Nijo Jacob
CS-583-001
4/27/2021
Dr Lou Kratz

Assignment 2: Homographies

Skeleton Code:

def build_A(pts1, pts2):

where create a computation of an homograph mapping from the source points to the destination points, first create A which is 2N by 9 intermediate matrix. Then iterate over the points and populate the rows of A.

def compute_H(pts1, pts2):

We take the parameter points of source and destination points and build intermediate matrix by calling function build A. Then get the transpose of A and compute the symmetric matrix Ata using the numpy dot product of both A and transpose of A. (ATA). Find the eigenvalues and eigenvector of ATA using the numpy linalg.eig which compute the eigenvalues and right eigenvectors of a square array (ATA). Then we use the numpy argsort which sort the eigenvalues array by indices and return the smallest at the beginning of the array [0].

def bilinear_interp(image, point):

This function contains takes the image and the tuple of floating point x, y values. We get x, y from points and covert to int and make it I and j. We check to see if it is within the range of image by try and expect check and implement the bilinear interpolation.

def apply_homography(H, points):

Apply homography matrix H to the provided cartesian points and returns the results as cartesian coordinates. First creating a array of ones and combine with the points. Then apply the homography for each x,y,z using dot product and calculate the cartesian values, append the values to array and return it.

def warp_homography(source, target_shape, Hinv):

First taking the source(3-channel image numpy array) and allocating numpy array of zeros that is size of target_shape and dtype of source. Then iterate all pixel in target image, create a numpy array of [x,y] reshape x,y to 1x2 matrix and apply_homograph with Hinv and points(x,y). Apply numpy sequeeze to remove axes of length one from homoresult. Check if homograph is outside the source image with if statement, if true continue else then perform bilinear interpolation.

def rectify_image(image, source_points, target_points, crop):

The rectify function wraps the input image source points to the plane defined by target points. First compute the rectifying homograph and apply to a rectange of the bounding box. Apply Homography to bounding bounding box and rectifying homograph. Sort the array with numpy of both x and y values and find the min values if statement utilize which points the first smallest else the second smallest. Then we compute the rectified bounding box by applying the translation matrix (applr_homography). Compute

the inverse homograph by compute_H, once again sort the computed H and crop the max values of the x and y of the first and second values and finally warp the homograph.

def blend_with_mask(source, target, mask):

Blends the source image with the target image according to the mask. Normalize the mask values by divide by 255 giving 0 and 1 values and blend the mask with the linear combination of the mask with target and source. Finally convert the result as be the same type as the source.

def composite_image(source, target, source_pts, target_pts, mask):

Finally we composites source image masked planar region on to a planar region of the target image. From compute the homograph using the target and source points to wrap homograph with homograph shape of target and source and blend the mask with wrapped image from wrap function, target image and the mask image.

Results:

Test Case results

```
nj398@tux4:~$ python3 hw2_test.py -v
test_apply_homography (__main__.Homework2Test) ... ok
test_bilinear_interp (__main__.Homework2Test) ... ok
test_build_A (__main__.Homework2Test) ... ok
test_compute_H (__main__.Homework2Test) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.002s

OK
```

Rectify => Original Example:

```
python3 hw2.py rectify example_rectify_input.jpg 3 481 80 0 602 215 637 475
example_rectify_output.jpg
```

Input

Output

```
python3 hw2.py rectify example_rectify_input.jpg 3 481 80 0 602 215 637 475 --crop
example_rectify_output_crop.jpg
```

Input

Output

Rectify => My Own Example:

```
python3 hw2.py rectify own_rectify_input.jpg 3 481 80 0 602 215 637 475
own_rectify_output.jpg
```

Input

Output

Though the crop does not work on this image, I test it out to see what it would look like

```
python3 hw2.py rectify own_rectify_input.jpg 3 481 80 0 602 215 637 475 --crop
own_rectify_result_crop.jpp
```

Input

Output

Composition => original Example Output:

```
python3 hw2.py composite example_panda.png example_laptop.png 100 520 50 300 280 259
325 460 example_mask.jpg example_composite_output.jpg
```

Composition => my own Example Input and Output:

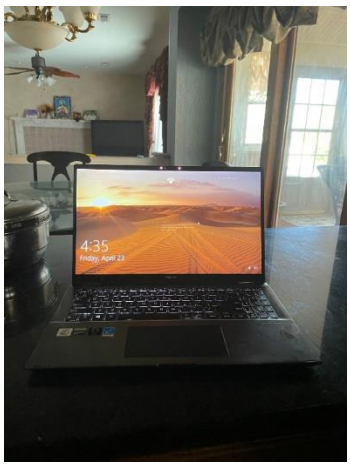
The points on the image were modified to incorporate images on the screen of the laptop as seen below. The pixel points on the image are modified multiple times to fixate the image on screen and then utilize the X,Y points image website: https://www.mobilefish.com/services/record_mouse_coordinates/record_mouse_coordinates.php

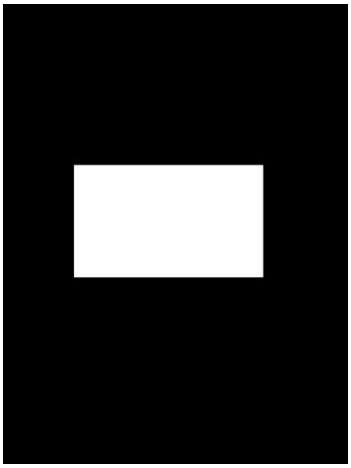
To get actually points after multiple tries. I created the mask image using the GIMP software and GIMP software to remove 4 channel to 3 channels (rbga to rbg) of the input images.

The images types are exactly same as example from professor using png, unlike it was as mention in the instruction the input image to be in jpg.

```
python3 hw2.py composite own_composite_input_2.png own_composite_input_1.png 108 411
109 243 392 245 394 414 example_own.jpg own_composite_output.jpg
```

Input:



Output: