تبدیل هندسی

ترانه قندی

اطلاعات گزارش

تاریخ: ۱۳۹۸۰۸۰۲

واژگان کلیدی:
تبدیل هندسی
تبدیل affine
درون‌یابی
درون‌یابی نزدیک‌ترین همسایه
درون‌یابی دوخطی

چکیده

تبدیل های هندسی در پردازش تصویر کاربرد زیادی دارند. از این تبدیل‌ها می‌توان برای افزایش یا کاهش مقیاس، تغییر جهت تصویر و ... استفاده کرد. این تبدیل‌ها علاوه بر تغییر تصویر برای کاربردهای زیباسازی، زیرساختی برای سایر روش های پردازش تصویر به شمار می‌روند.

۱-مقدمه

تبدیل های هندسی توابعی برای پردازش تصویر هستند که نقاط، خطوط مستقیم و صفحات را حفظ می‌کنند. خطوط موازی پس از اعمال تبدیل بر روی آن‌ها موازی باقی می‌مانند. تبدیل های هندسی الزاما زوایا را حفظ نمی‌کنند؛ اما نسبت فواصل بین نقاطی که روی یک خط مستقیم قرار گرفته اند را حفظ می‌کنند. از تبدیل‌های هندسی می‌توان به identity، scaling، rotation، translation، shear(عمودی و افقی) می‌توان اشاره کرد.

۲-شرح تکنیکال

تبدیل هندسی به صورت زیر تعریف می‌شود:

اگر (v,w) مختصات در تصویر اصلی و (x,y) مختصات در تصویر تبدیل یافته باشد آنگاه:

$$(x,y) = T\{(v,w)\}$$

تبدیل affine یکی از رایجترین تبدیل های هندسی است که به صورت زیر تعریف می‌شود:

$$[x\ y\ ۱]=[v\ w\ ۱]T=[v\ w\ ۱]\begin{bmatrix} t11 & t12 & ۰ \\ t21 & t22 & ۰ \\ t31 & t32 & ۱ \end{bmatrix}$$

برای انجام انواع تبدیل‌های هندسی،کافیست ضرایب $t_{۱۱}$ تا$t_{۳۲}$ را تغییر دهیم تا تغییر مقیاس، دوران، و تغییر جهت را به دست آوریم. انواع ماتریس‌های تبدیل برای به دست آمدن تبدیل ها به صورت زیر می‌باشد.v و w مختصات در تصویر اصلی و x وy مختصات در تصویر تبدیل یافته هستند:

- **Identity**

این تبدیل، تبدیل همانی است که در آن تصویر تبدیل یافته با تصویر اصلی فرقی نخواهد داشت.

$$\begin{bmatrix} ۱ & ۰ & ۰ \\ ۰ & ۱ & ۰ \\ ۰ & ۰ & ۱ \end{bmatrix}$$

$$x=v$$
$$y=w$$

- **Scaling**

این تبدیل برای افزایش و یا کاهش مقیاس به کار می‌رود.

$$\begin{bmatrix} Cx & ۰ & ۰ \\ ۰ & Cy & ۰ \\ ۰ & ۰ & ۱ \end{bmatrix}$$

x=Cx * v

y=Cy * w

- **Rotation**

این تبدیل برای چرخش تصویر به کار می‌رود. زاویه‌ی چرخش همان پارامتر توابع سینوس و کسینوس می‌باشد.

$$\begin{bmatrix} \cos(theta) & \sin(theta) & ۰ \\ -\sin(theta) & \cos(theta) & ۰ \\ ۰ & ۰ & ۱ \end{bmatrix}$$

x=v*cos(theta)-w*sin(theta)

y=v*cos(theta)+w*sin(theta)

- **Translation**

این تبدیل برای تغییر موقعیت تصویر به کار می‌رود. میزان انتقال تصویر با T_x (میزان انتقال در راستای x) و T_y (میزان انتقال در راستای y) مشخص می‌شود.

$$\begin{bmatrix} ۱ & ۰ & ۰ \\ ۰ & ۱ & ۰ \\ T_x & T_y & ۱ \end{bmatrix}$$

$x=v+T_x$

$y=w+T_y$

- **Shear(vertical)**

از این تبدیل برای کشیده شدن گوشه‌ی تصویر در راستای عمودی استفاده می‌شود. میزان کشیده شدن نیز به پارامتر S_v وابسته است.

$$\begin{bmatrix} ۱ & ۰ & ۰ \\ S_v & ۰ & ۰ \\ ۰ & ۰ & ۱ \end{bmatrix}$$

$x=v+S_v*w$

y=w

- **Shear(horizontal)**

از این تبدیل برای کشیده شدن گوشه‌ی تصویر در راستای افقی استفاده می‌شود. میزان کشیده شدن نیز به پارامتر S_h وابسته است.

$$\begin{bmatrix} ۱ & S_h & ۰ \\ ۰ & ۱ & ۰ \\ ۰ & ۰ & ۱ \end{bmatrix}$$

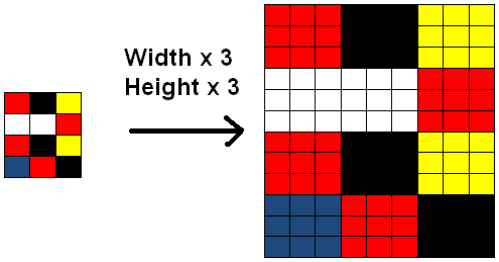
x= v

$y=S_h*v+w$

در برخی از پس از تبدیل، نقاطی که در تصویر اصلی مجاور بوده اند، مجاور نخواهند بود. در نتیجه نقاط بین آن ها در تصویر بدون مقدار باقی خواهند ماند. برای از بین بردن این مشکل و مقداردهی به نقاطی که مقدار آن ها نامشخص است، روش‌های مختلفی وجود دارد که در این تمرین به دو دسته از آنها پرداخته ایم:

- **درون یابی نزدیک‌ترین همسایه**

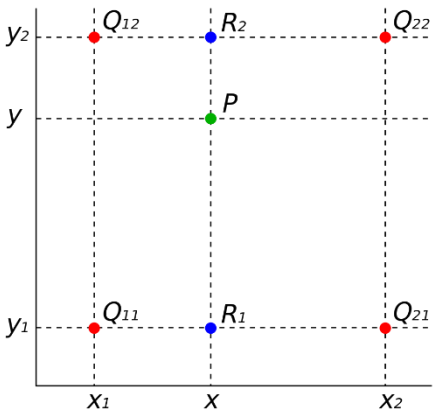
در این روش، مقدار پیکسل با در نظر گرفتن نزدیک ترین پیکسل به آن مشخص می‌شود.



در مثال بالا، پس از تغییر مقیاس تصویر و سه برابر شدن ابعاد آن، پیکسل های جدیدی در تصویر تبدیل یافته خواهیم داشت که مقدارآنها را با توجه به نزدیک ترین همسایه ی آن ها تخمین می‌زنیم. در این روش مقدار جدیدی ایجاد نمی‌شود؛ بلکه تمامی مقادیری که برای پیکسل های جدید درنظر می‌گیریم، جزو مقادیری هستند که در تصویر پیش از تبدیل داشته ایم.

- **درون‌یابی دوخطی**

در این روش مقدار پیکسل های جدید را با توجه به نزدیک ترین چهار همسایه‌ی آنها تخمین می‌زنیم.



در مثال بالا، برای تخمین مقدار نقطه‌ی p ، ابتدا دو همسایه ی آن را که بالاتر قرار دارند در نظر می گیریم و نقطه ی میان آنها را به دست می‌آوریم. سپس دو همسایه ی پایین تر را در نظر گرفته و نقطه ی میان آنها را نیز به دست می‌آوریم. مقدار p با توجه به این دو نقطه ی میانی که به دست آمد، محاسبه می‌شود.

$$v(x,y)= a*x + b*y + c*x*y + d$$

در این تمرین، ۶ نوع تبدیل بر روی تصویر "room" انجام شد:

- انتقال به اندازه ی ۱۰۲.۵ و ۱۸۰.۲ در راستای x وy
- Scale به اندازه ی ۱.۲ و ۰.۸۵ در راستای x و y
- دوران نسبت به مرکز تصویر به اندازه ی ۳۰ درجه
- Shear عمودی با مقدار ۱.۲۵
- Shear افقی با مقدار ۱.۵
- و انتقال به اندازه ی ۱۰۲.۵ و ۱۰۸.۲ پیکسل در راستای x وy، Scale به میزان ۱.۲ و ۰.۸۵ در راستای x و y و دوران نسبت به مرکز تصویر به اندازه ی ۳۰ درجه.

۳-شرح نتایج

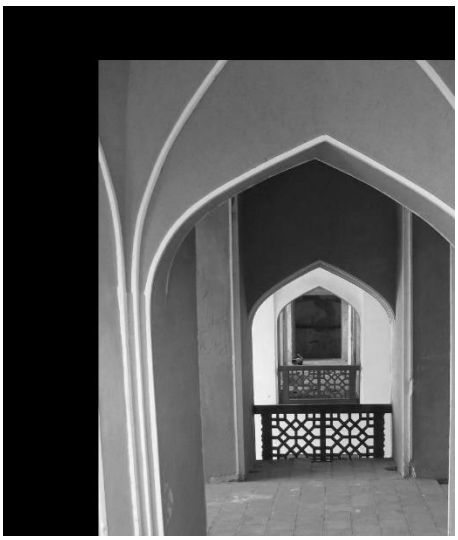
تصویر اصلی که تبدیل ها بر روی آن انجام شد، تصویر "room" می‌باشد:

تصویر ۱ - "room" ، تصویر اصلی

نتیجه ی تبدیل های ذکر شده در قسمت پیشین به صورت زیر می‌باشد:

- **نتیجه ی translation:**

از آنجا که درصورتی که تمامی تصویر نمایش داده شود، انتقال آن مشخص نخواهد شد، آن را به صورت زیر نمایش داده ایم:

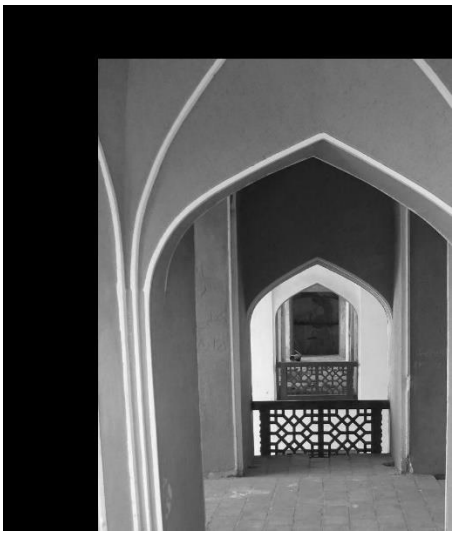


تصویر ۲- انتقال

تصویر خروجی حتی بدون درونیابی، نقطه ای ندارد که مقدار آن مشخص نباشد. دلیل آن این است که موقعیت پیکسل ها پس از انتقال به میزان ۱۰۲.۵ و ۱۰۸.۲، گرد شده و به یک عدد صحیح تبدیل می‌شود.

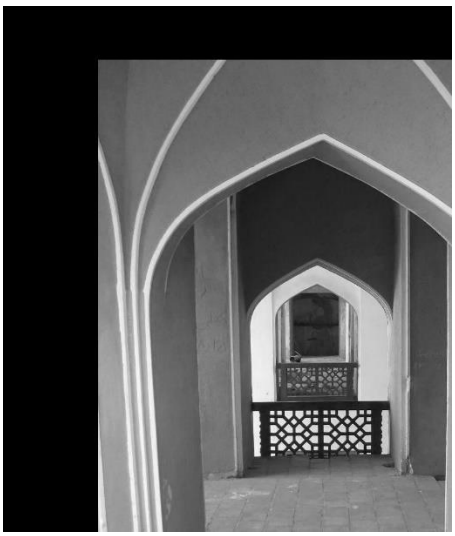
بنابراین نقاطی که در تصویر پیش از تبدیل مجاور بوده اند پس از تبدیل نیز مجاور خواهند بود و نیازی به درون یابی نخواهیم داشت.

نتیجه پس از درون یابی نزدیک ترین همسایه:



تصویر ۳- انتقال-نزدیک ترین همسایه

نتیجه پس از درون یابی دوخطی:



تصویر ٤-انتقال-دوخطی

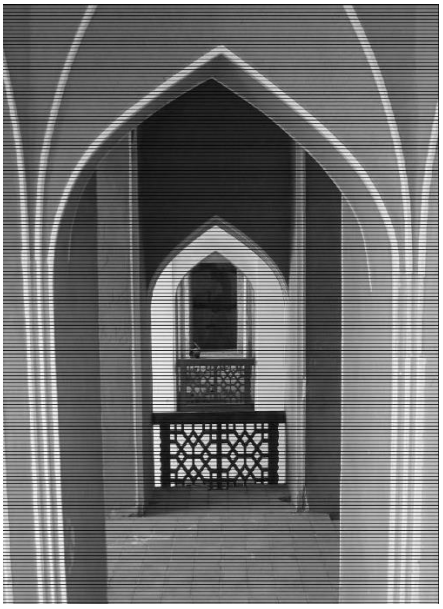
- **نتیجه ی scale:**

تصویر ٥-scale

نتیجه با استفاده از درون یابی نزدیک ترین همسایه:

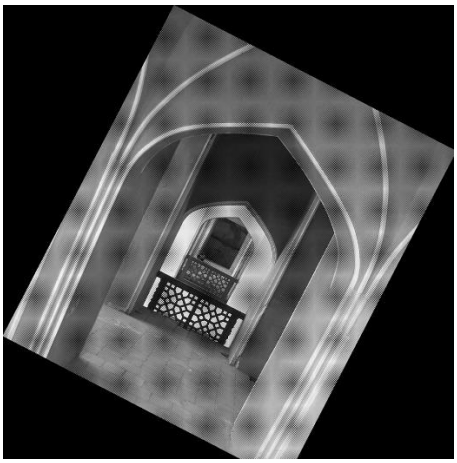
تصویر ٦-scale نزدیک ترین همسایه

نتیجه با استفاده از درون یابی دوخطی:



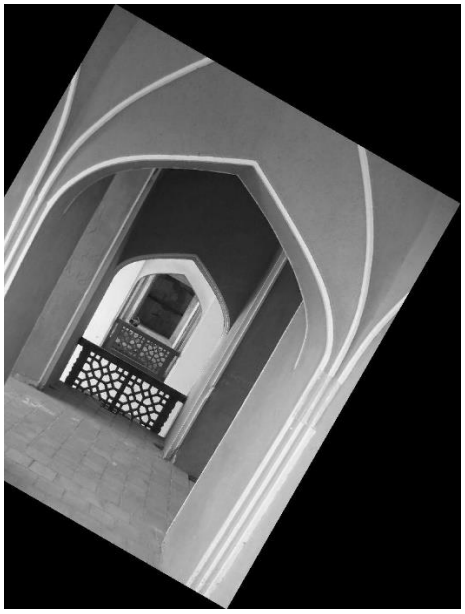
تصویر ۷-scale- درون یابی دوخطی

- نتیجه ی دوران:



تصویر ۸-چرخش

نتیجه با استفاده از درون‌یابی نزدیک ترین همسایه:



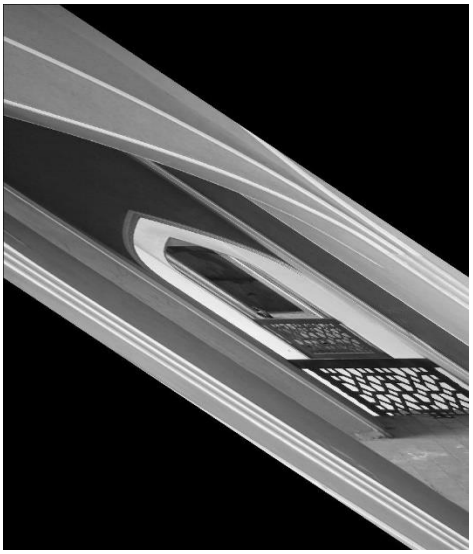
تصویر ۹-چرخش-نزدیک تزین همسایه

- نتیجه ی shear (عمودی)

تصویر ۱۰-vertical shear

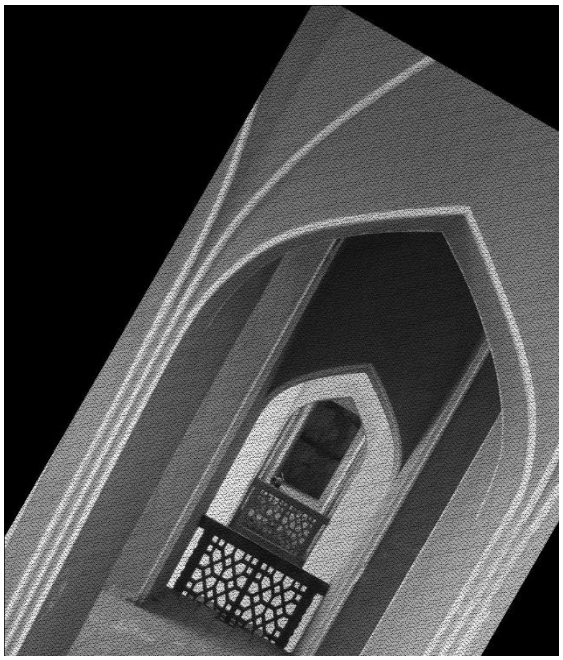
در این تبدیل فاصه یا gap بین پیکسل ها ایجاد نمی‌شود؛ زیرا بین نقاطی که پیش از نبدیل همسایه بوده اند پس از نبدیل فاصله نمی‌افتد، بلکه هر نقطه همسایه ی جدیدی خواهد داشت که پیش از تبدیل بالاتر آن در تصویر قرار داشته است، و درواقع نیازی به درون یابی نیست.

- نتیجه ی shear افقی



تصویر ۱۱-horizontal shear

-نتیجه ی ترکیب انتقال،shear و scale



۴- پیوست (کد برنامه)

کد برنامه به این صورت است:

ابتدا تصویر را خوانده و ماتریس های تبدیل را ایجاد و مقدار دهی می کنیم:

```
image_input = plt.imread('room.jpg')

# ----------------------TRANSLATION--
-----------------------------------
matrix_translation = numpy.array([[1,
0, 0], [0, 1, 0], [102.5, 180.2, 1]])
# -------------------------SCALE-----
-----------------------------------
matrix_scale = numpy.array([[1.2, 0,
0], [0, 0.85, 0], [0, 0, 1]])
# -------------------------ROTATION--
-----------------------------------
matrix_rotation =
np.array([[math.cos(math.radians(30))
, math.sin(math.radians(30)), 0],
                              [-
math.sin(math.radians(30)),
math.cos(math.radians(30)), 0], [0,
0, 1]])
# -------------------------
SHEAR(HORIZONTAL)--------------------
-------------------
matrix_shear_horiziontal =
numpy.array([[1, 1.5, 0], [0, 1, 0],
[0, 0, 1]])
# -------------------------
SHEAR(VERTICAL)----------------------
-----------------
# --------------------------ALL-----
-----------------------------------
matrix_shear_vertical =
numpy.array([[1, 0, 0], [1.25, 1, 0],
[0, 0, 1]])
matrix_all=matrix_translation.dot(mat
rix_scale)
matrix_all=matrix_all.dot(matrix_rota
tion)
```

سپس سه عکس را به صورت ماتریس ایجاد می کنیم: image_output_raw که تصویر تبدیل یافته بدون اعمال درون یابی است؛ image_interpolated_nearest_neighbor که تصویر خروجی پس از اعمال درون یابی نزدیک ترین همسایه است و image_output_bilinear که تصویر خروجی پس از اعمال درون یابی دوخطی است.

```
image_output_raw =
transform_image(image_input,
matrix_rotation)
image_interpolated_nearest_neighbor =
np.empty((image_input.shape[0] * 3,
image_input.shape[1] * 3, 3),
dtype=np.uint8)
image_interpolated_bilinear =
np.empty((image_input.shape[0] * 3,
image_input.shape[1] * 3, 3),
dtype=np.uint8)

pad_image =
np.empty((image_input.shape[0] * 3,
```

```
image_input.shape[1] * 3, 3),
dtype=np.uint8)
for i, row in enumerate(image_input):
    for j, col in enumerate(row):
        pad_image[i +
image_input.shape[0], j +
image_input.shape[1], :] =
image_input[i, j, :]

for i, row in enumerate(pad_image):
    for j, col in enumerate(row):

image_interpolated_nearest_neighbor[i
, j, :] = nearest_neighbors(i, j,
pad_image, matrix_rotation)

for i, row in enumerate(pad_image):
    for j, col in enumerate(row):

image_interpolated_bilinear[i, j, :]
= bilinear_interpolation(i, j,
pad_image, matrix_rotation)
```

تابع ضرب دو ماتریس نیز نوشته شده است:

```
def multiply(input, T):
    out = [0, 0, 1]
    for i in range(0, 3, 1):
        out[i] = input[0] * T[0][i] +
input[1] * T[1][i] + input[2] *
T[2][i]
    return out
```

تابع زیر تبدیل هندسی را به صورت پیکسل به پیکسل انجام می دهد. برای اینکه تبدیل ها حول مرکز انجام بگیرد، مرکز را محاسبه کرده و در ضرب اثر می دهیم:

```
def transform_image(image, matrice):
    image_transformed =
np.empty((image.shape[0],
image.shape[1], 3), dtype=np.uint8)
    center_x = int(image.shape[0] /
2)
    center_y = int(image.shape[1] /
2)
    for i, row in enumerate(image):
        for j, col in enumerate(row):
            pixel_data = image[i, j,
:]
            i2 = i - center_x
            j2 = j - center_y
            input_coords =
np.array([i2, j2, 1])
            out =
multiply(input_coords, matrice)
            i_out = int(out[0] +
center_x)
            j_out = int(out[1] +
center_y)
            # print("i_out=
"+str(i_out)+" j_out= "+str(j_out))
            if 0 < i_out <
image.shape[0] and 0 < j_out <
image.shape[1]:

image_transformed[i_out, j_out, :] =
pixel_data
    return image_transformed
```

تابع زیر درون یابی نزدیک ترین همسایه را انجام می دهد:

```
def nearest_neighbors(i, j, image,
matrix):
    matrix_inverse =
np.linalg.inv(matrix)
    # print("i: " + str(i) + " j: " +
str(j))
    x_max, y_max = image.shape[0] -
1, image.shape[1] - 1
    x, y, _ = matrix_inverse @
np.array([i, j, 1])

    if np.floor(x) == x and
np.floor(y) == y:
        x, y = int(x), int(y)
        return image[x, y]
    if np.abs(np.floor(x) - x) <
np.abs(np.ceil(x) - x):
        x = int(np.floor(x))
    else:
        x = int(np.ceil(x))
    if np.abs(np.floor(y) - y) <
np.abs(np.ceil(y) - y):
        y = int(np.floor(y))
    else:
        y = int(np.ceil(y))
    if x > x_max:
        x = x_max
    if y > y_max:
        y = y_max
    return image[x, y,]
```

و توابع زیر درون یابی دوخطی را انجام می دهند:

```
def bilinear_interpolation(i, j,
image, matrix):
    matrix_inverse =
np.linalg.inv(matrix)
    # print("i: " + str(i) + " j: " +
str(j))
    x_max, y_max = image.shape[0] -
1, image.shape[1] - 1
    x, y, _ = matrix_inverse @
```

```
np.array([i, j, 1])

    # we dont need to interpolate
this is very good
    if np.floor(x) == x and
np.floor(y) == y:
        x, y = int(x), int(y)
        return image[x, y]
    # we are gooing to interpolate ha
ha
    x_rounded_up = int(ceil(x))
    x_rounded_down = int(floor(x))
    y_rounded_up = int(ceil(y))
    y_rounded_down = int(floor(y))

    ratio_x = x - x_rounded_down
    ratio_y = y - y_rounded_down

    if x_rounded_up > x_max:
        x_rounded_up = x_max
    if y_rounded_up > y_max:
        y_rounded_up = y_max

    if x_rounded_down > x_max:
        x_rounded_down = x_max
    if y_rounded_down > y_max:
        y_rounded_down = y_max

    interpolate_x1 =
interpolate(get_array_value(x_rounded
_down, y_rounded_down, image),

get_array_value(x_rounded_up,
y_rounded_down, image),

ratio_x)
    interpolate_x2 =
interpolate(get_array_value(x_rounded
_down, y_rounded_up, image),

get_array_value(x_rounded_up,
y_rounded_up, image),

ratio_x)
    interpolate_y =
interpolate(interpolate_x1,
interpolate_x2, ratio_y)

    return interpolate_y


def interpolate(first_value: float,
second_value: float, ratio: float) ->
float:
    return first_value * (1 - ratio)
+ second_value * ratio


def get_array_value(x: int, y: int,
```

```
array: np.ndarray):
    return array[x, y]
```

و در نهایت تصاویر به دست آمده را نمایش داده و ذخیره می کنیم:

```
plt.figure(figsize=(5, 5))
plt.imsave('padinput.jpg', pad_image)
plt.imsave('output_raw.jpg',
image_output_raw)
plt.imsave('output_nearest_neighbor.j
pg',
image_interpolated_nearest_neighbor)
plt.imsave('output_bilinear.jpg',
image_interpolated_bilinear)
plt.imshow(image_output_raw)
plt.imshow(image_interpolated_nearest
_neighbor)
plt.imshow(image_interpolated_bilinea
r)
```