Computer Vision

NTUT 資訊工程系 111598070 賴俊霖

Assignment 4

Q1. Set initial points

```
# 初始輪廓配置
def initialContour(img,a,b,c,d):
    init_ct = np.zeros(img.shape, dtype=np.int8)
    init_ct[a:b, c:d] = 1
    return init_ct
```

This function take in an image (img), and four integers (a, b, c, and d).

It returns a new array, init_ct, that has the same shape as the input image img, but some parts is initialized to all zeros.

The purpose of this function is to create an initial contour (a boundary or outline) for the input image.

The contour is defined by the region of the image with indices a through b for the rows, and c through d for the columns.

The values of a, b, c, and d specify the range of indices that fall within the contour.

For example, if the input image is a 5x5 array, and a=1, b=3, c=1, and d=3, the resulting init_ct image would be:

Copy code

```
[[0, 0, 0, 0, 0],
  [0, 1, 1, 1, 0],
  [0, 1, 1, 1, 0],
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0]]
```

Note that the contour defined by the indices a through b for the rows and c through d for the columns is represented by the region of 1s in the init_ct image. The rest of the image is filled with 0s.

Q2. Find the contour (3 result images)

```
def activeContour(image, iterations, initLevelSet='circle', smoothing=1, threshold=0.1, balloon=0,
                  iter_callback=lambda x: None):
    initLevelSet = _initLevelSet(initLevelSet, image.shape)

    structure = np.ones((3,) * len(image.shape), dtype=np.int8)
    dimage = np.gradient(image)
    # threshold_mask = image > threshold
    if balloon != 0:
        threshold_mask_balloon = image > threshold / np.abs(balloon)

    u = np.int8(initLevelSet > 0)
    iter_callback(u)

    for _ in range(iterations):
        # print("itr: = " + str(_) + "!")
        # Balloon
        if balloon > 0:
            aux = ndi.binary_dilation(u, structure)
        elif balloon < 0:
            aux = ndi.binary_erosion(u, structure)
        if balloon != 0:
            u[threshold_mask_balloon] = aux[threshold_mask_balloon]

        # Image attachment
        aux = np.zeros_like(image)
        du = np.gradient(u)
        for el1, el2 in zip(dimage, du):
            aux += el1 * el2
        u[aux > 0] = 1
        u[aux < 0] = 0

        # Smoothing
        for _ in range(smoothing):
            u = _curvop(u)
        iter_callback(u)
    return u
```

In this function, the parameter image should be gray-scale image.

The iterations is to iterate the function again and again until the iterations reach our setting, in this function it was set in 400 , cuz every image will converge before reach itr 400.

The smoothing is for _curvop which is below

```python
class _fcycle(object):
    def __init__(self, iterable):
        """Call functions from the iterable each time it is called."""
        self.nextStep = cycle(iterable)

    def __call__(self, *args, **kwargs):
        f = next(self.nextStep)
        return f(*args, **kwargs)
```

The __call__ method is called when the object is "called" as a function. It retrieves the next element from the iterable using the next function and calls it as a function, passing any arguments that were provided when the object was called. This allows you to use the object as if it were a function that returns the next element of the iterable each time it is called.

```python
_curvop = _fcycle([lambda u: erode(dilate(u)),
                   lambda u: dilate(erode(u))])
```

The _curvop variable is a cycle object that alternately applies dilation and erosion to a binary image.

Dilation expands the white regions of a binary image by adding pixels to the boundaries of these regions, while erosion shrinks the white regions by removing pixels from their boundaries.

The cycle object is initialized with a list of two anonymous functions (lambda functions) that apply dilation and erosion to a binary image. When the _curvop object is called with an image as an argument, it will apply dilation to the image, then erosion, then dilation again, and so on. This process of alternately applying dilation and erosion is known as morphological smoothing and is often used to remove noise or small isolated regions from binary images.

```
if balloon != 0:
    threshold_mask_balloon = image > threshold / np.abs(balloon)
```

```
for _ in range(iterations):
    # print("itr: = " + str(_) + "!")
    # Balloon
    if balloon > 0:
        aux = ndi.binary_dilation(u, structure)
    elif balloon < 0:
        aux = ndi.binary_erosion(u, structure)
    if balloon != 0:
        u[threshold_mask_balloon] = aux[threshold_mask_balloon]
```

This line of code creates a binary mask that is used to apply balloon force to the active contour. Balloon force is a term used in active contours to describe a force that expands or shrinks the contour depending on its sign. If balloon is positive, the force expands the contour. If it is negative, the force shrinks the contour. If balloon is zero, no balloon force is applied.

The mask is created by thresholding the image with the value threshold / np.abs(balloon). This means that pixels in the image with intensity greater than this value will be set to 1 in the mask, and all other pixels will be set to 0. The mask is then used to apply balloon force to the contour

For example, if u is the active contour, balloon is 2, structure is a 3x3 square structuring element, and threshold_mask_balloon is a binary mask with 1s at pixels in the image with intensity greater than 0.05 and 0s at all other pixels, the contour will be expanded using the binary_dilation function wherever threshold_mask_balloon is 1.

```
# Image attachment
aux = np.zeros_like(image)
du = np.gradient(u)
for el1, el2 in zip(dimage, du):
    aux += el1 * el2
u[aux > 0] = 1
u[aux < 0] = 0
```

The aux variable is initialized as an array of zeros with the same shape as the input image. The du variable is the gradient of the contour. The zip function is used to iterate over the elements of dimage and du simultaneously. For each element el1 in dimage and el2 in du, the element-wise product el1 * el2 is added to the aux variable.

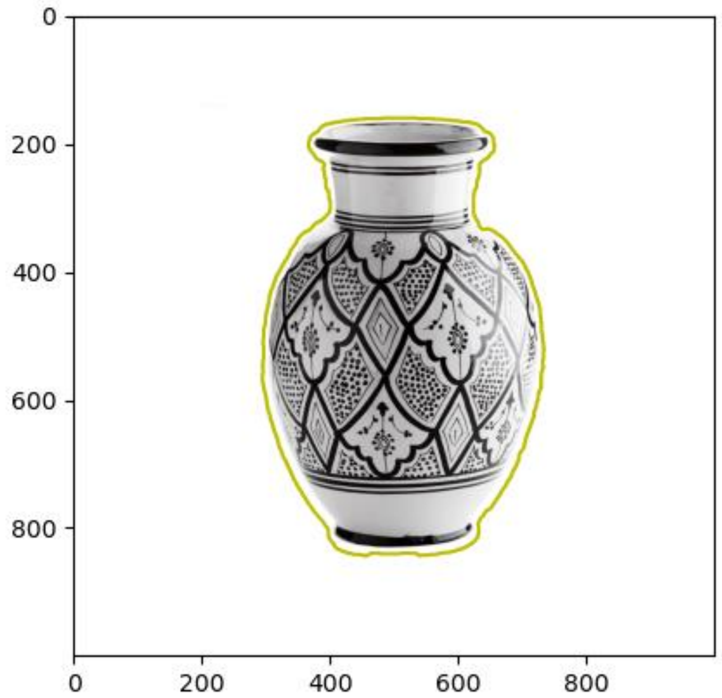
After all the element-wise products have been added, the aux variable is thresholded to create a binary mask. Pixels in the contour with positive values in aux are set to 1, and all other pixels are set to 0. This causes the contour to move towards image features with positive values in aux, which correspond to image features with intensity greater than the threshold.

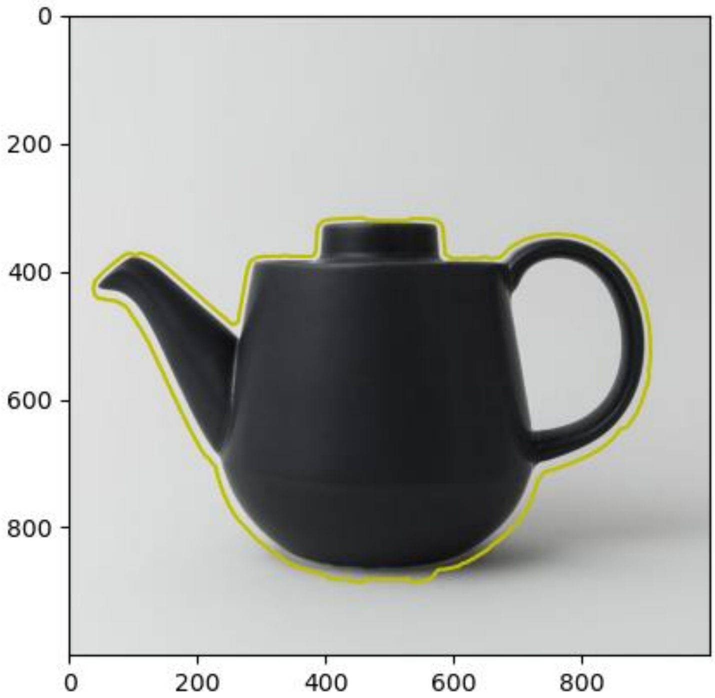
For example, if dimage is the gradient of the image, du is the gradient of the contour, and aux is the sum of the element-wise products of dimage and du, the contour will be attracted towards image features with positive values in aux.

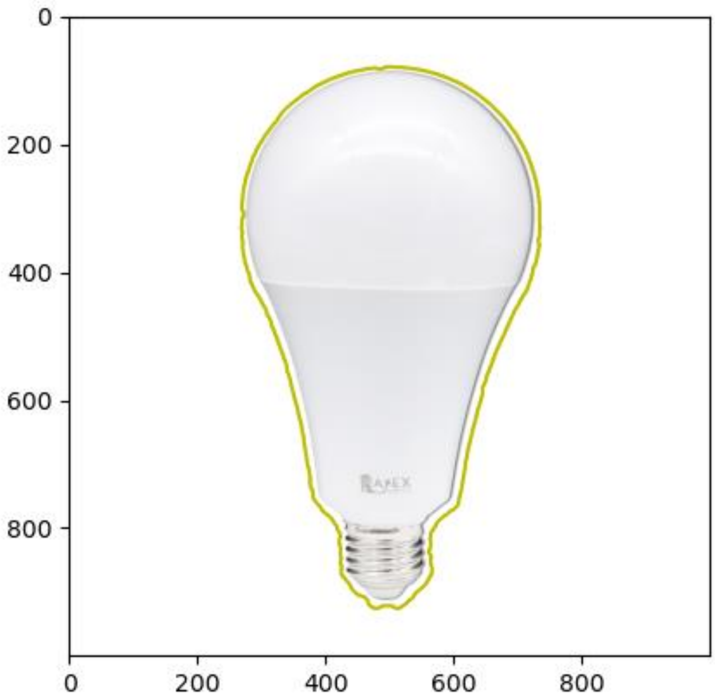
```
for _ in range(smoothing):
    u = _curvop(u)
```

This loop applies morphological smoothing to the active contour. Morphological smoothing is a process of removing noise or small isolated regions from a binary image by alternately applying dilation and erosion.

And there are the output images.




0
200
400
600
800
0
200
400
600
800


0
200
400
600
800
0
200
400
600
800

Q3. Save Convergence video (3 result videos)