Image processing course – homework #2

imageprocessinghaifau@gmail.com

In this homework you will be writing a morphing sequence using transformations and image mappings.

Hw2_functions.py:

1. getImagePts(im1, im2, varName1, varName2, nPoints)

Input: im1, im2 - **grayscale** images in the range [0..255] . Not necessarily same size.
varName1, varName2 – strings that represent the names of variables to be saved as (for example if I want to save the pointset as imagePts1.npy and imagePts2.npy, I will pass the names varName1 = "imagePts1" and varName2 = "imagePts2".
nPoints – number of points the user chooses.

Output: None.

Method: Allows user to select corresponding pairs of points, one set in im1 and one in im2 (in same order). Function opens a figure of im1 and let's the user select nPoints on the image. Then does the same for im2.
Coordinates (x,y) of the points in each image are collected in imagePts1, imagePts2 – np arrays Nx2. After collecting all coordinates, round them using np.round and add a third dimension of ones which will make them Nx3 arrays, then save the array using:

```
np.save("imagePts1.npy", imagePts1)
np.save("imagePts2.npy", imagePts2)
```

This will save the arrays as files in the project folder.
later, you can load them instead of choosing points again, by using:

```
imagePts1 = np.load("imagePts1.npy")
imagePts2 = np.load("imagePts2.npy")
```

- Use plt.ginput to interactively choose points
  Documentation:
  https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.ginput.html

- Pay very close attention to the x,y order. In Image assignment image[x,y] – x is the rows, but in x,y = ginput – x is the horizontal axis.

2. findAffineTransform (pointsSet1,pointsSet2)

Input: pointsSet1, pointsSet2 - arrays Nx3 of coordinates representing corresponding points between the 2 sets.

Output: T – a 3x3 np matrix representing an affine transformation.

Method: Calculates the parameters of the affine transform that best maps points in pointsSet1 to corresponding points in pointsSet2 in the least mean square sense.

$$h_{affine} = \begin{pmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{pmatrix}$$

Note: use functions np.matmul, np.linalg.pinv, np.reshape.
(see slides 43-49 in lecture **Lesson 5: Geometric Operations**)

Note: you can loop over the N points.

3. findProjectiveTransform (pointsSet1,pointsSet2)

Input: pointsSet1, pointsSet2 - arrays Nx3 of coordinates.
representing corresponding points between the 2 sets.

Output: T – a 3x3 np matrix representing a projective transformation.

Method: Calculates the parameters of the projective transform that best maps points in pointsSet1 to corresponding points in pointsSet2 in the least mean square sense.

$$h_{projective} = \begin{pmatrix} a & b & e \\ c & d & f \\ g & h & 1 \end{pmatrix}$$

(see slides 50-56 in lecture **Lesson 5: Geometric Operations**)

Note: you can loop over the N points.

4. mapImage (im,T,sizeOutIm)

Input: im - a grayscale image array in the range [0..255].
T - a 3x3 matrix representing a transformation.
sizeOutIm – a tuple (numRows, numCols) representing the size of the output image.

Output: newIm - a grayscale image in the range [0..255] of size sizeOutIm containing the transformed image.

Method: Create newIm as an empty array of size sizeOutIm. For every coordinate of newIm, apply inverse mapping to determine source coordinates. Map value of image im at the source coordinates to the new image.
Use bilinear Interpolation (tip: implement nearest neighbor interpolation first as it is easier to code, once everything is running – replace it with bilinear interpolation).
Ensure that source coordinates do not fall outside im range. (points whose source are outside source image are assigned gray value zero).

Note: use functions np.linalg.inv , np.meshgrid, np.vstack, np.matmul, np.round, np.any, np.delete.

Note: Do not iterate over pixels of im. You don't need a for loop to implement this function.

5. createMorphSequence (im1,imagePts1, im2,imagePts2, t_vec, transformType)

Input: im1, im2 - grayscale image arrays in the range [0..255].

imagePts1, imagePts2 - np arrays Nx3 of chosen coordinates of im1, im2.
t_vec – a vector with t values where t is in [0..1]. You can use np.linspace(0,1,M) (outside the function) to create a vector with M equal steps between 0 and 1.
transformType – a scalar. 0 = affine, 1 = projective.

Output: images – a list of images. The same size as t_vec.

Method: Input images must be of same size.
Calculate the transforms T12 and T21 that map im1 to im2 and im2 to im1 respectively. For every t do the following:
Calculate T12_t, the transformation from im1 to im2, t parts of the way.
As we did in the tutorial:

$$T_{12\_t} = (1-t)\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} + t\,T_{12}$$

Calculate T21_1_t, the transformation from im2 to im1, (1-t) parts of the way:

$$T_{21\_t} = (1-t)t\,T_{21} + t\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$$

Map im1 using T12_t, producing newIm1
Map im2 using T21_1_t, producing newIm2
Crossdisolve newIm1 and newIm2 with weights associated with t.

Note: use function np.eye(3) to create I matrix of size 3x3.

use functions inside createMorphSequence:

- findProjectiveTransform/findAffineTransform to acquire transform.
- mapImage to map to newIm1, newIm2.

Hw2_script.py:

a. Create a sequence of images morphing from one face image to another. Choose and save 12 points based on the Locations.jpeg image provided (you can choose more). Then morph them using the functions you created. Use numFrames large enough to create a smooth transition. Display the created morph sequence.
   You are provided with a function writeMorphingVideo(image_list, name_video) which creates an mp4 video with a given name out of the images list returned from createMorphSequence (the example video provided was done with numFrames=100 and 12 points).
b. Show an example where the projective transform works better than the affine transform. (choose the images yourself, it can be anything other than faces). You can save location points beforehand with the images you chose.
c. Show that the points chosen affect the transform calculation:
   I. Show that the number of points chosen affect the morph result.
      Display the image at t=0.5 for both: with small number of points, and with large number of points (display side by side using subplot).
   II. Show that the location of points chosen affect the morph result.
      Display the image at t=0.5 for both: with points distributed well in the image, and with points focused in a small area (display side by side using subplot).
d. Enter the class CONTEST!!
   Create a morph sequence of your choice. Be CREATIVE!!
   the best morph sequence will get 10 bonus points on this exercise!!
   Winner will be chosen based on creativity and execution.
   Name your video sequence “forContest<IDstudent1>_<IDstudent2>.mp4”

Good luck!