COMP2005 Laboratory Sheet 8: Interactive Segmentation

1. Graph Cut

Graph Cut is an interactive segmentation approach which segments out sub-regions of the user's interest. The user can draw curves of different colours to indicate the foregrounds and backgrounds, and based on the given curves, Graph Cut will generate foreground masks. If the user is unsatisfied with the segmented result, the user can improve the results by drawing more curves.

Note that Graph Cut is not available in OpenCV. As a result, an unofficial implementation will be used in this lab session. A Python implementation (*GraphCut.py*) and a test image (*CMS_livingroom.png*) are available on Moodle, or you may also use your own images. To use *GraphCut.py*, follow these steps:

Step 1: Import *GraphCutMaster* from *GraphCut.py* to your current script. (You can skip this step if you are writing your code in *GraphCut.py*).

```
from GraphCut import GraphCutMaster
```

Step 2: Load a test image.

```
image = cv2.imread("CMS_livingroom.png")
```

Step 3: Initialise an instance of *GraphCutMaster*.

```
graph_cut = GraphCutMaster(image)
```

Step 4: Call the segmentation method of *GraphCutMaster* and save the segmented binary mask (0 indicating the background and 1 indicating the foreground) to a variable.

```
mask = graph_cut.segmentation()
```

Step 5: Show the segmented mask on the screen.

```
cv2.imshow("mask", mask * 255)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Run your script or *GraphCut.py* and follow the prompts raised by the program. An example is shown below:

The result gained from the first interaction might not be the best, but you can improve it by further drawing curves.

After pressing the “S” key, the final segmented mask will be shown on the screen.

If you would like to learn more about the unofficial implementation, you can read and try to understand *GraphCut.py* and the original paper of the algorithm, which are available on Moodle. If you like the implementation, don’t forget to give a star to the GitHub repository: https://github.com/YFaris/GraphCut

2. Grab Cut

As an improvement from Graph Cut, Grab Cut requires fewer interactions from the user. Specifically, the algorithm only requires the user to draw a rectangle around the foreground of interest to conduct the segmentation. Like Graph Cut, the user can improve the segmentation result by drawing curves to indicate the foreground and background of the image. Grab Cut can be performed using OpenCV’s built-in function *grabCut*:

mask, bgdmodel, fgdmodel = cv2.grabCut(img, mask, rect, bgdModel, fgdModel, iterCount, mode)

- *img*: the input image
- *mask*: the mask indicating the foreground and background. There are four possible values for the pixels in *mask*:
  - *cv2.GC_BGD* or 0: this pixel is the background.
  - *cv2.GC_FGD* or 1: this pixel is the foreground.
  - *cv2.GC_PR_BGD* or 2: this pixel is probably the background.
  - *cv2.GC_PR_FGD* or 3: this pixel is probably the foreground.
- *rect*: the coordinates of the rectangle drawn by the user. It is a list containing four values:
  - *xmin*: the x coordinate of the top-left corner of the rectangle

  - *ymin*: the y coordinate of the top-left corner of the rectangle
  - *w*: the width of the rectangle
  - *h*: the height of the rectangle

- *bgdModel*: the model of the background colour which is initialised as a zeros array: *np.zeros((1, 65), np.float64)*
- *fgdModel*: the model of the foreground colour which is initialised as a zeros array: *np.zeros((1, 65), np.float64)*
- *iterCount*: the number of iterations the algorithm should run.
- *mode*: the mode of interaction. It has two possible values:
  - *cv2.GC_INIT_WITH_RECT*: the user should draw a rectangle to indicate the foreground.
  - *cv2.GC_INIT_WITH_MASK*: the user should draw curves to indicate the foreground and background (like Graph Cut).

mask returned by *grabCut* is the segmentation result, and just like the *mask* parameter, it also contains four possible values: *cv2.GC_BGD, cv2.GC_FGD, cv2.GC_PR_BGD, cv2.GC_PR_FGD*. As a result, it can be used as the *mask* parameter of *grabCut* when conducting further interactions.

Now let's try this method. Create a Python file called *quickGrabCut.py* and follow the steps below:

Step 1: Import OpenCV and NumPy library.

Step 2: Load a test image.

```
image = cv2.imread("CMS_livingroom.png")
```

Step 3: Allow the user to draw a rectangle to indicate the foreground of interest and save the coordinates of the rectangle drawn by the user.

```
rect = cv2.selectROI('input', img, False)
```

Here, what we did was create a window named 'input' to display the image so that the user can draw a rectangle on it.

Step 4: Initialise the three parameters of *grabCut*: *mask, bgdModel,* and *fgdModel.*

```
# initialise the mask indicating foreground and background with 0
mask = np.zeros(img.shape[:2], np.uint8)
# initialise the foreground and background model
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
```

Step 5: Call the *grabCut* method.

```
mask, bgdModel, fgdModel = cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5,
cv2.GC_INIT_WITH_RECT)
```

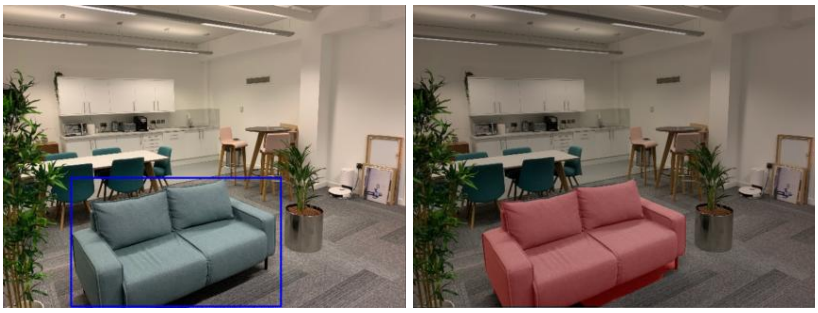
Step 6: Select the foreground and possible foreground as the segmented mask.

```
result = np.where((mask == cv2.GC_FGD) | (mask == cv2.GC_PR_FGD), 1, 0).astype('uint8')
```

Step 7: Display the result on the screen.

```
canvas = np.zeros((img.shape[0], img.shape[1], 3))
canvas = canvas.astype(np.uint8)
canvas[result == 1, 2] = 255
cv2.imshow("visualization", cv2.addWeighted(img, 0.7, canvas, 0.3, 0))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here is the result of this process:



You might have noticed that this example does not allow the user to keep improving the result through further iterations. To solve this, a much more complicated example is given in *GrabCut.py*, which is also available on Moodle.

To run the example in *GrabCut.py*, use the following code:

```
from GrabCut import GrabCutMaster
import cv2
image = cv2.imread("CMS_livingroom.PNG")
grab_cut = GrabCutMaster(image)
mask = grab_cut.segmentation()
```

You can also develop your own solution by following the procedure below:

Step 1: Write a function which receives the user's input whereby the user can draw bounding boxes or red and blue curves. You can find an example in *GrabCut.GrabCutMaster.draw_curve.*

Step 2: Write a function to get the pixels covered by the curves drawn by the user. Note that pixels covered by red curves (foreground) and blue curves (background) should be collected separately. You can refer to *GrabCut.GrabCutMaster.collect_pixels_covered_by_curves.*

Step 3: Consider how to generate a mask based on the pixels collected in Step 2. For example:

```
obj_pixels, bkg_pixels =
self.collect_pixels_covered_by_curves(self.additional_points, self.row, self.col)
if len(obj_pixels) != 0:
    mask[obj_pixels[:, 0], obj_pixels[:, 1]] = cv2.GC_FGD
if len(bkg_pixels) != 0:
    mask[bkg_pixels[:, 0], bkg_pixels[:, 1]] = cv2.GC_BGD
```

Step 4: Write a segmentation method which guides the user to interact with the algorithm. An example is given in *GrabCut.GrabCutMaster.segmentation*. It should meet these requirements:

- In the first interaction, the user can only draw a rectangle on the image.
- The user is encouraged to press Key 1 to start drawing the rectangle.
- The user can start segmentation after drawing rectangles or curves by pressing "Enter".
- After the first interaction, the user can choose to draw red curves indicating the foreground by pressing Key 2.
- After the first interaction, the user can choose to draw blue curves indicating the background by pressing Key 3.
- The user can press Key 4 to finish the segmentation and close all windows.

Run your script or *GrabCut.py* and follow the prompts raised by the program. An example is shown below.

Further interactions by drawing curves: