```python
import numpy as np
import math


def convolve(in_arr, kernel, stride=1, padding=0, padding_mode="edge"):
    """
    Convolve a 2d array with a giving kernel.
    Args:
        in_arr (array): input array.
        kernel (array): kernel.
        stride (int, optional): the stride of moving windows. Tuple of (int, int) can be given to control the vertical stride and the horizontal stride independently. Defaults to 1.
        padding (int, optional): size of padding for the input array. Tuple of ((int,int),(int,int)) can be given to control top, bottom, left, and right padding size independently. Defaults to 0.
        padding_mode (str, optional): padding type. More info: https://numpy.org/doc/stable/reference/generated/numpy.pad.html. Defaults to "edge".

    Returns:
        [array]: convolved array.
    """
    # Expand scaler values to tuple
    if np.isscalar(padding):
        padding = (
            (padding, padding),  # (top, bottom)
            (padding, padding),  # (left, right)
        )
    if np.isscalar(stride):
        stride = (stride, stride)  # (verticle stride, horizontal stride)

    # Calculate output array size
    out_arr_height = (
        math.floor((in_arr.shape[0] + np.sum(padding[0]) - kernel.shape[0]) / stride[0])
        + 1
    )
    out_arr_width = (
        math.floor((in_arr.shape[1] + np.sum(padding[1]) - kernel.shape[1]) / stride[1])
        + 1
    )

    # Add padding to input array
    in_arr = np.pad(in_arr, pad_width=padding, mode=padding_mode)

    # Create output array
    out_arr = np.zeros((out_arr_height, out_arr_width))

    # Perform convolution between input array and kernel
    for h in range(0, out_arr_height):
        for w in range(0, out_arr_width):
            out_arr[h, w] = np.sum(
                in_arr[
                    h * stride[0] : h * stride[0] + kernel.shape[0],
                    w * stride[1] : w * stride[1] + kernel.shape[1],
                ]
                * kernel
            )

    return out_arr


def roberts(img):
    """Perform Roberts Cross filter on a given grayscale image.

    Args:
        img (array): a given grayscale image.

    Returns:
        [array]: convolved image.
    """
    Gx = np.array([[1, 0], [0, -1]])
    Gy = np.array([[0, 1], [-1, 0]])
    roberts_x = convolve(img, Gx, padding=((0, 1), (0, 1)))
    roberts_y = convolve(img, Gy, padding=((0, 1), (0, 1)))
    img = np.sqrt(roberts_x ** 2 + roberts_y ** 2)
    return img


def gaussian_kernel(n=3, sigma=1.0):
    """Generate a 2d gaussian kernel. Credit: https://stackoverflow.com/questions/29731726/how-to-calculate-a-gaussian-kernel-matrix-efficiently-in-numpy

    Args:
        n (int, optional): size of the kernel. Defaults to 3.
        sigma (float, optional): the standard deviation used to generate the kernel. Defaults to 1.0.

    Returns:
        [array]: gaussian kernel.
    """
    ax = np.linspace(-(n - 1) / 2.0, (n - 1) / 2.0, n)
    gauss = np.exp(-0.5 * np.square(ax) / np.square(sigma))
    kernel = np.outer(gauss, gauss)
    return kernel / np.sum(kernel)


def gaussian(img, sigma=1.0):
    """Perform Gaussian filter on a given grayscale image.

    Args:
        img (array): a grayscale image.
        sigma (float, optional): the standard deviation. Defaults to 1.0.

    Returns:
        [array]: convolved image.
    """
    kernel = gaussian_kernel(sigma=sigma)
    out_img = convolve(img, kernel, padding=1)
    return out_img


def unsharp_mask(img, scaling=0.45):
    """Perform Unsharp Masking filter on a given grayscale image.

    Args:
        img (array): a grayscale image.
        scaling (float, optional): the scaling factor of unsharp masking. Defaults to 0.45.

    Returns:
        [array]: convolved image.
    """
    return img + scaling * (img - gaussian(img))


def transform(point, origin, translate=(0, 0), angle=0):
    """Transform a point to a new coordinate.

    Args:
        point (tuple): the original point. Ex: (4, 5).
        origin (tuple): the origin. Ex: (0,0).
        translate (tuple, optional): the vertical and the horizontal translation. Ex: (-10, 5). Defaults to (0, 0).
        angle (int, optional): the angle of rotation in degree. Defaults to 0.

    Returns:
        [tuple]: the new point.
    """
    # Convert angle from degree to radians
    angle = math.radians(angle)

    # Translate pixels based by translate
    point = tuple(np.add(point, translate))
    origin = tuple(np.add(origin, translate))

    # Rotate pixels based on the given angle
    row0, col0 = origin
    row1, col1 = point
    col2 = math.cos(angle) * (col1 - col0) - math.sin(angle) * (row1 - row0) + col0
    row2 = math.sin(angle) * (col1 - col0) + math.cos(angle) * (row1 - row0) + row0

    return (round(row2), round(col2))
```