_transform(im1,H[:2,:2],

(H[0,2],H[1,2]),im2.shape[:2])

alpha = (im1\_t > 0)

return (1-alpha)\*im2 + alpha\*im1\_t

Смешивая два изображения, мы создаем альфа-отображение, которое определяет, какую часть цвета пикселя брать из каждого изображения. Здесь для создания бинарного альфа-отображения мы пользуемся тем фактом, что деформированное изображение заполнено нулями вне границ, области совмещения. Для большей строгости можно было бы прибавить небольшое число к потенциально нулевым пикселям первого изображения. Обратите внимание, что точки изображения описываются в однородных координатах.

def image\_in\_image(im1, im2, tp): - это определение функции image\_in\_image, которая принимает три аргумента: im1 - исходное изображение, im2 - изображение, в которое будет вставлено im1, и tp - массив координат точек, используемых для определения аффинного преобразования.

m,n = im1.shape[:2] - здесь m и n используются для получения ширины и высоты исходного изображения im1.

fp = array([[0,m,m,0],[0,0,n,n],[1,1,1,1]]) - здесь создается массив fp, который содержит координаты четырех точек. Первая точка имеет координаты (0, m), вторая точка имеет координаты (0, 0), третья точка имеет координаты (n, n), и четвертая точка имеет координаты (1, 1).

H = Haffine\_from\_points(tp, fp) - здесь вызывается функция Haffine\_from\_points, которая принимает два аргумента: tp - массив координат точек, используемых для определения аффинного преобразования, и fp - массив координат точек, используемых для определения аффинного преобразования. Эта функция возвращает матрицу H, которая представляет собой матрицу аффинного преобразования.

im1\_t=ndimage.affine\_transform(im1,H[:2,:2], (H[0,2],H[1,2]),im2.shape[:2]) - здесь вызывается функция ndimage.affine\_transform, которая применяет аффинное преобразование к изображению im1 с использованием матрицы H. H[:2,:2] - это верхняя левая часть матрицы H, которая используется для преобразования осей x и y. (H[0,2],H[1,2]) - это координаты центра преобразования. im2.shape[:2] - это размер изображения im2 по осям x и y, который используется для определения размера выходного изображения.

alpha = (im1\_t > 0) - здесь создается булево значение alpha, которое равно True, если значение в im1\_t больше нуля, и False в противном случае.

return (1-alpha)\*im2 + alpha\*im1\_t - здесь возвращается результат функции. Если alpha равно True, то возвращается im1\_t, иначе возвращается im2.

Для тестирования этой функции поместим изображение цветка на изображение сптицами. Показанный ниже код вставляет левое изображение на рис. 2 во второе изображение. Координаты были определены вручную.

import warp

# пример аффинного деформирования im1 в im2

im1 = array(Image.open("i.jpg").convert ("L"))

im2 = array(Image.open("v1.jpg").convert ("L"))

# задать точки

tp = array([[50, 50,200,50], [200, 50,200,200], [0,0,1,1]])

im3 = image\_in\_image(im1, im2, tp)

figure()

gray()

imshow(im3)

axis('equal')

axis('off')

show()

  

Рис. 2. Пример вставки одного изображения внутрь другого с помощью аффинного преобразования

Снова отметим, что опорная точка tp выражена в однородных координатах. Если другие координаты, то изображение окажется в другой части изображения.

Функция Haffine\_from\_points() находит наилучшее аффинное преобразование для заданных пар соответственных точек. Если эффект перспективы мал, то это дает хорошие результаты. Но невозможно с помощью одного аффинного преобразования совместить все четыре угловые точки (но можно было бы подобрать проективное преобразование общего вида). Если мы все же хотим воспользоваться аффинным деформированием, так чтобы совместить все четыре угла, то можно применить полезный прием.

**Упражнение**

1. Найдите изображение с рекламным щитом и поместите в рекламный щит другое изображение.

*Три пары точек с помощью аффинного преобразования можно совместить точно, поскольку у такого преобразования шесть степеней свободы, а три пары соответствий дают ровно шесть ограничений (уравнения для трех пар координат х и у). Поэтому, если мы хотим, чтобы изображение точно заполнило весь рекламный щит, то можем разбить его на два треугольника и деформировать их по отдельности*

*Ниже приведен код.*

*# сопоставим исходные точки углам im1*

*m,n = im1.shape[:2]*

*fp = array([[0,m,m,0],[0,0,n,n],[1,1,1,1]])*

*# первый треугольник*

*tp2=tp[:,:3]*

*fp2=fp[:,:3]*

*# вычислить H*

*H =* homography.*Haffine\_from\_points(tp2, fp2)*

*im1\_t = ndimage.affine\_transform(im1,H[:2,:2],*

*(H[0,2],H[1,2]),im2.shape[:2])*

*# альфа – отображение для треугольника*

*alpha = warp.alpha\_for\_triangle(tp2, im2.shape[0], im2.shape[1])*

*im3 = (1-alpha)\*im2 + alpha\*im1\_t*

*# второй треугольник*

*tp2 = tp[:,[0,2,3]]*

*fp2 = fp[:,[0,2,3]]*

*# вычислить H*

*H =* homography.*Haffine\_from\_points(tp2, fp2)*

*im1\_t = ndimage.affine\_transform(im1, H[:2,:2],*

*(H[0,2],H[1,2]),im2.shape[:2])*

*# альфа – отображение для треугольника*

*alpha = warp.alpha\_for\_triangle(tp2, im2.shape[0],im2.shape[1])*

*im4 = (1-alpha)\*im3 + alpha\*im1\_t*

*figure()*

*gray()*

*imshow(im4)*

*axis('equal')*

*axis('off')*

*show()*

*Здесь мы создаем альфа-отображение для каждого треугольника, а затем объединяем все изображения. Чтобы вычислить альфа-отображение для треугольника, мы проверяем, можно ли за писать координаты пикселя в виде выпуклой комбинации угловых точек треугольника". Если да, то пиксель лежит внутри треугольника.*

*Добавьте функцию alpha\_for\_triangle(), использованную в предыдущем коде, в файл warp.py:*

*def alpha\_for\_triangle(points,m,n):*

*""" Создает альфа-отображение размера (пл) для треугольника, определенного своими вершинами (заданными нормированными однородными координатами). """*

*alpha = zeros ((m,n))*

*for i in range (min(points[0]),max (points [0])):*

*for j in range(min(points[1]) ,max (points [1])):*

*x=linalg.solve (points, [i,j,1])*

*if min(x) > 0: #все коэффициенты положительные*

*alpha[i,j] = 1*

*return alpha*