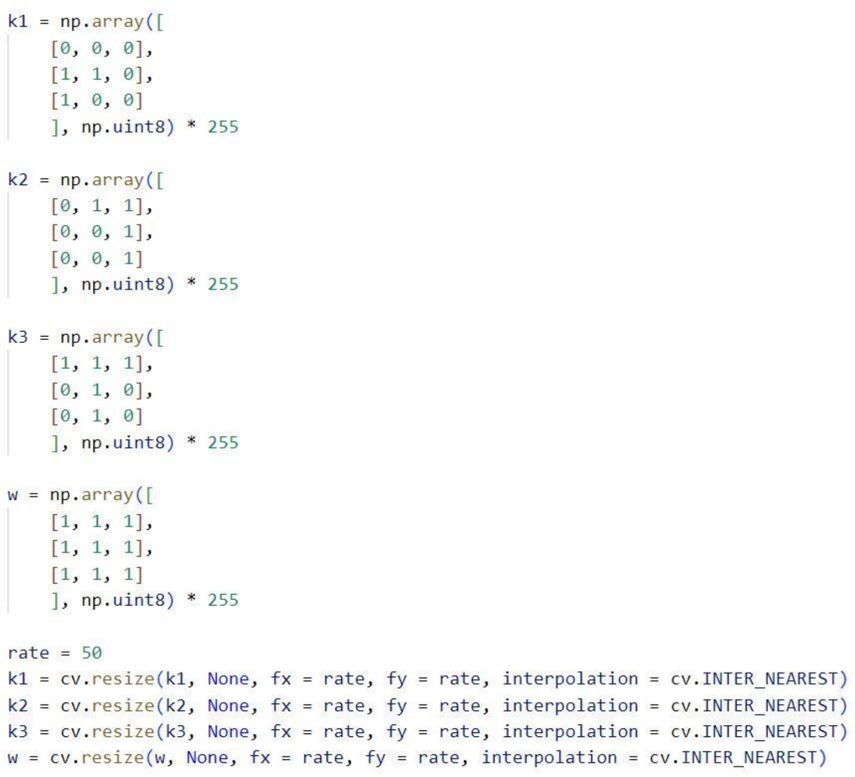
Code For Hit-or-Miss Transform



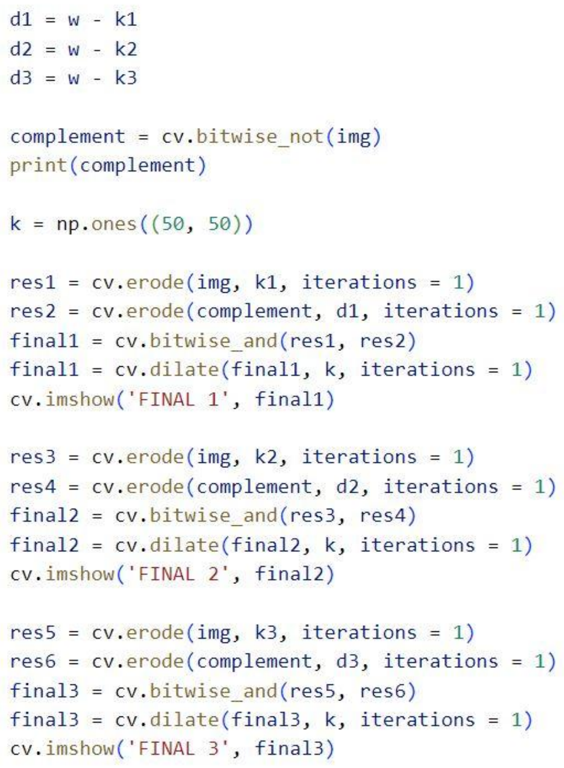
```python
k1 = np.array([
    [0, 0, 0],
    [1, 1, 0],
    [1, 0, 0]
    ], np.uint8) * 255

k2 = np.array([
    [0, 1, 1],
    [0, 0, 1],
    [0, 0, 1]
    ], np.uint8) * 255

k3 = np.array([
    [1, 1, 1],
    [0, 1, 0],
    [0, 1, 0]
    ], np.uint8) * 255

w = np.array([
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1]
    ], np.uint8) * 255

rate = 50
k1 = cv.resize(k1, None, fx = rate, fy = rate, interpolation = cv.INTER_NEAREST)
k2 = cv.resize(k2, None, fx = rate, fy = rate, interpolation = cv.INTER_NEAREST)
k3 = cv.resize(k3, None, fx = rate, fy = rate, interpolation = cv.INTER_NEAREST)
w = cv.resize(w, None, fx = rate, fy = rate, interpolation = cv.INTER_NEAREST)
```



```python
d1 = w - k1
d2 = w - k2
d3 = w - k3

complement = cv.bitwise_not(img)
print(complement)

k = np.ones((50, 50))

res1 = cv.erode(img, k1, iterations = 1)
res2 = cv.erode(complement, d1, iterations = 1)
final1 = cv.bitwise_and(res1, res2)
final1 = cv.dilate(final1, k, iterations = 1)
cv.imshow('FINAL 1', final1)

res3 = cv.erode(img, k2, iterations = 1)
res4 = cv.erode(complement, d2, iterations = 1)
final2 = cv.bitwise_and(res3, res4)
final2 = cv.dilate(final2, k, iterations = 1)
cv.imshow('FINAL 2', final2)

res5 = cv.erode(img, k3, iterations = 1)
res6 = cv.erode(complement, d3, iterations = 1)
final3 = cv.bitwise_and(res5, res6)
final3 = cv.dilate(final3, k, iterations = 1)
cv.imshow('FINAL 3', final3)
```

Output For Hit-or-Miss Transform

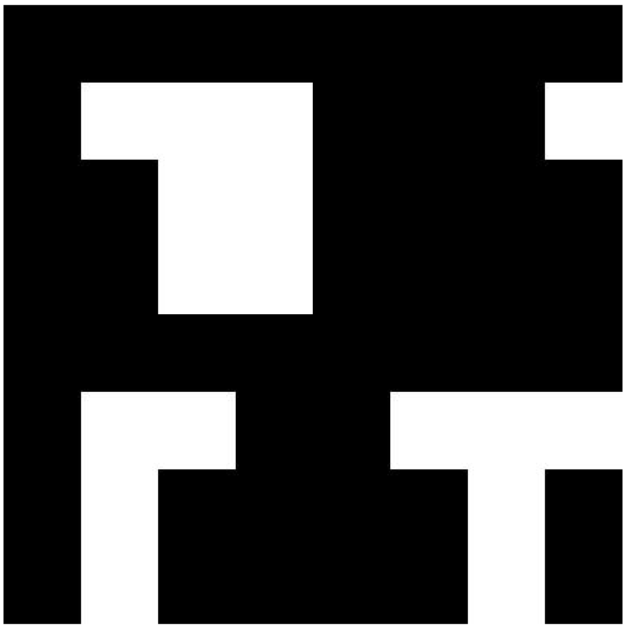


Fig 5.2: Input Image

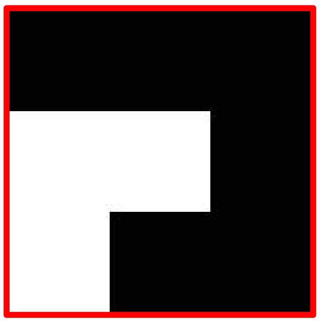


Fig 5.3: Kernel 1

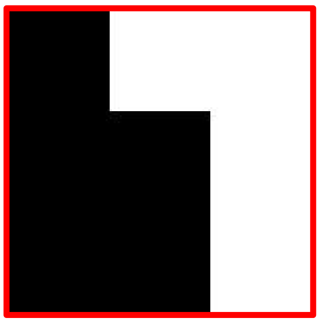


Fig 5.4: Kernel 2

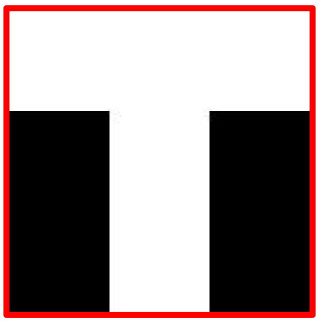


Fig 5.5: Kernel 3

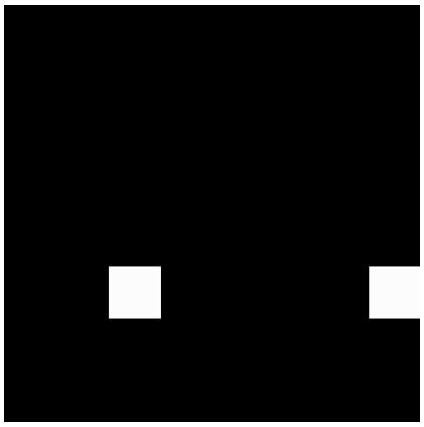


Fig 5.6: Ouput 1

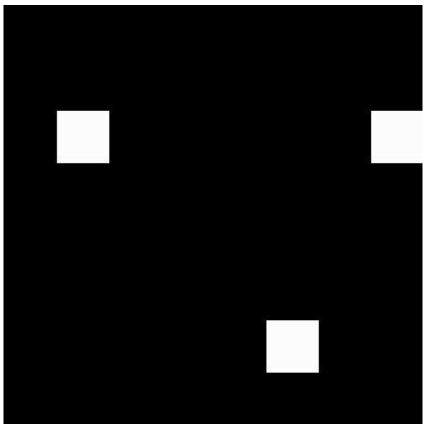


Fig 5.7: Output 2

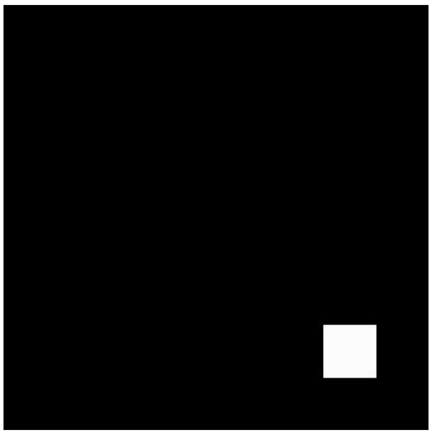
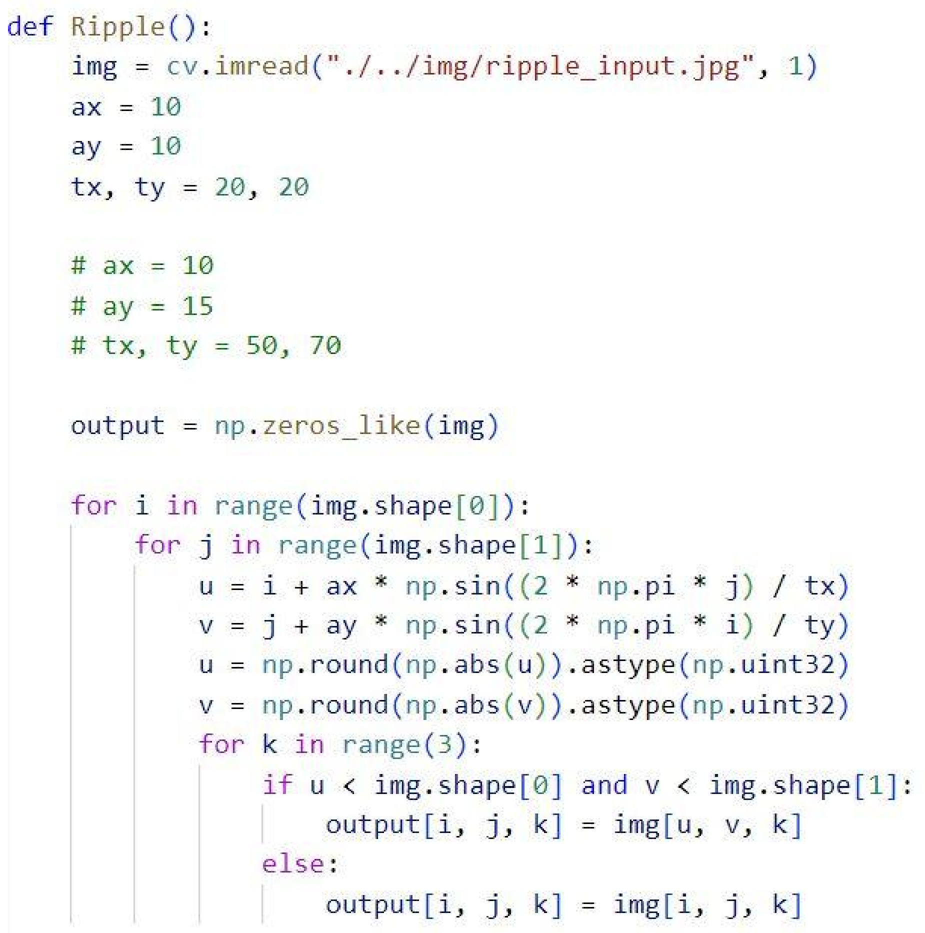


Fig 5.8: Output 3

Code For Ripple and Tapestry Transform

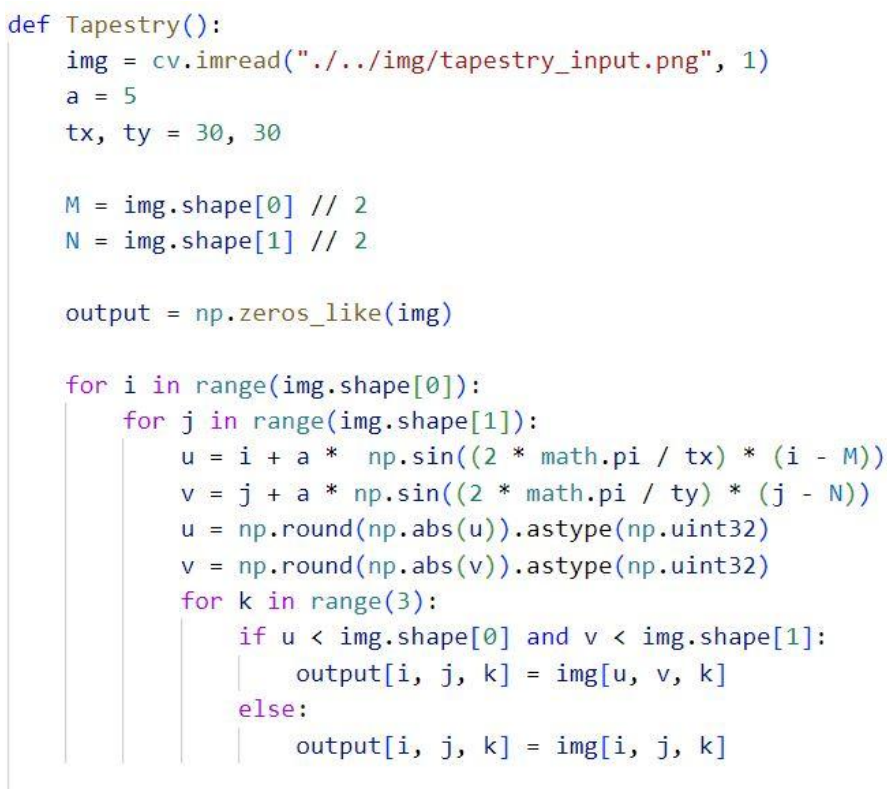


```python
def Ripple():
    img = cv.imread("./../img/ripple_input.jpg", 1)
    ax = 10
    ay = 10
    tx, ty = 20, 20

    # ax = 10
    # ay = 15
    # tx, ty = 50, 70

    output = np.zeros_like(img)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            u = i + ax * np.sin((2 * np.pi * j) / tx)
            v = j + ay * np.sin((2 * np.pi * i) / ty)
            u = np.round(np.abs(u)).astype(np.uint32)
            v = np.round(np.abs(v)).astype(np.uint32)
            for k in range(3):
                if u < img.shape[0] and v < img.shape[1]:
                    output[i, j, k] = img[u, v, k]
                else:
                    output[i, j, k] = img[i, j, k]
```



```python
def Tapestry():
    img = cv.imread("./../img/tapestry_input.png", 1)
    a = 5
    tx, ty = 30, 30

    M = img.shape[0] // 2
    N = img.shape[1] // 2

    output = np.zeros_like(img)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            u = i + a *  np.sin((2 * math.pi / tx) * (i - M))
            v = j + a * np.sin((2 * math.pi / ty) * (j - N))
            u = np.round(np.abs(u)).astype(np.uint32)
            v = np.round(np.abs(v)).astype(np.uint32)
            for k in range(3):
                if u < img.shape[0] and v < img.shape[1]:
                    output[i, j, k] = img[u, v, k]
                else:
                    output[i, j, k] = img[i, j, k]
```

Output For Ripple and Tapestry Transform

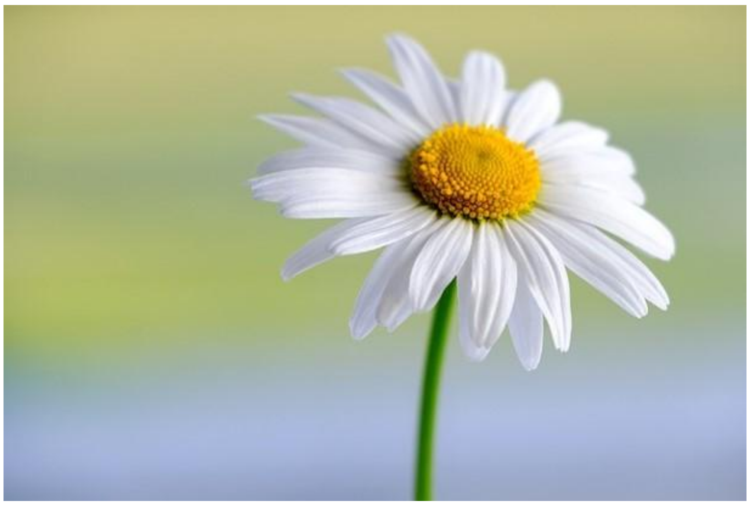


Fig 5.9: Input Image for Ripple Transform

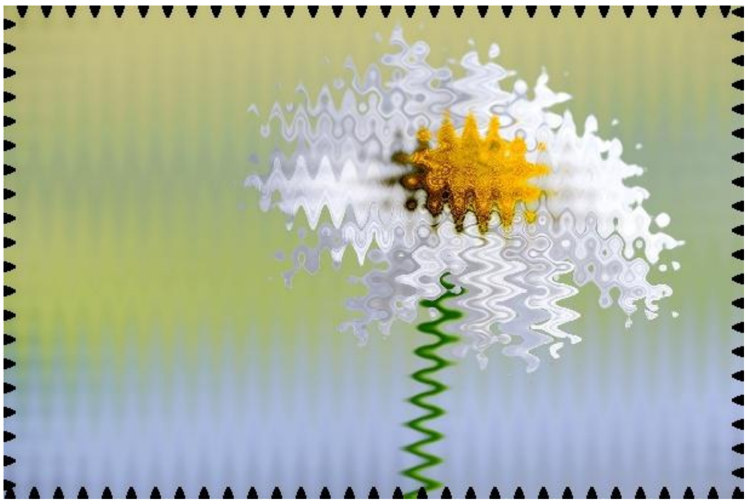


Fig 5.10: Output Image for Ripple Transform

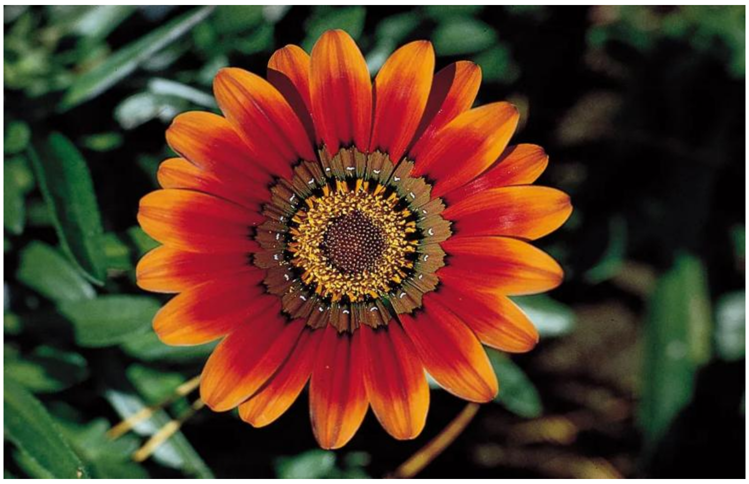


Fig 5.11: Input Image for Tapestry Transform

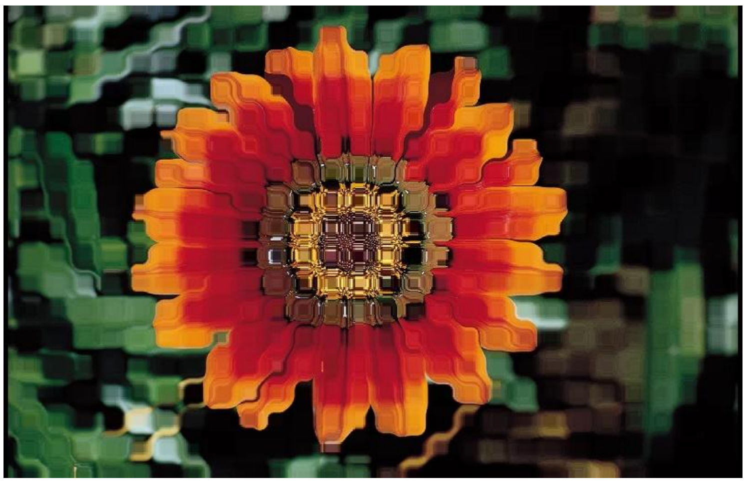


Fig 5.12: Input Image for Tapestry Transform