GEBZE
TEKNİK ÜNİVERSİTESİ

CSE464 HOMEWORK 1 REPORT

SÜLEYMAN KORAMAZ
1901042615

1. INTRODUCTION

The purpose of this report is to present and compare three different techniques for image transformation: forward mapping, backward mapping without interpolation, and backward mapping with bilinear interpolation. These techniques are commonly employed in computer graphics and image processing to achieve operations such as scaling, rotation, shearing (vertical and horizontal), and zooming.

2. PARTS

2.1 FORWARD MAPPING

Forward mapping involves iterating over the pixels of the output image and determining the corresponding pixel in the input image. This technique is implemented by dividing the output image into a grid and mapping each pixel to its corresponding position in the input image.

2.2 BACKWARD MAPPING WITHOUT INTERPOLATION

Backward mapping without interpolation involves iterating over the pixels of the input image and mapping each pixel to its corresponding position in the output image. This technique does not use interpolation to estimate pixel values at non-grid points.

2.3 BACKWARD MAPPING WITH BILINEAR INTERPOLATION

Backward mapping with bilinear interpolation incorporates bilinear interpolation to estimate pixel values at non-grid points during the mapping process. This interpolation method enhances the accuracy of the transformed image.

INTERPOLATION

Upper left neighbor = y2, x2 => p11

Upper right neighbor = y1, x2 => p12

Lower left neighbor = y2, x1 => p21

Lower right neighbor = y1, x1 => p22

Formula:

(1 - (x - x1)) * (1 - (y - y1)) * p11 + (x - x1) * (1 - (y - y1)) * p12 + (1 - (x - x1)) * (y - y1) * p21 + (x - x1) * (y - y1) * p22

3. RESULTS

3.1 SCALING

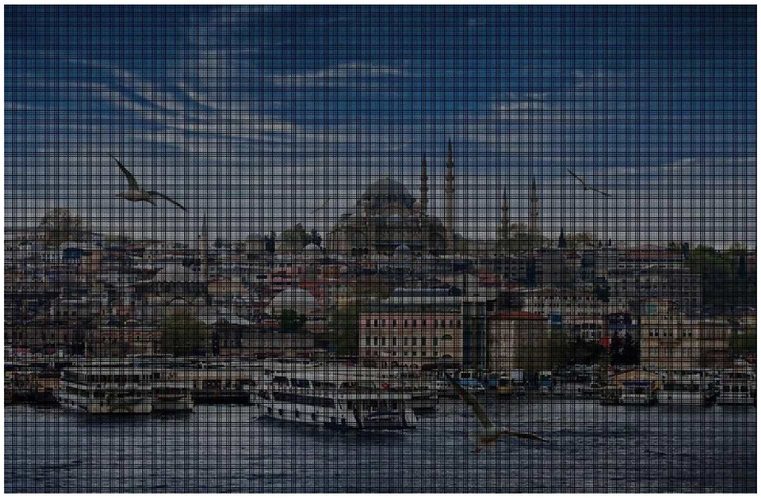
General scaling formula with affine matrix:

(output_x = input_x * scale_rate)

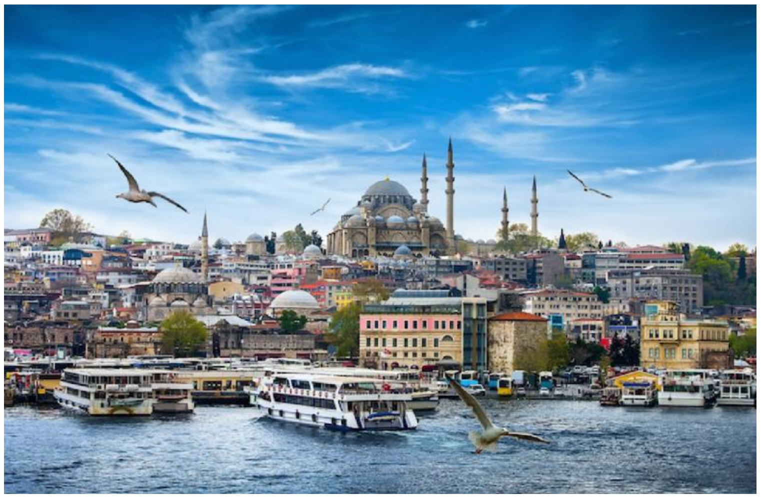
(output_y = input_y * scale_rate)

In forward mapping, there is an obvious pixel lose. We define the output pixels depending on the input pixels but if there will any float number after the calculations, we assume that they are the closest integers. For example, this cause that we have 100x100 image. If we scale it x2, there will 200x200 image, but we have only 10.000 pixel. So, 1000 pixel will be empty. On the other hand, if we scale it x1/2, there will 50x50 image, but we have more pixels. So, there is overlaps.

Forward Mapping (Enlarge image):



Backward Mapping (Enlarge image):



3.1 SCALING

In backward mapping, image became better quality than forward mapping. But still, there is issues. We have no empty pixel in output image, but we did not place output pixels exactly with the input pixels. We are still rounding float numbers to int. So, we used to some of the pixels more than one time.

Forward Mapping (Reduce Image):

Backward Mapping (Reduce Image):

3.1 SCALING

When we use bilinear interpolation in backward mapping, we obtain results closest to the original image. Because, for every pixel we are making interpolation process, and we get very similar color to the input pixel.

Backward Mapping with Bilinear Interpolation (Enlarge Image):

Backward Mapping with Bilinear Interpolation (Reduce Image):

3.2 ROTATION

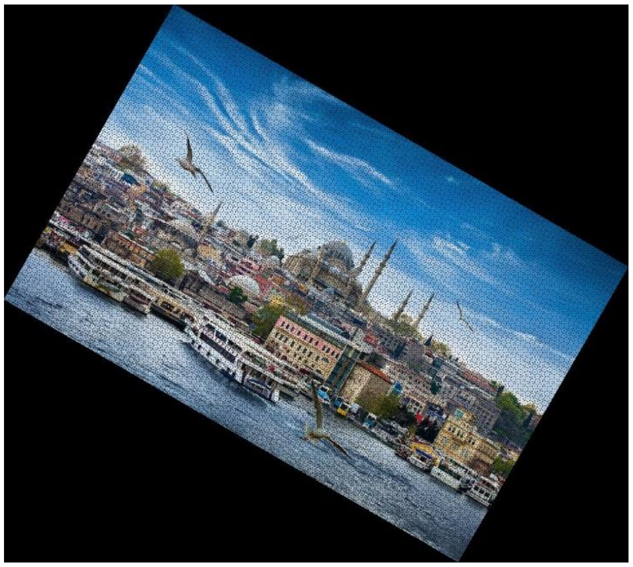
General rotating formula with affine matrix:

(output_x = input_x * cos(angle) – input_y * sin(angle))

(output_y = input_x*sin(angle) + input_y*cos(angle))

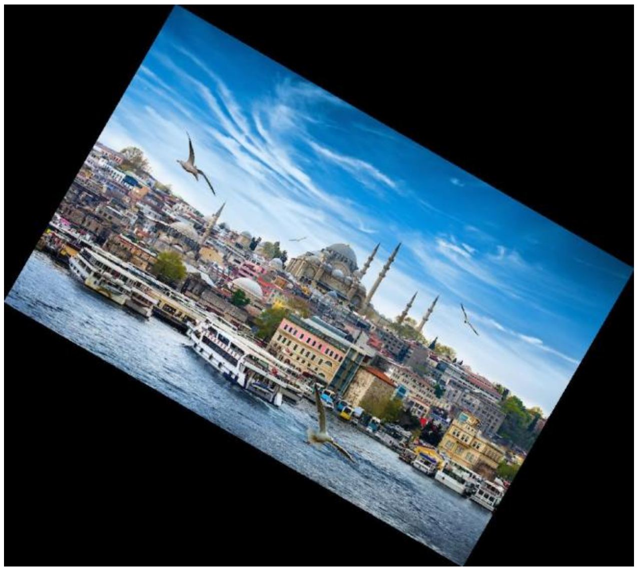
In forward mapping, we have same issue with scaling. There are more pixel in output image than input image. In output, there is empty pixels.

Forward Mapping:



Backward mapping is same with scaling too. Better quality image but there is a loss of detail compared to the original image.

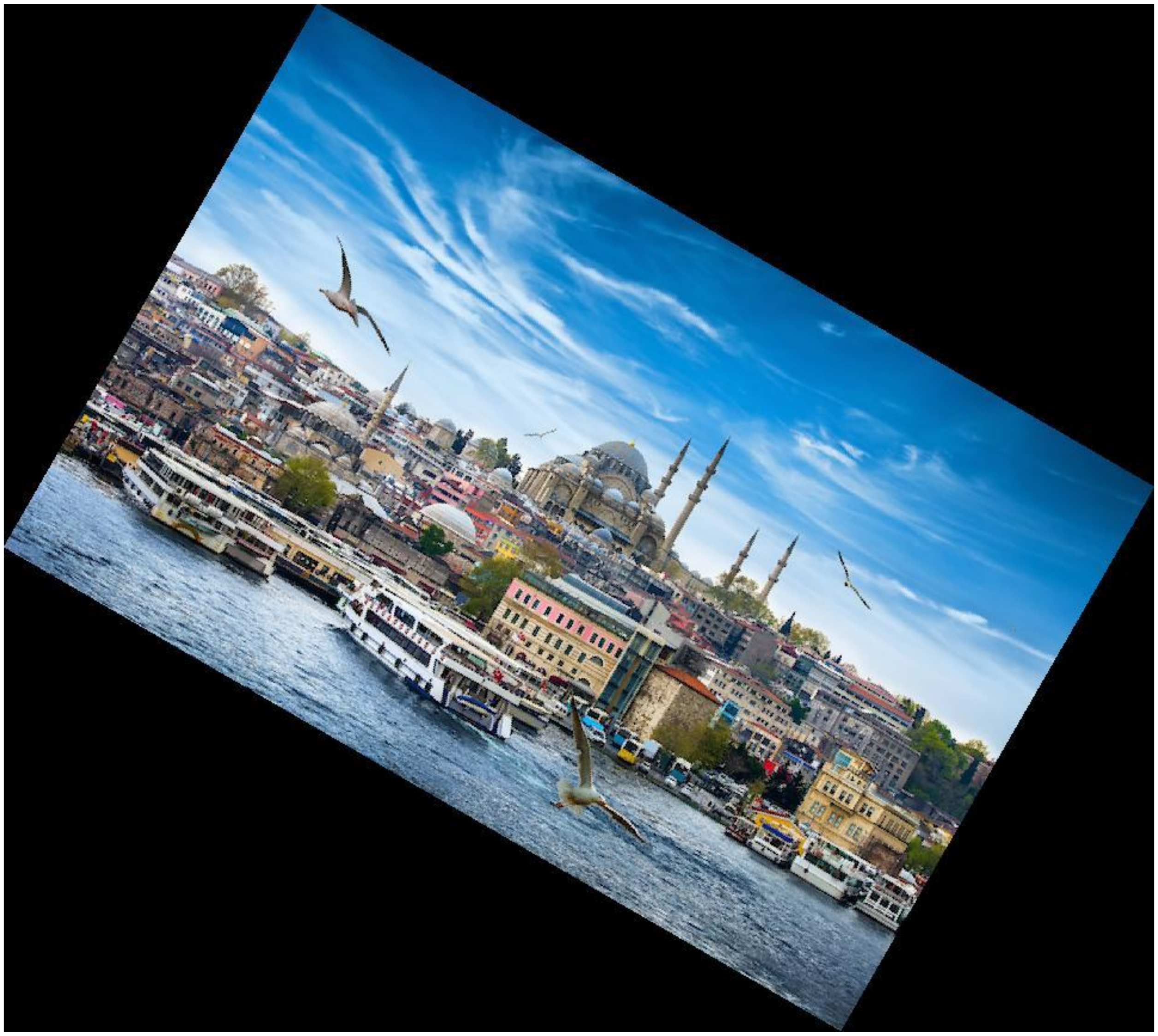
Backward Mapping:



3.2 ROTATION

When we do interpolation, image quality became very similar to original image.

Backward Mapping with Bilinear Interpolation:



3.3 SHEARING (VERTICAL AND HORIZONTAL)

General horizontal shearing formula with affine matrix:

(output_x = input_x)

(output_y = sh*input_x + input_y)

Vertical shearing:

(output_x = input_x + sh*input_y)

(output_y = input_y)

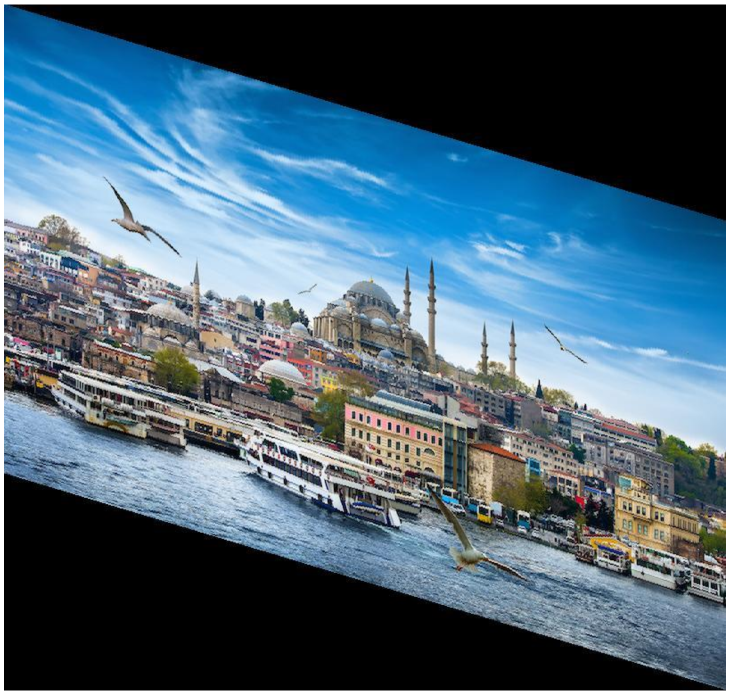
In shearing, three methods have almost same results. But still there is small quality difference between them. Backward mapping with interpolation is still best option and most similar to the original image.

Forward Mapping (Vertical):

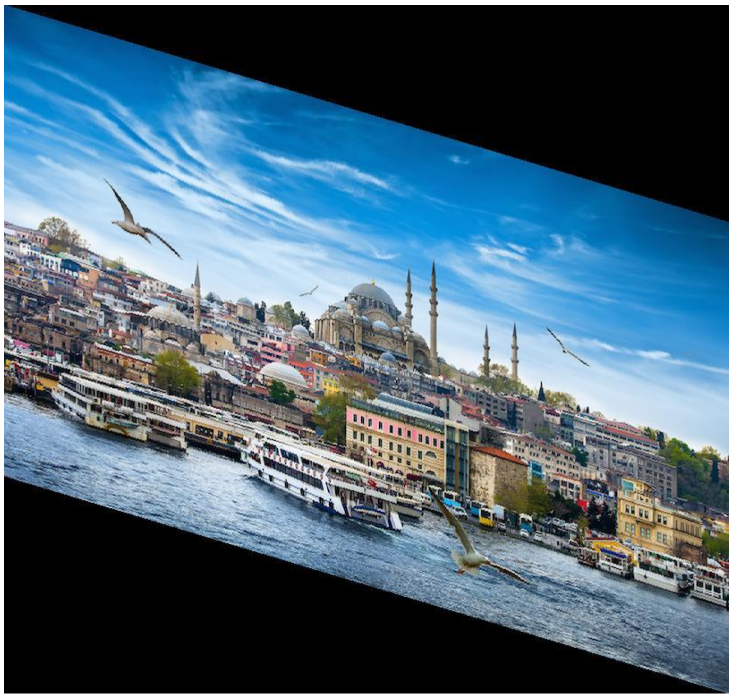
Backward Mapping (Vertical):

3.3 SHEARING (VERTICAL AND HORIZONTAL)

Forward Mapping (Horizontal):

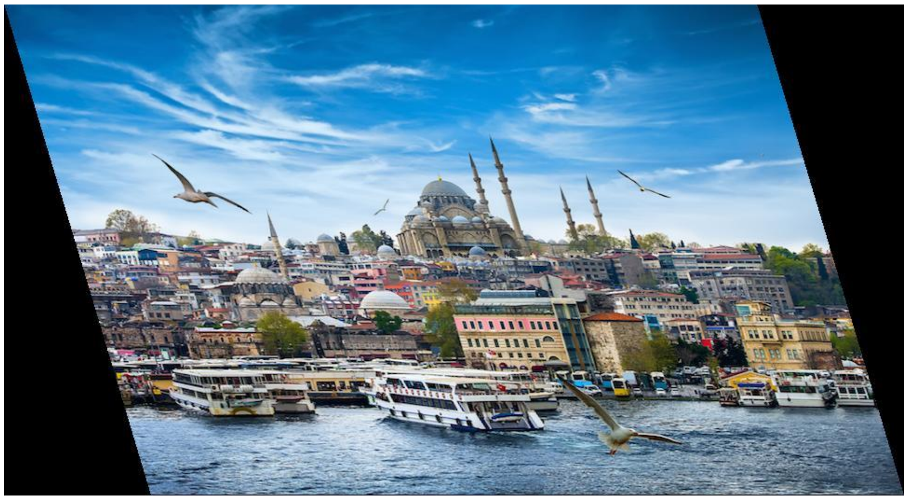


Backward Mapping (Horizontal):

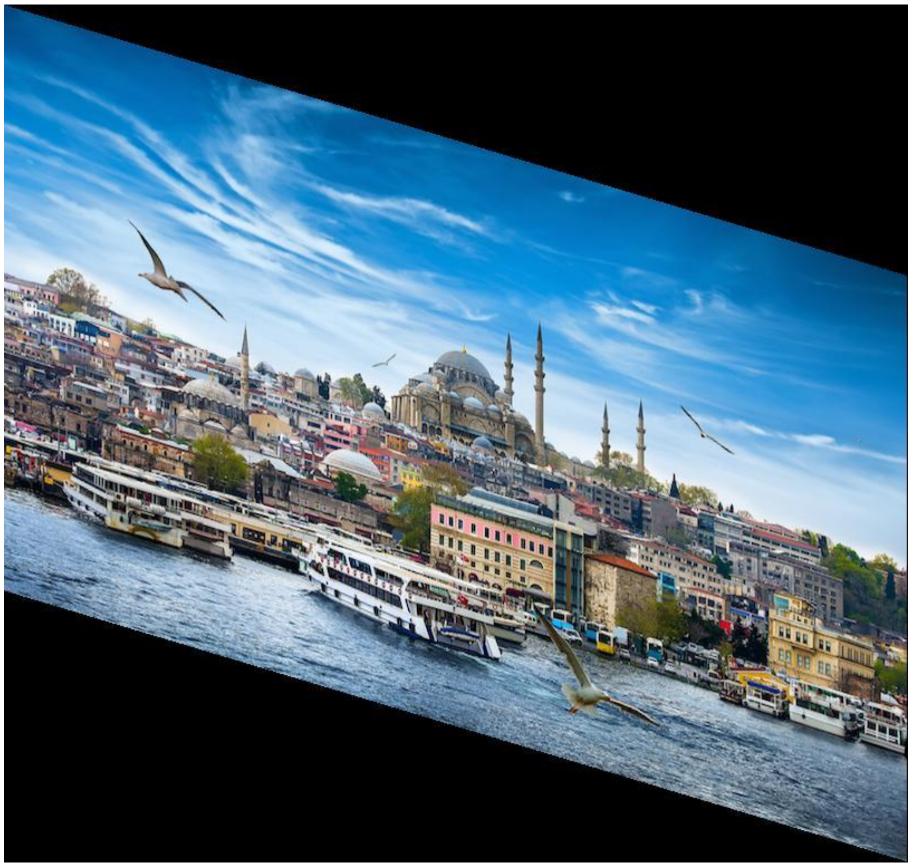


3.3 SHEARING (VERTICAL AND HORIZONTAL)

Backward Mapping with Bilinear Interpolation (Vertical):



Backward Mapping with Bilinear Interpolation (Horizontal):



3.4 ZOOMING

General zooming formula with affine matrix:

(output_x = input_x / zoom_rate)

(output_y = input_y / zoom_rate)

Zoom is same with the others.

Forward Mapping:

Backward Mapping:

3.4 ZOOMING

Backward Mapping with Bilinear Interpolation:

5. DISCUSSION

Scaling: The forward mapping technique for scaling demonstrates a straightforward approach, but it suffers from pixelation and loss of details, especially when upscaling an image

Rotating: All three methods successfully achieve the rotation operation, but noticeable differences emerge in the preservation of image details.

Shearing: Both forward and backward mapping without interpolation exhibit distortions in sheared images, with visible artifacts along the sheared axis. Bilinear interpolation significantly improves the results, especially in regions far from the origin.

Zooming: Forward mapping introduces noticeable pixelation during zooming, degrading visual quality, especially in high-resolution images. Backward mapping without interpolation improves on pixelation but introduces jagged edges. Backward mapping with bilinear interpolation provides the best compromise, producing smoother and more visually appealing results.

6. CONCLUSION

Forward mapping is simple and intuitive but may suffer from pixelation and quality degradation, especially in complex transformations.

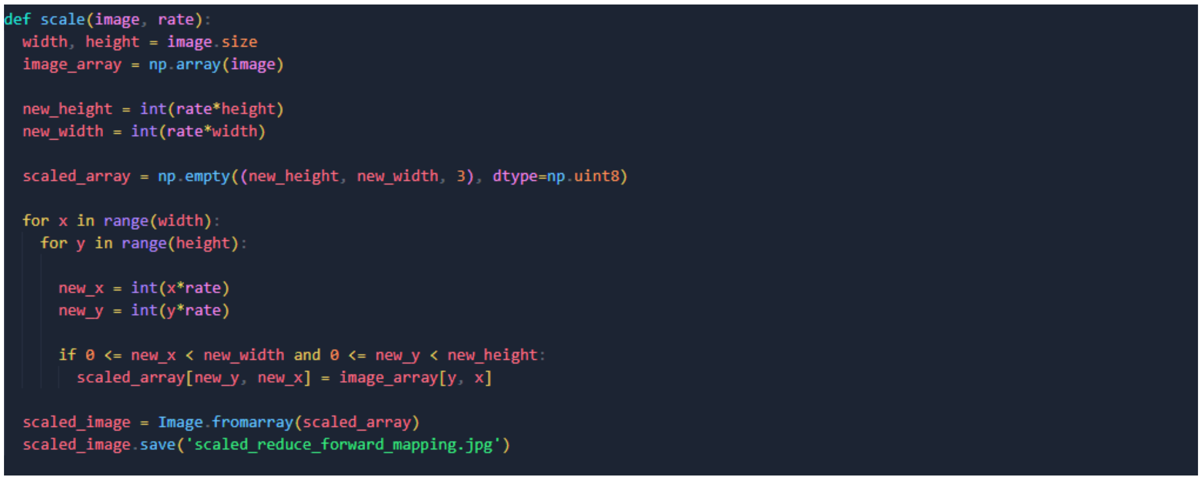
Backward mapping without interpolation avoids pixelation but struggles with smoothness, leading to jagged edges in transformed images.

Backward mapping with bilinear interpolation provides the best compromise, enhancing visual quality through interpolation, while still maintaining reasonable computational efficiency.

6. CODE

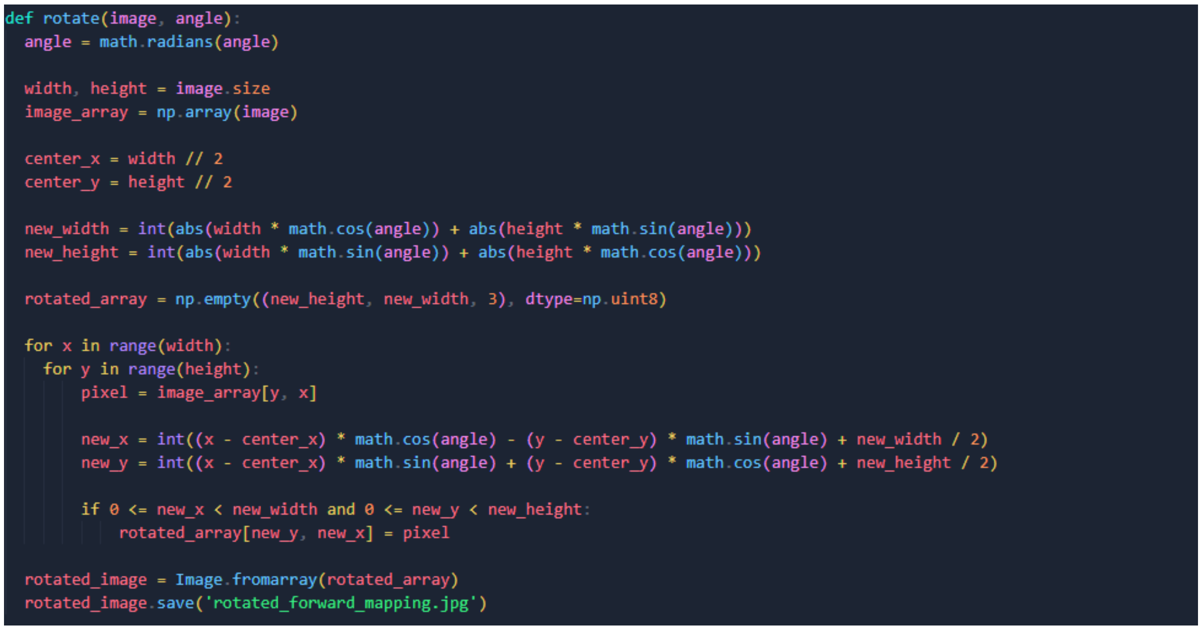
FORWARD

Scaling:



```
def scale(image, rate):
  width, height = image.size
  image_array = np.array(image)

  new_height = int(rate*height)
  new_width = int(rate*width)

  scaled_array = np.empty((new_height, new_width, 3), dtype=np.uint8)

  for x in range(width):
    for y in range(height):

      new_x = int(x*rate)
      new_y = int(y*rate)

      if 0 <= new_x < new_width and 0 <= new_y < new_height:
        scaled_array[new_y, new_x] = image_array[y, x]

  scaled_image = Image.fromarray(scaled_array)
  scaled_image.save('scaled_reduce_forward_mapping.jpg')
```

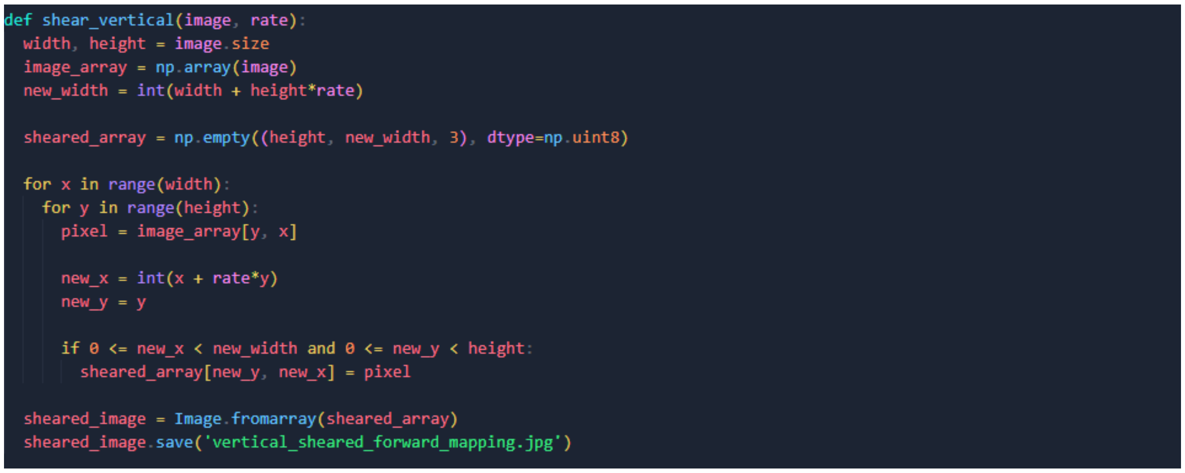
Rotating:



```
def rotate(image, angle):
  angle = math.radians(angle)

  width, height = image.size
  image_array = np.array(image)

  center_x = width // 2
  center_y = height // 2

  new_width = int(abs(width * math.cos(angle)) + abs(height * math.sin(angle)))
  new_height = int(abs(width * math.sin(angle)) + abs(height * math.cos(angle)))

  rotated_array = np.empty((new_height, new_width, 3), dtype=np.uint8)

  for x in range(width):
    for y in range(height):
        pixel = image_array[y, x]

        new_x = int((x - center_x) * math.cos(angle) - (y - center_y) * math.sin(angle) + new_width / 2)
        new_y = int((x - center_x) * math.sin(angle) + (y - center_y) * math.cos(angle) + new_height / 2)

        if 0 <= new_x < new_width and 0 <= new_y < new_height:
            rotated_array[new_y, new_x] = pixel

  rotated_image = Image.fromarray(rotated_array)
  rotated_image.save('rotated_forward_mapping.jpg')
```

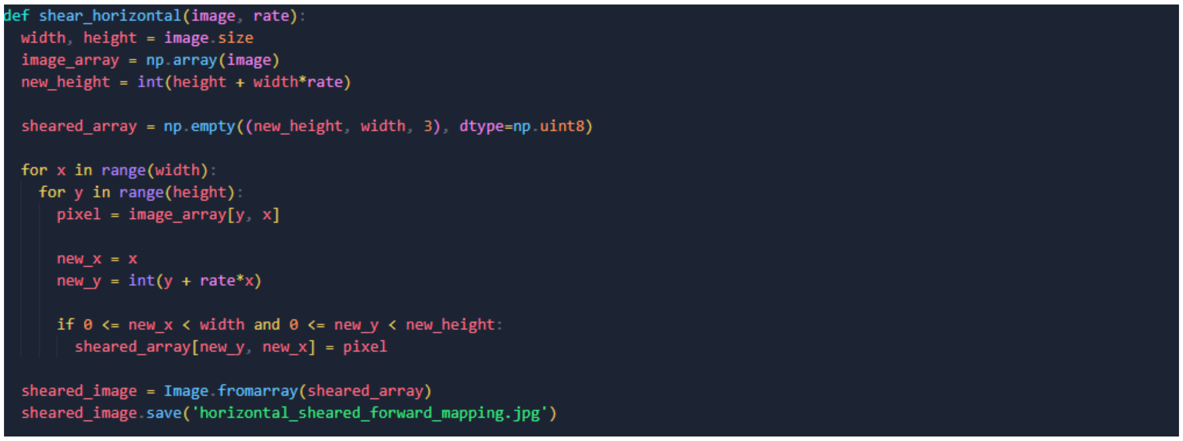
FORWARD

Shear (Vertical):



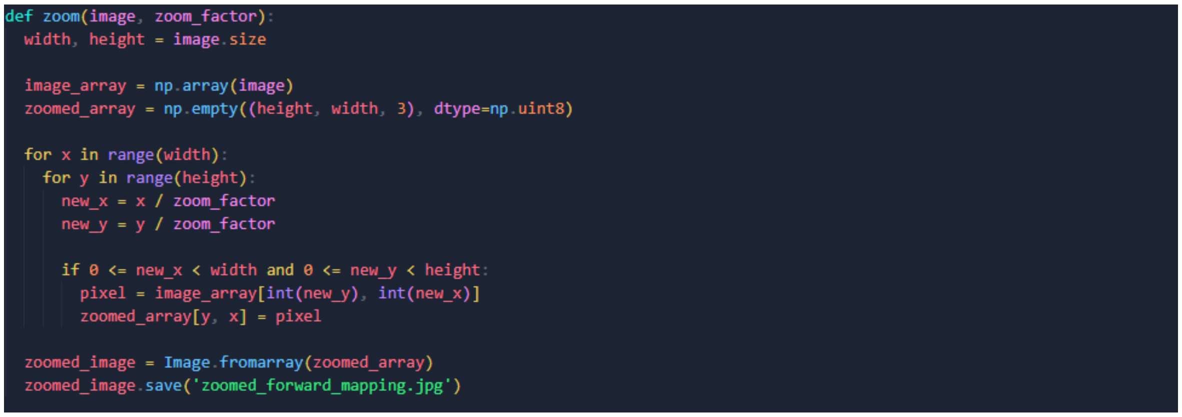
```python
def shear_vertical(image, rate):
  width, height = image.size
  image_array = np.array(image)
  new_width = int(width + height*rate)

  sheared_array = np.empty((height, new_width, 3), dtype=np.uint8)

  for x in range(width):
    for y in range(height):
      pixel = image_array[y, x]

      new_x = int(x + rate*y)
      new_y = y

      if 0 <= new_x < new_width and 0 <= new_y < height:
        sheared_array[new_y, new_x] = pixel

  sheared_image = Image.fromarray(sheared_array)
  sheared_image.save('vertical_sheared_forward_mapping.jpg')
```

Shear (Horizontal):



```python
def shear_horizontal(image, rate):
  width, height = image.size
  image_array = np.array(image)
  new_height = int(height + width*rate)

  sheared_array = np.empty((new_height, width, 3), dtype=np.uint8)

  for x in range(width):
    for y in range(height):
      pixel = image_array[y, x]

      new_x = x
      new_y = int(y + rate*x)

      if 0 <= new_x < width and 0 <= new_y < new_height:
        sheared_array[new_y, new_x] = pixel

  sheared_image = Image.fromarray(sheared_array)
  sheared_image.save('horizontal_sheared_forward_mapping.jpg')
```

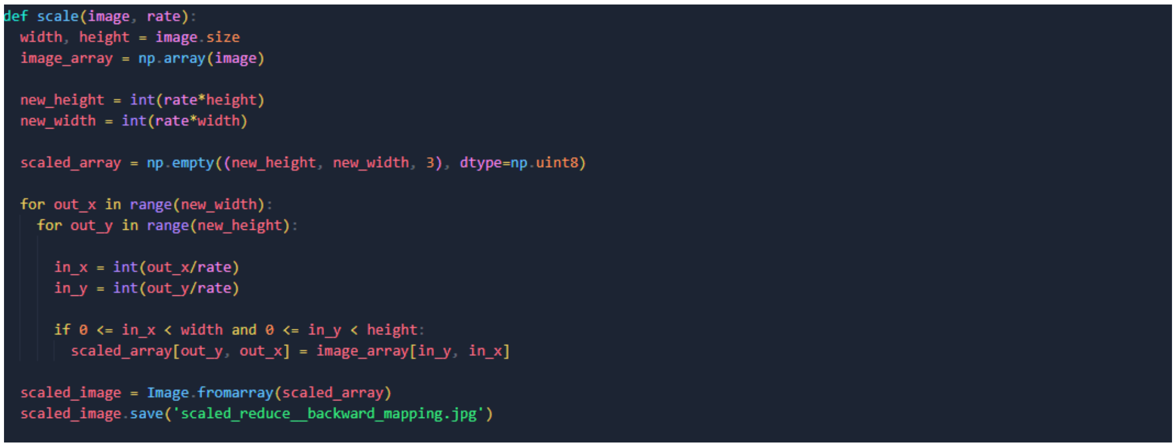
Zoom:



```python
def zoom(image, zoom_factor):
  width, height = image.size

  image_array = np.array(image)
  zoomed_array = np.empty((height, width, 3), dtype=np.uint8)

  for x in range(width):
    for y in range(height):
      new_x = x / zoom_factor
      new_y = y / zoom_factor

      if 0 <= new_x < width and 0 <= new_y < height:
        pixel = image_array[int(new_y), int(new_x)]
        zoomed_array[y, x] = pixel

  zoomed_image = Image.fromarray(zoomed_array)
  zoomed_image.save('zoomed_forward_mapping.jpg')
```

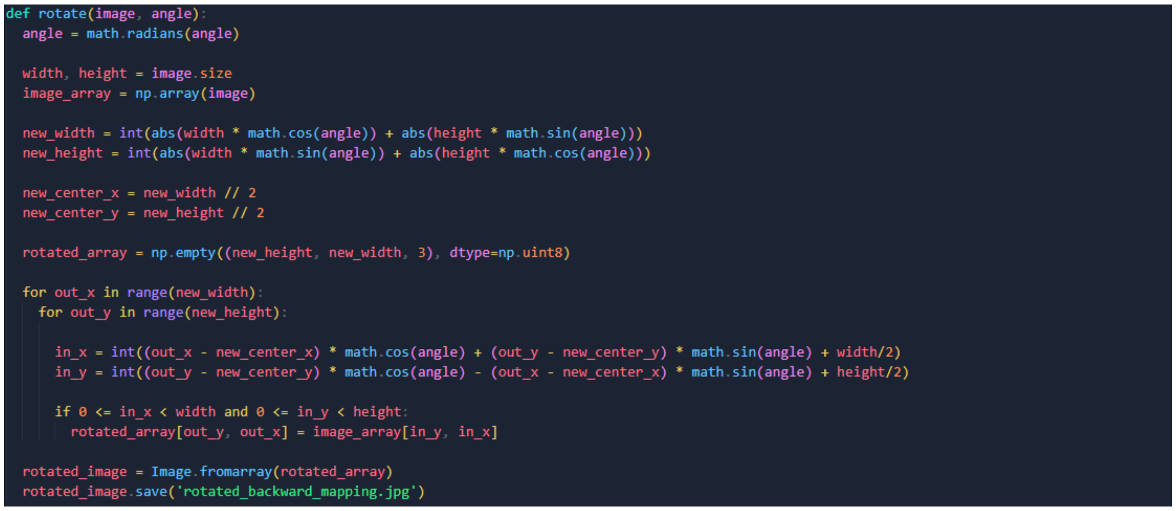
BACKWARD WITHOUT INTERPOLITION

Scaling:



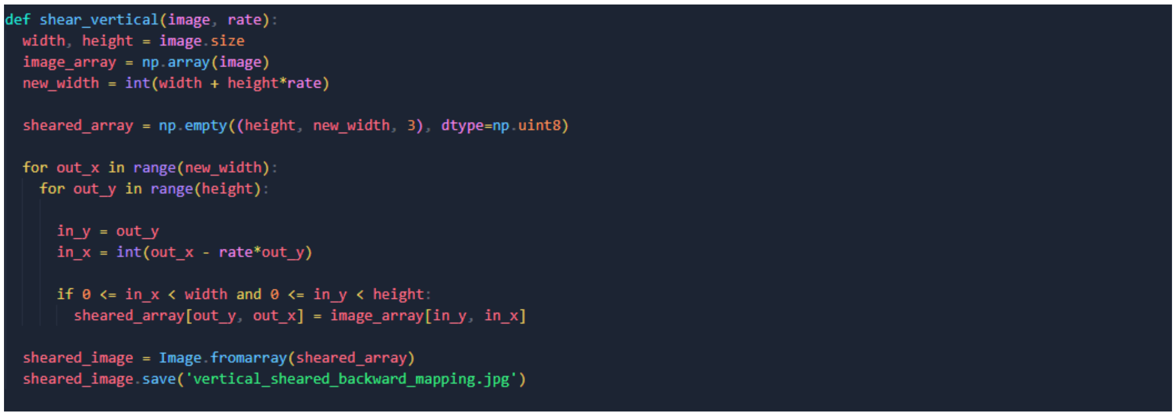
```python
def scale(image, rate):
  width, height = image.size
  image_array = np.array(image)

  new_height = int(rate*height)
  new_width = int(rate*width)

  scaled_array = np.empty((new_height, new_width, 3), dtype=np.uint8)

  for out_x in range(new_width):
    for out_y in range(new_height):

      in_x = int(out_x/rate)
      in_y = int(out_y/rate)

      if 0 <= in_x < width and 0 <= in_y < height:
        scaled_array[out_y, out_x] = image_array[in_y, in_x]

  scaled_image = Image.fromarray(scaled_array)
  scaled_image.save('scaled_reduce__backward_mapping.jpg')
```

Rotating:



```python
def rotate(image, angle):
  angle = math.radians(angle)

  width, height = image.size
  image_array = np.array(image)

  new_width = int(abs(width * math.cos(angle)) + abs(height * math.sin(angle)))
  new_height = int(abs(width * math.sin(angle)) + abs(height * math.cos(angle)))

  new_center_x = new_width // 2
  new_center_y = new_height // 2

  rotated_array = np.empty((new_height, new_width, 3), dtype=np.uint8)

  for out_x in range(new_width):
    for out_y in range(new_height):

      in_x = int((out_x - new_center_x) * math.cos(angle) + (out_y - new_center_y) * math.sin(angle) + width/2)
      in_y = int((out_y - new_center_y) * math.cos(angle) - (out_x - new_center_x) * math.sin(angle) + height/2)

      if 0 <= in_x < width and 0 <= in_y < height:
        rotated_array[out_y, out_x] = image_array[in_y, in_x]

  rotated_image = Image.fromarray(rotated_array)
  rotated_image.save('rotated_backward_mapping.jpg')
```

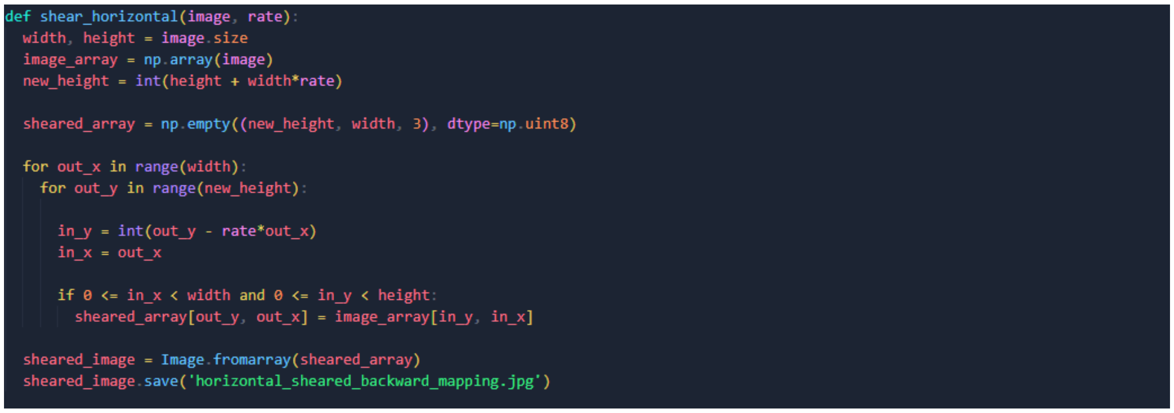
Shear (Vertical):



```python
def shear_vertical(image, rate):
  width, height = image.size
  image_array = np.array(image)
  new_width = int(width + height*rate)

  sheared_array = np.empty((height, new_width, 3), dtype=np.uint8)

  for out_x in range(new_width):
    for out_y in range(height):

      in_y = out_y
      in_x = int(out_x - rate*out_y)

      if 0 <= in_x < width and 0 <= in_y < height:
        sheared_array[out_y, out_x] = image_array[in_y, in_x]

  sheared_image = Image.fromarray(sheared_array)
  sheared_image.save('vertical_sheared_backward_mapping.jpg')
```

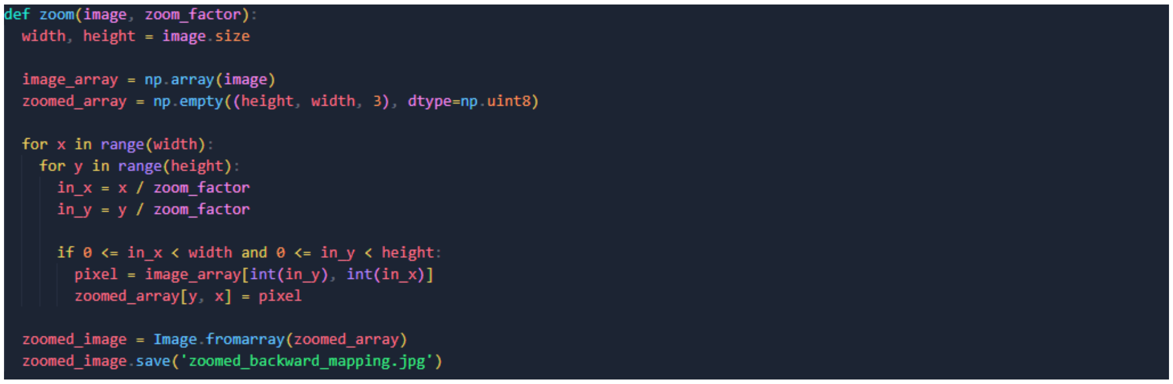
BACKWARD WITHOUT INTERPOLITION

Shear (Horizontal):



```python
def shear_horizontal(image, rate):
  width, height = image.size
  image_array = np.array(image)
  new_height = int(height + width*rate)

  sheared_array = np.empty((new_height, width, 3), dtype=np.uint8)

  for out_x in range(width):
    for out_y in range(new_height):

      in_y = int(out_y - rate*out_x)
      in_x = out_x

      if 0 <= in_x < width and 0 <= in_y < height:
        sheared_array[out_y, out_x] = image_array[in_y, in_x]

  sheared_image = Image.fromarray(sheared_array)
  sheared_image.save('horizontal_sheared_backward_mapping.jpg')
```

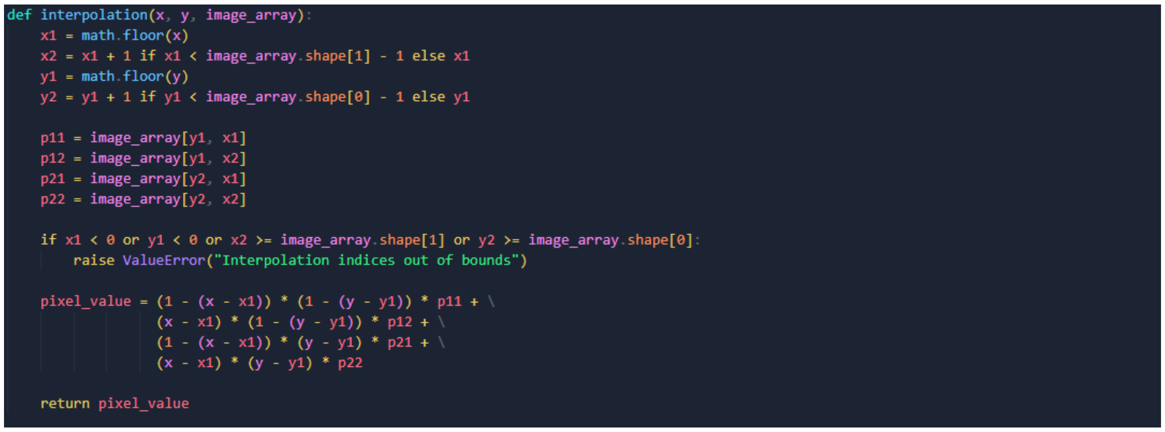
Zoom:



```python
def zoom(image, zoom_factor):
  width, height = image.size

  image_array = np.array(image)
  zoomed_array = np.empty((height, width, 3), dtype=np.uint8)

  for x in range(width):
    for y in range(height):
      in_x = x / zoom_factor
      in_y = y / zoom_factor

      if 0 <= in_x < width and 0 <= in_y < height:
        pixel = image_array[int(in_y), int(in_x)]
        zoomed_array[y, x] = pixel

  zoomed_image = Image.fromarray(zoomed_array)
  zoomed_image.save('zoomed_backward_mapping.jpg')
```

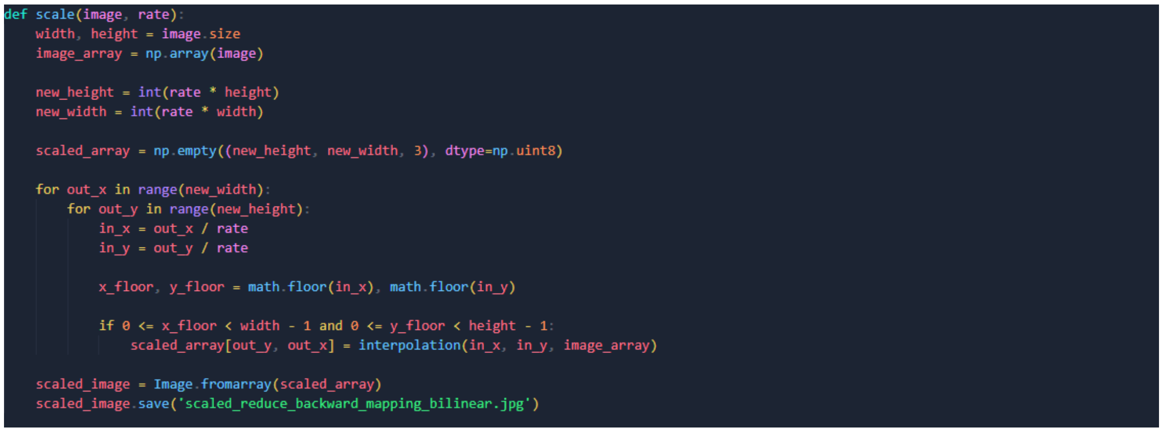
BACKWARD WITH INTERPOLITION

Interpolition:



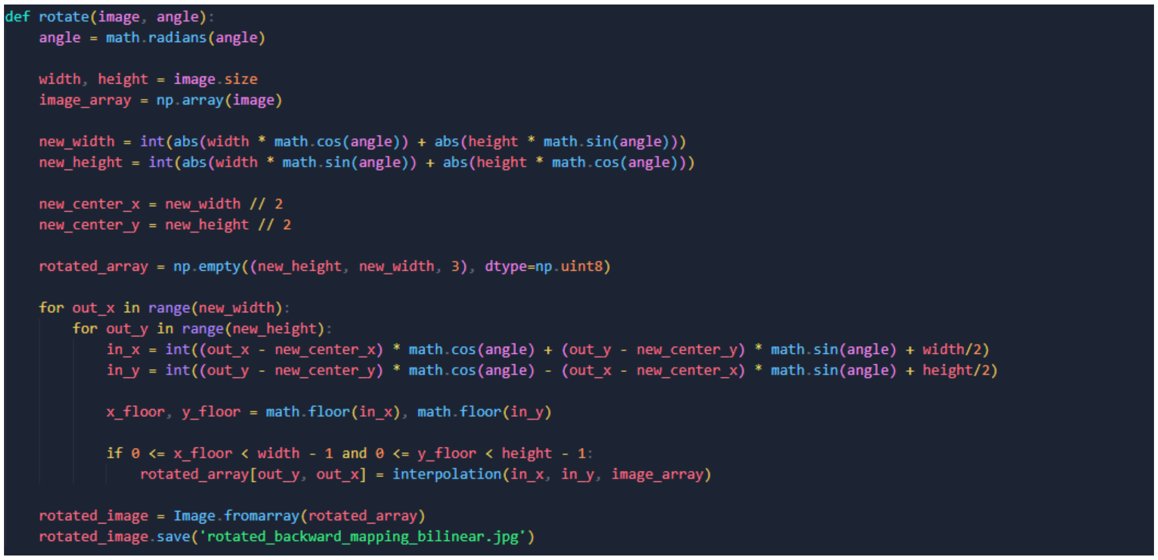
```python
def interpolation(x, y, image_array):
    x1 = math.floor(x)
    x2 = x1 + 1 if x1 < image_array.shape[1] - 1 else x1
    y1 = math.floor(y)
    y2 = y1 + 1 if y1 < image_array.shape[0] - 1 else y1

    p11 = image_array[y1, x1]
    p12 = image_array[y1, x2]
    p21 = image_array[y2, x1]
    p22 = image_array[y2, x2]

    if x1 < 0 or y1 < 0 or x2 >= image_array.shape[1] or y2 >= image_array.shape[0]:
        raise ValueError("Interpolation indices out of bounds")

    pixel_value = (1 - (x - x1)) * (1 - (y - y1)) * p11 + \
                  (x - x1) * (1 - (y - y1)) * p12 + \
                  (1 - (x - x1)) * (y - y1) * p21 + \
                  (x - x1) * (y - y1) * p22

    return pixel_value
```

Scaling:



```python
def scale(image, rate):
    width, height = image.size
    image_array = np.array(image)

    new_height = int(rate * height)
    new_width = int(rate * width)

    scaled_array = np.empty((new_height, new_width, 3), dtype=np.uint8)

    for out_x in range(new_width):
        for out_y in range(new_height):
            in_x = out_x / rate
            in_y = out_y / rate

            x_floor, y_floor = math.floor(in_x), math.floor(in_y)

            if 0 <= x_floor < width - 1 and 0 <= y_floor < height - 1:
                scaled_array[out_y, out_x] = interpolation(in_x, in_y, image_array)

    scaled_image = Image.fromarray(scaled_array)
    scaled_image.save('scaled_reduce_backward_mapping_bilinear.jpg')
```

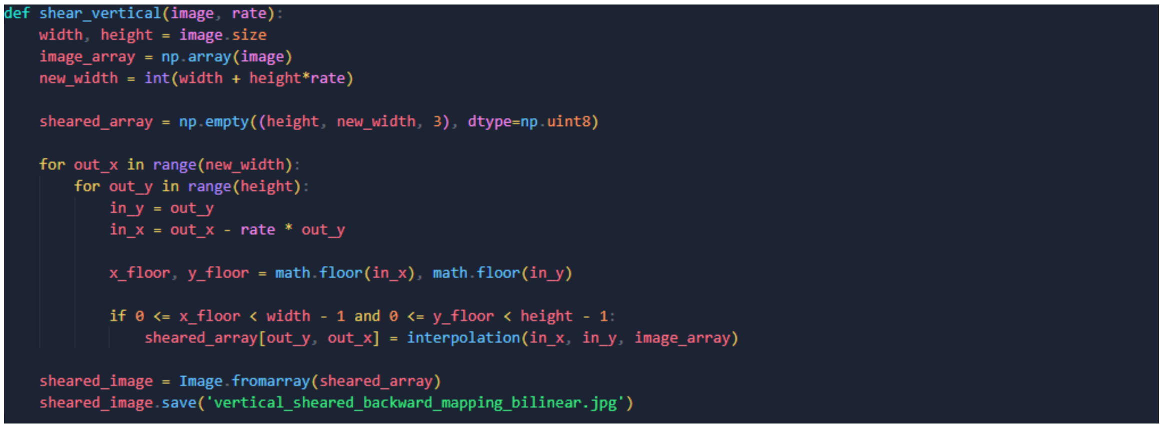
Rotating:



```python
def rotate(image, angle):
    angle = math.radians(angle)

    width, height = image.size
    image_array = np.array(image)

    new_width = int(abs(width * math.cos(angle)) + abs(height * math.sin(angle)))
    new_height = int(abs(width * math.sin(angle)) + abs(height * math.cos(angle)))

    new_center_x = new_width // 2
    new_center_y = new_height // 2

    rotated_array = np.empty((new_height, new_width, 3), dtype=np.uint8)

    for out_x in range(new_width):
        for out_y in range(new_height):
            in_x = int((out_x - new_center_x) * math.cos(angle) + (out_y - new_center_y) * math.sin(angle) + width/2)
            in_y = int((out_y - new_center_y) * math.cos(angle) - (out_x - new_center_x) * math.sin(angle) + height/2)

            x_floor, y_floor = math.floor(in_x), math.floor(in_y)

            if 0 <= x_floor < width - 1 and 0 <= y_floor < height - 1:
                rotated_array[out_y, out_x] = interpolation(in_x, in_y, image_array)

    rotated_image = Image.fromarray(rotated_array)
    rotated_image.save('rotated_backward_mapping_bilinear.jpg')
```

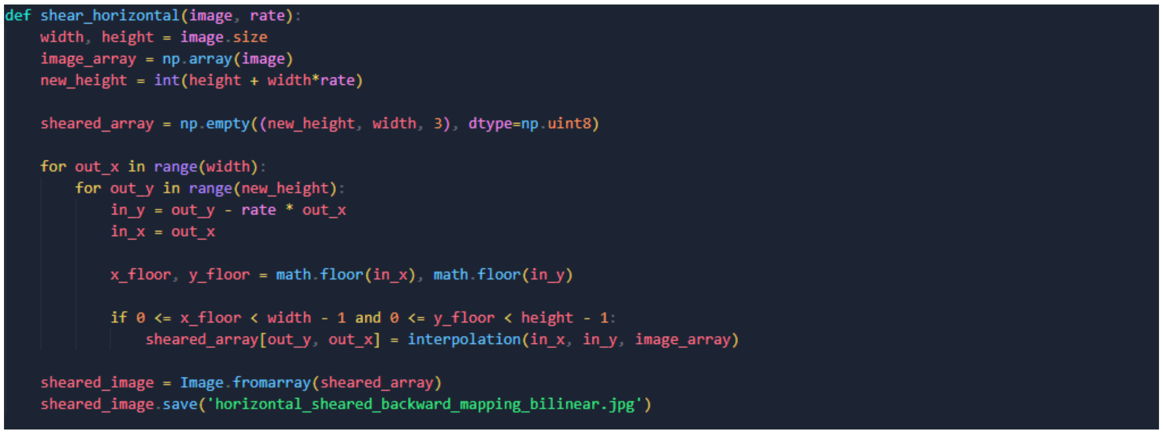
BACKWARD WITH INTERPOLITION

Shear (Vertical):



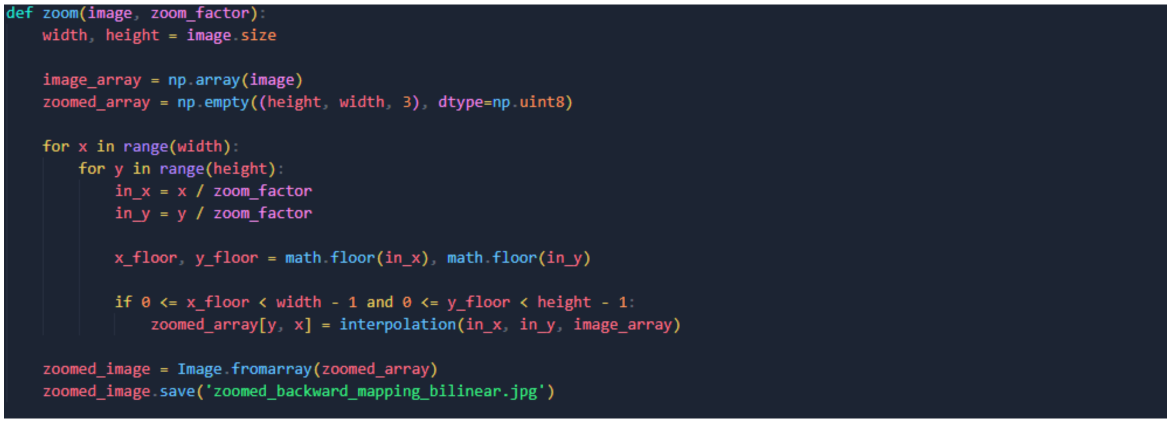
```python
def shear_vertical(image, rate):
    width, height = image.size
    image_array = np.array(image)
    new_width = int(width + height*rate)

    sheared_array = np.empty((height, new_width, 3), dtype=np.uint8)

    for out_x in range(new_width):
        for out_y in range(height):
            in_y = out_y
            in_x = out_x - rate * out_y

            x_floor, y_floor = math.floor(in_x), math.floor(in_y)

            if 0 <= x_floor < width - 1 and 0 <= y_floor < height - 1:
                sheared_array[out_y, out_x] = interpolation(in_x, in_y, image_array)

    sheared_image = Image.fromarray(sheared_array)
    sheared_image.save('vertical_sheared_backward_mapping_bilinear.jpg')
```

Shear (Horizontal):



```python
def shear_horizontal(image, rate):
    width, height = image.size
    image_array = np.array(image)
    new_height = int(height + width*rate)

    sheared_array = np.empty((new_height, width, 3), dtype=np.uint8)

    for out_x in range(width):
        for out_y in range(new_height):
            in_y = out_y - rate * out_x
            in_x = out_x

            x_floor, y_floor = math.floor(in_x), math.floor(in_y)

            if 0 <= x_floor < width - 1 and 0 <= y_floor < height - 1:
                sheared_array[out_y, out_x] = interpolation(in_x, in_y, image_array)

    sheared_image = Image.fromarray(sheared_array)
    sheared_image.save('horizontal_sheared_backward_mapping_bilinear.jpg')
```

Zoom:



```python
def zoom(image, zoom_factor):
    width, height = image.size

    image_array = np.array(image)
    zoomed_array = np.empty((height, width, 3), dtype=np.uint8)

    for x in range(width):
        for y in range(height):
            in_x = x / zoom_factor
            in_y = y / zoom_factor

            x_floor, y_floor = math.floor(in_x), math.floor(in_y)

            if 0 <= x_floor < width - 1 and 0 <= y_floor < height - 1:
                zoomed_array[y, x] = interpolation(in_x, in_y, image_array)

    zoomed_image = Image.fromarray(zoomed_array)
    zoomed_image.save('zoomed_backward_mapping_bilinear.jpg')
```