مبانی تصویر ۱.۱.۱

رضا عباس زاده دربان

اطلاعات گزارش

تاریخ: ۱۴۰۰/۱/۱۳

واژگان کلیدی:
تبدیلات هندسی
Affine mapping

چکیده

در این تمرین تبدیل هندسی affine و نحوه محاسبه پارامترهای آن با استفاده از نقاط مشترک دو تصویر شرح داده شده است.

۱-مقدمه

منطبق کردن تصاویر (image registeration) عموما به کمک نقاط کنترلی یا feature point انجام می‌شود و یک مدل ساده برای این کار، مدل affine است که در ادامه شرح داده شده است.

۲-توضیحات

برای منطبق کردن تصاویر ابتدا باید تعدادی نقطه مشترک در هر دو تصویر شناسایی کنیم. به عنوان مثال یک ساختمان، تابلو یا... می‌تواند یک نقطه مشترک بین دو تصویر باشد.

سپس با حل یک دستگاه، رابطه تبدیل این نقاط را به دست می‌آوریم.

۳-شکل‌ها، جدول‌ها و روابط (فرمول‌ها)

فرمول تبدیل هندسی affine برای یک نقطه مشترک در ۲ تصویر به صورت زیر است:

$$[x \quad y \quad 1] = [v \quad w \quad 1]\mathbf{T} = [v \quad w \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

این فرمول را به صورت زیر بازنویسی میکنیم:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} v & w & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & v & w & 1 \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{12} \\ t_{22} \\ t_{32} \end{bmatrix}$$

نتیجه حاصل از فرمول بازنویسی شده مشابه فرمول اول است.

حال اگر تعداد نقاط هم ارزی که در دو تصویر پیدا کرده‌ایم بیشتر (N>=3) باشد، فرمول به این صورت تغییر می‌کند:

$$\begin{bmatrix} x1 \\ y1 \\ x2 \\ y2 \\ x3 \\ y3 \end{bmatrix} = \begin{bmatrix} v1 & w1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & v1 & w1 & 1 \\ v2 & w2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & v2 & w2 & 1 \\ v3 & w3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & v3 & w3 & 1 \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{12} \\ t_{22} \\ t_{32} \end{bmatrix}$$

اگر به هر یک از ماتریس‌های بالا اسمی نسبت دهیم، رابطه به این شکل تبدیل می‌شود:

Projection = M * Transition

بنابراین برای به دست آوردن ماتریس انتقال affine که Transition نام گذاری کرده‌ایم کافیست دستگاه زیر حل شود:

Transition = M^{-1} * Projection

۴-نتیجه‌گیری

همانطور که از روابط بالا مشخص است، برای حل این معادله حداقل به سه نقطه‌ی اشتراکی نیاز داریم تا بتوانیم ۶ مجهول در ماتریس affine را محاسبه کنیم.

مبانی تصویر ۱.۱.۲

رضا عباس زاده دربان

اطلاعات گزارش

تاریخ: ۱۴۰۰/۱/۱۳

واژگان کلیدی:
Panoramic
Image stitching

چکیده

در این تمرین دو تصویر از یک منظره با دو زاویه متفاوت را به یکدیگر متصل می‌کنیم و تصویری وسیع‌تر ایجاد می‌کنیم.

۱-مقدمه

برای انطباق و اتصال تصاویر(image registeration)، مشابه استفاده از تبدیل affine، نیاز به تعدادی نقطه کنترلی داریم تا بتوانیم تبدیل هندسی بین دو تصویر را پیدا کنیم. در این تمرین از مدل دو خطی برای مدل کردن تبدیل هندسی استفاده می‌کنیم.

در این روابط تصویر Car1 مرجع در نظر گرفته می‌شود و تابع تبدیل به دست آمده روی Car2 اعمال می‌شود. بعد از اعمال تابع تبدیل، به منظور تکمیل کردن پیکسل‌های بدون مقدار، درونیابی به روش نزدیکترین همسایه تا شعاع ۲ پیکسلی انجام می‌شود.

۲-توضیحات

در مدل دوخطی به کمک چهار نقطه کنترلی، می‌توان ۸ مجهول رابطه زیر را به دست آورد:

$$\begin{cases} x = c_1 + c_2 v + c_3 w + c_4 vw \\ y = c_5 + c_6 v + c_7 w + c_8 vw \end{cases}$$

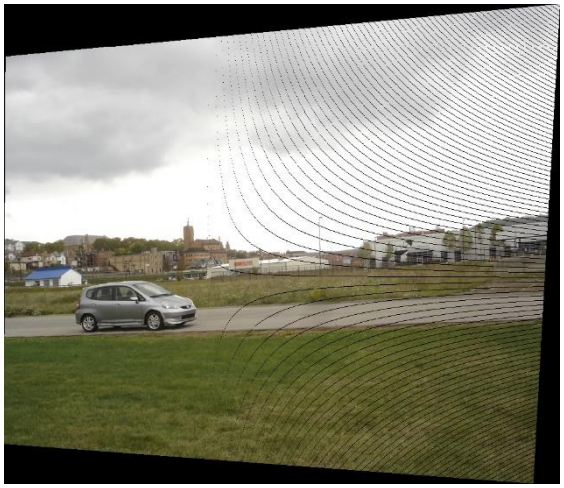
الگوریتم های مختلفی برای به دست آوردن نقاط کنترلی وجود دارد اما در این تمرین به صورت دستی ۴ نقطه مشترک در دو تصویر را مشخص می‌کنیم.

۳-شکل‌ها، جدول‌ها و روابط (فرمول‌ها)

تصویر car1 به همراه نقاط کنترلی:

تصویر car2 به همراه نقاط کنترلی:

اعمال تبدیل هندسی روی تصویر دوم:



درونیابی تصویر دوم بعد از اعمال تبدیل هندسی:

نتیجه اتصال دو تصویر:

با تغییر نقاط کنترلی نتایج دیگری به دست آمد که برخی از آن‌ها در ادامه آمده است:

۴-نتیجه‌گیری

از نتایج به دست آمده می‌توان نتیجه گرفت که انتخاب نقاط کنترلی تاثیر زیادی در تصویر نهایی خواهد داشت.

پیوست

کتابخانه های استفاده شده:

```
import numpy as np
import cv2
from scipy import interpolate
```

خواندن تصاویر از حافظه و مشخص کردن نقاط کنترلی:

```
img1 = cv2.imread('Car1.jpg')
img2 = cv2.imread('Car2.jpg')

point1_src = [835, 397]  # billboard
point1_dst = [415, 414]
point2_src = [438, 482]  # road beginning
point2_dst = [3, 505]
point3_src = [692, 703]  # grass
point3_dst = [273, 726]
point4_src = [440, 252]  # cloud
point4_dst = [4, 255]
```

حل معادله معرفی شده درتوضیحات و به دست آوردن مجهول های c1 تا c8 :

```
v_w = np.array([
                [1, point1_dst[0], point1_dst[1], point1_dst[0] *
point1_dst[1]],

                 [1, point2_dst[0], point2_dst[1], point2_dst[0] *
point2_dst[1]],

                 [1, point3_dst[0], point3_dst[1], point3_dst[0] *
point3_dst[1]],

                 [1, point4_dst[0], point4_dst[1], point4_dst[0] *
point4_dst[1]]])

x = np.array([point1_src[0], point2_src[0], point3_src[0], point4_src[0]])

c1, c2, c3, c4 = np.linalg.solve(v_w, x)

y = np.array([point1_src[1], point2_src[1], point3_src[1], point4_src[1]])

c5, c6, c7, c8 = np.linalg.solve(v_w, y)
```

پیدا کردن گوشه های تصویر تبدیل شده به منظور تشکیل تصویر نهایی:

```
bottom_left_y = int(c5 + (c6 * 0) + (c7 * img2.shape[0]) + (c8 *
img2.shape[0] * 0))
bottom_left_x = int(c1 + (c2 * 0) + (c3 * img2.shape[0]) + (c4 *
img2.shape[0] * 0))

bottom_right_y = int(c5 + (c6 * img2.shape[1]) + (c7 * img2.shape[0]) + (c8
* img2.shape[0] * img2.shape[1]))
bottom_right_x = int(c1 + (c2 * img2.shape[1]) + (c3 * img2.shape[0]) + (c4
* img2.shape[0] * img2.shape[1]))
```

```
top_right_y = int(c5 + (c6 * img2.shape[1]) + (c7 * 0) + (c8 * 0 *
img2.shape[1]))
top_right_x = int(c1 + (c2 * img2.shape[1]) + (c3 * 0) + (c4 * 0 *
img2.shape[1]))

top_left_y = int(c5)
top_left_x = int(c1)

min_x = min(top_left_x, bottom_left_x)
max_x = max(top_right_x, bottom_right_x)
min_y = min(top_right_y, top_left_y)
max_y = max(bottom_left_y, bottom_right_y)
```

تشکیل تصویر تبدیل شده:

```
img2_converted = np.zeros((max_y - min_y, max_x - min_x, 3), dtype=int)

for i in range(0, img2.shape[0]):
    for j in range(0, img2.shape[1]):
        for k in range(3):
            xnew = int(c1 + (c2 * j) + (c3 * i) + (c4 * i * j)) - min_x
            ynew = int(c5 + (c6 * j) + (c7 * i) + (c8 * i * j)) - min_y
            img2_converted[ynew, xnew, k] = img2[i, j, k]
```

انجام درونیابی روی تصویر تبدیل شده:

```
cv2.imwrite("img2-converted.jpg", img2_converted)
img2_converted = nearest_interpolate(img2_converted, max_distance=2)
cv2.imwrite("img2-interpolated.jpg", img2_converted)
min_y -= 2
max_y += 2
min_x -= 2
max_x += 2
```

و قرار گرفتن دو تصویر در تصویر نهایی:

```
stitched_img = np.zeros((max_y - min(0, min_y), max_x, 3), dtype=int)

for i in range(0, img1.shape[0]):
    for j in range(0, img1.shape[1]):
        for k in range(3):
            stitched_img[i - min(0, min_y), j, k] = img1[i, j, k]

for i in range(0, max_y - min_y):
    for j in range(0, max_x - min_x):
        for k in range(3):
            if img2_converted[i, j, k] != 0:
                stitched_img[i - min(0, min_y) + min_y, j + min_x, k] =
img2_converted[i, j, k]

cv2.imwrite("res-1.1.2.jpg", stitched_img)
```

تابع مربوط به درونیابی به روش نزدیکترین همسایه که تا شعاع دو پیکسل انجام شده است:

```
def add_padding(img, padSize=3):
    padded = np.zeros((img.shape[0] + padSize * 2, img.shape[1] + padSize *
2, img.shape[2]))
```

```
    padded[padSize:img.shape[0] + padSize, padSize:img.shape[1] + padSize,
:] = img
    return padded


def nearest_interpolate(img, max_distance=2):
    if len(img.shape) == 2:
        img = img[:, :, np.newaxis]
    padding_size = max_distance
    img = add_padding(img, padding_size * 2)
    copied_img = copy.deepcopy(img)
    img = img.astype(np.float32)
    for i in range(padding_size, img.shape[0] - padding_size):
        for j in range(padding_size, img.shape[1] - padding_size):
            for k in range(0, img.shape[2]):
                if img[i, j, k] == 0:
                    distance = 1
                    found = False
                    while distance <= max_distance and not found:
                        for i2 in range(i - distance, i + distance + 1):
                            for j2 in range(j - distance, j + distance +
1):
                                if img[i2, j2, k] != 0:
                                    copied_img[i, j, k] = img[i2, j2, k]
                                    found = True
                                    break
                            if found:
                                break
                        distance += 1
        print("interpolation row", i, "done")
    return copied_img
```

بعد از استفاده از این تابع، به تصویر مورد نظر از هر سمت ۲ پیکسل padding اضافه شده است. به دلیل اینکه برای پیدا کردن نزدیکترین همسایه شروط اضافه ایجاد نشود.

مبانی تصویر ۱.۱.۳

رضا عباس زاده دربان

اطلاعات گزارش

تاریخ: ۱۴۰۰/۱/۱۳

واژگان کلیدی:
چرخش
rotation

چکیده

در این تمرین، یک تصویر را حول مرکزش به اندازه های ۴۵و ۱۰۰ و ۶۷۰ درجه می-چرخانیم.

۱-مقدمه

برای چرخش تصویر از تابع تبدیل affine استفاده می-کنیم.

۲-توضیحات

تابع تبدیل چرخش را روی همه پیکسل‌های تصویر اعمال می‌کنیم و در تصویر جدید جایگزین می‌کنیم.

۳-شکل‌ها، جدول‌ها و روابط (فرمول‌ها)

فرمول کلی تبدیل هندسی affine :

$$[x \quad y \quad 1] = [v \quad w \quad 1]\mathbf{T} = [v \quad w \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

ماتریس تبدیل مورد استفاده برای چرخش به اندازه θ:

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

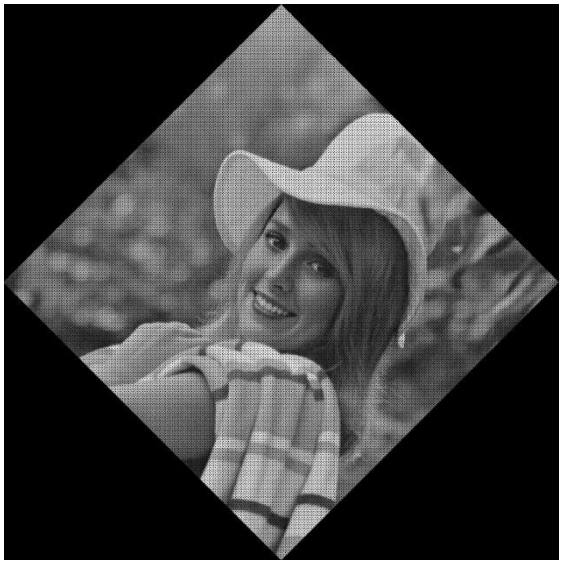
$x = v \cos\theta - w \sin\theta$
$y = v \sin\theta + w \cos\theta$

برای اینکه تصویر حول مرکز خودش بچرخد، فرمول بالا به شکل زیر تبدیل می‌شود:

$x = (v\text{-width}/2) \cos\theta - (w\text{-height}/2) \sin\theta$
$y = (v\text{-width}/2) \sin\theta + (w\text{-height}/2) \cos\theta$

نتایج:

چرخش ۴۵ درجه:



بعد از درونیابی:

چرخش ۱۰۰ درجه:

بعد از درونیابی:

چرخش ۶۷۰ درجه:

بعد از درونیابی:

پیوست

کتابخانه های استفاده شده:

```
from math import cos, sin
import numpy as np
import cv2
```

تابع چرخش با استفاده از روابط بالا:

```
def rotate(img, degree):
    w = int(img.shape[1])
    h = int(img.shape[0])
    result_size = max(w, h) * 2
    rotated = np.zeros((result_size, result_size), dtype=int)


    radian = degree / 180 * np.pi

    for i in range(0, img.shape[1]):
        for j in range(0, img.shape[0]):
            x = (j - img.shape[1] / 2) * cos(radian) - (i - img.shape[0] / 2) * sin(radian)
            y = (j - img.shape[1] / 2) * sin(radian) + (i - img.shape[0] / 2) * cos(radian)
            rotated[int(y + rotated.shape[0] / 2), int(x + rotated.shape[1] / 2)] = img[i, j]

    rotated = crop_gray_image(rotated)
    return rotated
```

در این تابع تصویر حاصل چرخش، یک تصویر مربعی با ابعاد دوبرابر ماکزیمم طول و عرض تصویر اصلی است که در خط آخر فضاهای اضافه اطراف تصویر با استفاده از تابع زیر برش داده شده است:

```
def crop_gray_image(img, tol=0):
    mask = img > tol
    return img[np.ix_(mask.any(1), mask.any(0))]
```

انجام چرخش با زوایای خواسته شده:

```
img = cv2.imread('Elaine.bmp', cv2.IMREAD_GRAYSCALE)

res45 = rotate(img, 45)
cv2.imwrite('1.1.3-45.jpg', res45)
cv2.imwrite('1.1.3-45-interpolated.jpg', nearest_interpolate(res45))

res100 = rotate(img, 100)
cv2.imwrite('1.1.3-100.jpg', res100)
cv2.imwrite('1.1.3-100-interpolated.jpg', nearest_interpolate(res100))

res670 = rotate(img, 670)
cv2.imwrite('1.1.3-670.jpg', res670)
cv2.imwrite('1.1.3-670-interpolated.jpg', nearest_interpolate(res670))
```

مبانی تصویر ۱.۲.۱

رضا عباس زاده دربان

اطلاعات گزارش	چکیده
تاریخ: ۱۴۰۰/۱/۱۳	در این تمرین یک تصویر را در دو حالت equalized و non equalized هیستوگرام به شش سطح 4,8,16,32,64,128 سطح بندی (quantize) می‌کنیم و سپس خطای میانگین مربعات نسبت به تصویر اصلی را برای هر حالت به دست می‌آوریم.
واژگان کلیدی: تبدیلات هندسی Affine mapping	

۱-مقدمه

ابتدا تصویر را quantized می‌کنیم و سپس آن را equalize می‌کنیم و به مقایسه این دو تصویر می‌پردازیم.

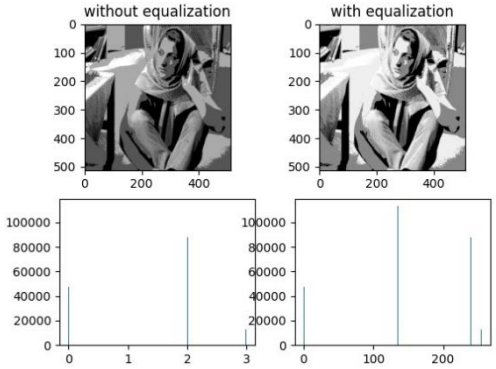
۲-توضیحات

برای quantize کردن یک تصویر، مقدار هر پیکسل را بر سطح مورد نظر، به صورت **صحیح** تقسیم می‌کنیم و سپس در همان سطح ضرب می‌کنیم تا حد پایین (floor) آن در سطح بندی جدید به دست آید.

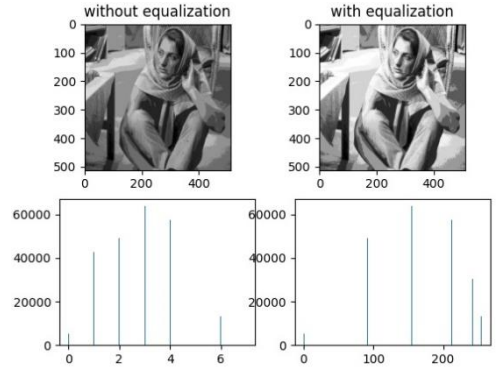
برای equalize کردن تصویر از تابع equalizeHist موجود در کتابخانه open cv استفاده شده‌است.

۳-شکل‌ها، جدول‌ها و روابط (فرمول‌ها)

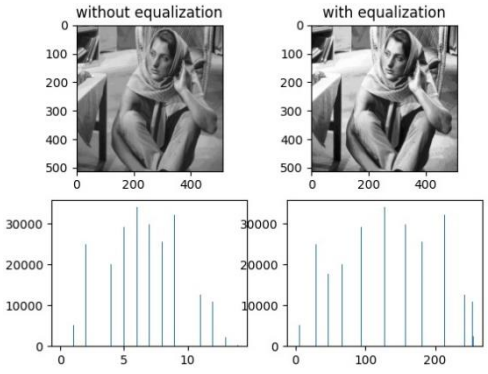
خروجی ۴ سطح خاکستری:



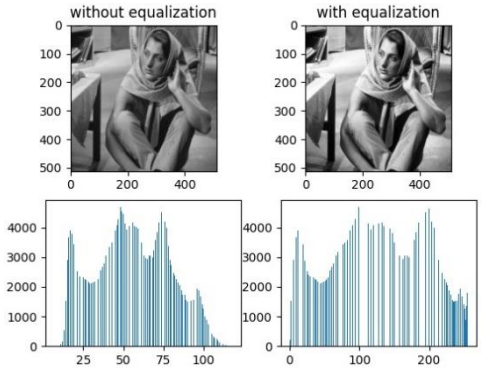
خروجی ۸ سطح خاکستری:



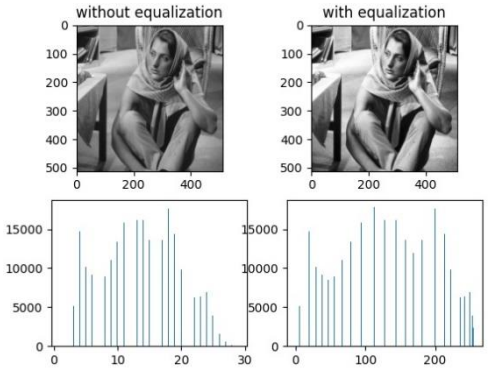
خروجی ۱۶ سطح خاکستری:



خروجی ۱۲۸ سطح خاکستری:



خروجی ۳۲ سطح خاکستری:



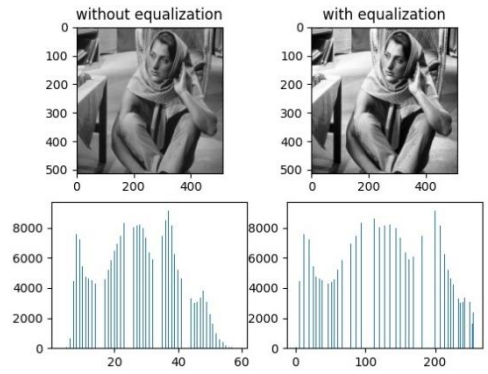
فرمول محاسبه mse:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

جدول خطای میانگین مربعات:

level	4	8	16	32	64	128
quantized	107.48942947387695	108.58281707763672	109.59280014038086	102.06668090820312	110.85026550292969	116.69033432006836
equalized + quantized	116.53982543945312	98.01694107055664	99.0640983581543	108.37175750732422	109.38105010986328	107.18338775634766

خروجی ۶۴ سطح خاکستری:



۴-نتیجه‌گیری

با استفاده از equalization معمولا کنتراست تصویر افزایش پیدا می‌کند و نسبت به تصویر اصلی تفاوت زیادی که با توجه به کاربرد می‌تواند مفید باشد.

به عنوان مثال در تصاویر پزشکی که می‌خواهیم تفاوت‌ها در تصویر مشخص تر باشند استفاده از equalization کمک می‌کند اما در تصاویری مانند تصویر این تمرین شاید از دید انسان تصویر از وضعیت عادی خود خارج شده باشد.

پیوست

کتابخانه‌های مورد استفاده:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

تابع quantize:

```
def quantize(img, level):
    return np.floor(img / (256 / level)).astype(np.uint8)
```

خواندن تصویر از حافظه

داخل حلقه برای هر سطح خاکستری ابتدا تصویر quantized می‌شود. سپس equalized می‌شود. خطای هر دو حالت محاسبه می‌شود و در جدول ذخیره می‌شود.

```
img = cv2.imread('Barbara.bmp', cv2.IMREAD_GRAYSCALE)

table_rows = [
    ["quantized"],
    ["equalized + quantized"]
]
table_cols = ("level", 4, 8, 16, 32, 64, 128)

for level in (4, 8, 12, 32, 64, 128):
    quantized = quantize(img, level)
    equalized_quantized = cv2.equalizeHist(quantized)

    mse_quantized = np.square(np.subtract(img, quantized)).mean()
    mse_equalized_quantized = np.square(np.subtract(img,
equalized_quantized)).mean()

    table_rows[0].append(mse_quantized)
    table_rows[1].append(mse_equalized_quantized)

    fig1, axs = plt.subplots(2, 2)
    axs[0][0].imshow(quantized, cmap='gray')
    axs[0][0].set_title("without equalization")
    axs[1][0].hist(quantized.flatten(), 255)

    axs[0][1].imshow(equalized_quantized, cmap='gray')
    axs[0][1].set_title("with equalization")
    axs[1][1].hist(equalized_quantized.flatten(), 255)

    fig1.savefig('images/res' + str(level) + '.jpg')

fig, ax = plt.subplots(dpi=300, figsize=(7, 1))
ax.axis('off')
ax.table(cellText=table_rows, colLabels=table_cols, loc='center')
fig.savefig('images/table.jpg')
```

مبانی تصویر ۱.۲.۲

رضا عباس زاده دربان

اطلاعات گزارش

تاریخ: ۱۴۰۰/۱/۱۳

واژگان کلیدی:
Down sampling
Up sampling
Bilinear interpolation
Pixel replication

چکیده

در این تمرین یک تصویر را به دو روش down sample می‌کنیم. سپس تصویر را با استفاده از روش های درونیابی up sample می‌کنیم.

۱-مقدمه

برای down sampleکردن یک بار با استفاده از فیلتر میانگین گیری و یک بار بدون استفاده از آن نرخ نمونه برداری تصویر را نصف ی‌کنیم.

برای up sample از دو روش تکرار پیکسل و درونیابی دوخطی استفاده خواهیم کرد.

۲-توضیحات

در down sample بدون فیلتر به صورت یکی در میان یک سطر و ستون را حذف می‌کنیم.

IMG (x,y) = img(2x,2y)

در روش استفاده از فیلتر میانگین سطر و ستون های حذف شده در تصویر نهایی تاثیر دارد:

IMG(x,y) = (img(2x,2y) + img(2x+1,2y) +
(img(2x,2y+1) + img(2x+1,2y+1)) / 4

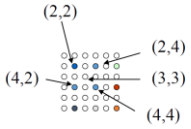
در up sample به روش pixel_replication هر سطر و ستون دو بار تکرار می‌شود:

IMG(x,y) = img(x/2 , y/2)

در روش درونیابی دوخطی دستگاه معادله ای تشکیل می-دهیم و مقدار پیکسل مرکزی را تخمین می‌زنیم:

$$v(x,y) = ax + by + cxy + d$$

$$125 = a(2) + b(2) + c(4) + d$$
$$170 = a(2) + b(4) + c(8) + d$$
$$172 = a(4) + b(2) + c(8) + d$$
$$170 = a(4) + b(4) + c(16) + d$$

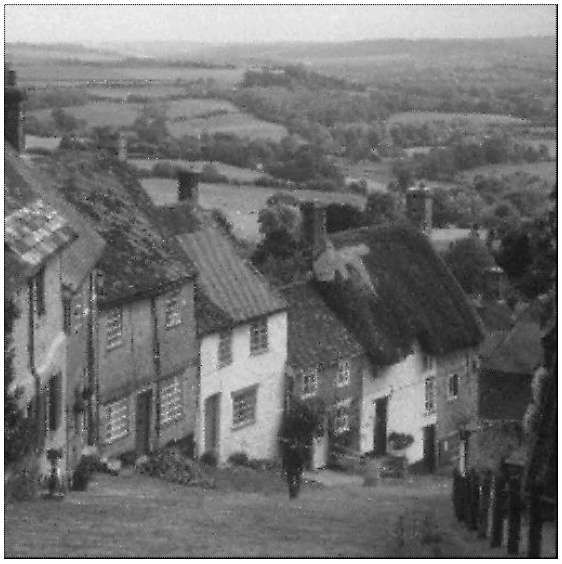


$$v(3,3) = a(3) + b(3) + c(9) + d$$

۳-شکل‌ها، جدول‌ها و روابط (فرمول‌ها)

Simple down sample:

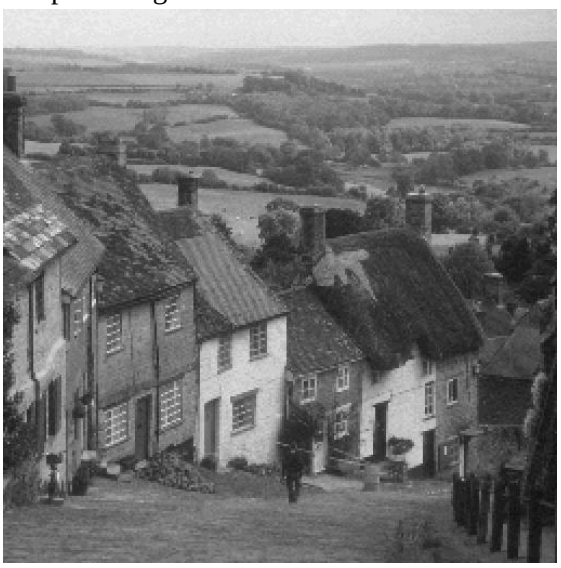
Up sample bilinear on simple down sampled image:



Up sample bilinear on Average down sampled image:

Average down sample:

Up sample pixel replication on simple down sampled image:



Up sample pixel replication on average down sampled image:

جدول خطای هر یک از عکس های up sample شده

نسبت به عکس اصلی:

*	Pixel replication up	bilinear up
Average down	136.935791015625	268.22383344173295
Simple down	133.075927734375	245.2845325469996

پیوست

تابع های down sample & up sample

```
def down_sample_simple(img):
    width = int(img.shape[1] / 2)
    height = int(img.shape[0] / 2)
    res = np.zeros((height, width))
    for i in range(0, height):
        for j in range(0, width):
            res[i, j] = img[i * 2, j * 2]
    return res


def down_sample_average(img):
    width = int(img.shape[1] / 2)
    height = int(img.shape[0] / 2)
    res = np.zeros((height, width))
    for i in range(0, height - 1):
        for j in range(0, width - 1):
            res[i, j] = (int(img[i * 2, j * 2]) + int(img[i * 2, j * 2 +
1]) +
                         int(img[i * 2 + 1, j * 2]) + int(img[i * 2 + 1, j
* 2 + 1])) / 4
    return res


def up_sample_pixel_replication(img):
    width = img.shape[1] * 2
    height = img.shape[0] * 2
    res = np.zeros((height, width))
    for i in range(0, height):
        for j in range(0, width):
            res[i, j] = img[int(i / 2), int(j / 2)]
    return res


def up_sample_bilinear_interpolation(img):
    width = img.shape[1] * 2
    height = img.shape[0] * 2
    res = np.zeros((height, width))
    for i in range(0, height):
        if i % 2 == 0:
            for j in range(0, width):
                if j % 2 == 0:
                    res[i, j] = img[int(i / 2), int(j / 2)]

    for i in range(1, height-2, 2):
        for j in range(1, width-2, 2):
            x_y = np.array([[i + 1, j - 1, (i + 1) * (j - 1), 1],
                            [i + 1, j + 1, (i + 1) * (j + 1), 1],
```

```
                            [i - 1, j + 1, (i - 1) * (j + 1), 1],
                            [i - 1, j - 1, (i - 1) * (j - 1), 1]])
            v = np.array(
                [res[i + 1, j - 1], res[i + 1, j + 1], res[i - 1, j + 1],
                 res[i - 1, j - 1]])
            x = np.linalg.solve(x_y, v)
            res[i, j] = x[0] * i + x[1] * j + x[2] * i * j + x[3]

    for i in range(1, height - 2):
        if i % 2 == 1:
            k = 2
        else:
            k = 1
        for j in range(k, width - 2, 2):
            x_y = np.array([[i + 1, j, (i + 1) * (j), 1],
                            [i, j + 1, (i) * (j + 1), 1],
                            [i - 1, j, (i - 1) * (j), 1],
                            [i, j - 1, (i) * (j - 1), 1]])

            v = np.array([res[i + 1, j], res[i, j + 1], res[i - 1, j],
                          res[i, j - 1]])
            if res[i, j] == 0:
                try:
                    x = np.linalg.solve(x_y, v)
                    res[i, j] = x[0] * (i) + x[1] * j + x[2] * i * j + x[3]
                except:
                    res[i, j] = (res[i + 1, j] + res[i, j + 1] + res[i - 1,
j] +
                                 res[i, j - 1]) / 4

    return res
```

استفاده از تابع های تعریف شده و ذخیره خروجی ها:

```
img = cv2.imread('Goldhill.bmp', cv2.IMREAD_GRAYSCALE)
down_avg = down_sample_average(img)
cv2.imwrite('down_avg.jpg', down_avg)
down_simple = down_sample_simple(img)
cv2.imwrite('down_simple.jpg', down_simple)

up_rep_down_avg = up_sample_pixel_replication(down_avg)
cv2.imwrite('up_rep_down_avg.jpg', up_rep_down_avg)
up_bilinear_down_avg = up_sample_bilinear_interpolation(down_avg)
cv2.imwrite('up_bilinear_down_avg.jpg', up_bilinear_down_avg)
up_rep_down_simple = up_sample_pixel_replication(down_simple)
cv2.imwrite('up_rep_down_simple.jpg', up_rep_down_simple)
up_bilinear_down_simple = up_sample_bilinear_interpolation(down_simple)
cv2.imwrite('up_bilinear_down_simple.jpg', up_bilinear_down_simple)

mse_up_rep_down_avg = np.square(np.subtract(img, up_rep_down_avg)).mean()
mse_up_bilinear_down_avg = np.square(np.subtract(img,
up_bilinear_down_avg)).mean()
mse_up_rep_down_simple = np.square(np.subtract(img,
up_rep_down_simple)).mean()
mse_up_bilinear_down_simple = np.square(np.subtract(img,
up_bilinear_down_simple)).mean()
data = [["Average down", mse_up_rep_down_avg, mse_up_bilinear_down_avg],
        ["Simple down", mse_up_rep_down_simple,
mse_up_bilinear_down_simple]]
```

```
col_labels = ("*", "Pixel replication up", "bilinear up")
fig, ax = plt.subplots(dpi=300, figsize=(5, 1))
ax.axis('off')
ax.table(cellText=data, colLabels=col_labels, loc='center')
fig.savefig('table1.2.2.png')
```

مبانی تصویر ۱.۲.۳

رضا عباس زاده دربان

اطلاعات گزارش

تاریخ: ۱۴۰۰/۱/۱۳

چکیده

در این تمرین به بررسی تاثیر تعداد سطوح خاکستری در کیفیت تصویر می‌پردازیم.

واژگان کلیدی:
Gray level

۱-مقدمه

عکس اولیه از ۸ بیت برای ذخیره هر پیکسل استفاده می-کند. در این تمرین بررسی می‌کنیم اگر این تعداد را به ۱،۲،۳،۴،۵ بیت کاهش دهیم، چه تغییری در تصویر به وجود می‌آید.

۲-توضیحات

برای کاهش تعداد سطوح خاکستری به n بیت از فرمول زیر استفاده می‌کنیم:

$$Img(x,y) = \left\lfloor \frac{img(x,y)}{2^{8-n}} \right\rfloor * 2^{8-n}$$

۳-شکل‌ها، جدول‌ها و روابط (فرمول‌ها)

تصویر اصلی:

1 bit:

2 bit:

3 bit:

4 bit:

5 bit:

۴-نتیجه‌گیری

همانطور که مشخص است، هرچه تعداد بیت ها ذخیره سازی بیشتر باشد، کیفیت تصویر بالاتر خواهد بود، چرا که جزییات بیشتری در تصویر قابل نمایش خواهد بود.

در این تصویر وجود ۵ بیت، تصویری مشابه با تصویر اصلی ایجاد می‌کند و افت کیفیت زیادی مشاهده نمی‌شود. اما با تعداد بیت کمتر می‌بینیم که تعداد نواحی به وجود می‌آید که در سراسر آن یک سطح خاکستری وجود دارد که باعث می‌شود کیفیت تصویر افت کند.

البته موضوع در همه نواحی عکس صادق نیست. به عنوان مثال در قسمت پارچه های راه‌راه به دلیل این که قسمت-های نزدیک به هم سطوح خاکستری متفاوتی ایجاد می-کنند، حتی با استفاده از ۳ بیت نیز کیفیت آن حفظ می-شود. اما در نواحی مانند زمین یا پوست که تغییر ناگهانی در رنگ نداریم، باعث ایجاد نواحی همرنگ و افت کیفیت می‌شود.