R Noah Padilla

CS 4363

2/24/2021

Lab 3 Report

Problem description

For this lab I was to implement manually guided Homography construction for replacing image regions. This was a very interesting topic to cover since the process underlying this is very important for image editing: photoshop, panoramas, etc.. There were 3 parts to this assignment however I only did 2 (part 1 and part 3). The 3 problems are shown below from the instructions.

1 Region Replacement

Write a program to replace a region in an image with the contents of another image. Your program should generate results similar to those in the images below.

Original image

Image to insert

Generated image

2 Image Stitching

Write a program to stitch a set of partially overlapping images into a single image. Your program should generate results similar to those in the images below.

Images to stitch

Generated image

3 Region Replacement and Greenscreening

Write a program to implement *virtual ads*, as commonly done in sports events. Your program should combine image warping and green screening to produce results similar to those in the images below.

Original image with green background area

Generated image

Ad to be inserted

That is –

1. **Region Replacement** - replace a region in an image with the contents of another image
2. **Image Stitching** - stitch a set of partially overlapping images into a single image
3. **Region Replacement and Green screening** - implement virtual ads, as commonly done in sports events

Algorithms Implemented

Region Replacement – multi_warp()

For this problem, it was very simple and I did not have to implement much since starter code was given however this process in its most basic terms is, we have two matrices – the image 1 (original image) and image 2 (the image we are projecting the original on). To project one image onto the other, the steps are –

1. Select 4 points on image 1 where we want to project image 2

2. Compute a homography, H, that multiplies image 2 so that it generates a new image 2, warped, to fit in those 4 points
3. Create a mask so that image 2 can overlay on it
4. Generate a final image by combining the warped image from step 2, the mask from step 3, and the original image
5. Repeat steps 1-4 *n* times if you are projecting *n* images

Region Replacement and Green Screening – green_screen()

Lastly, this problem was pretty much the same as the first since the goal was still to overlay an image but different in one aspect – only overlay image 2 in the given region on the green pixels only. What I did was –

1. Select 4 points on image 1 where we want to project image 2
2. Compute a homography, H, that multiplies image 2 so that it generates a new image 2, warped, to fit in those 4 points
3. Create a mask so that image 2 can overlay on it
4. Find all values in the image in that are considered "green enough" or above a threshold for the overlay to be placed on using the color_index function from lab 1. The color index function gets an image and returns the same image but its values are modified so that a certain channels values are more exaggerated.
5. Remove the mask and warped image values that were not considered green enough to be overlayed on the green screen using step 4 within the 4 points
6. Generate a final image by combining the warped image from step 2, the mask from step 3, and the original image

Results / Results

Region Replacement – multi_warp()

The modifications made from the original code were minor since steps 1-4 were already done, all I had to do was stick the given code into a for loop to make it work for more than one set of four points (step 5). The results are shown below after implementing the algorithm above-

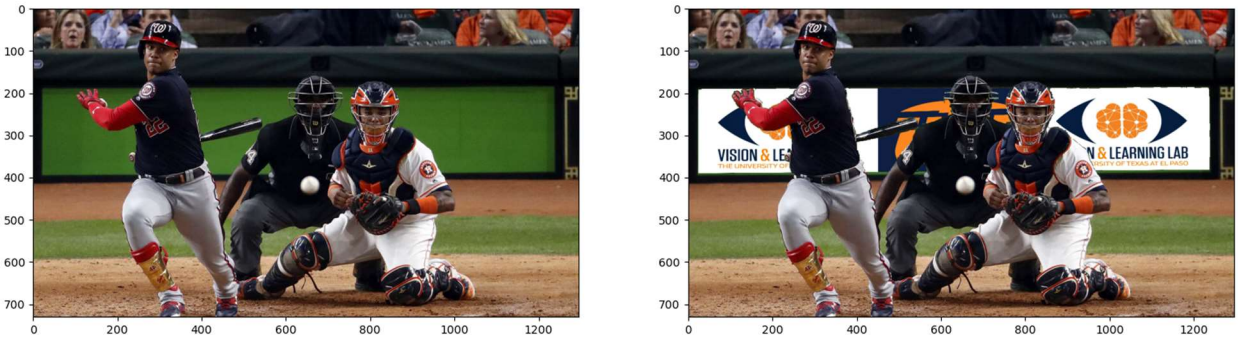
Below, to the left we see image 1 and to the right is the projection of image 2 on image 1 where 8 points were selected. The back of the mobile billboard points were selected in clockwise fashion from the top left whereas the side of the mobile billboard points were selected in a counter clockwise fashion starting from the top right to give a mirroring effect

Below, to the left we have image 1 and to the right we have image 2 with Bernie on the middle and right screens. Both points were chosen in a clockwise fashion starting from the top right.

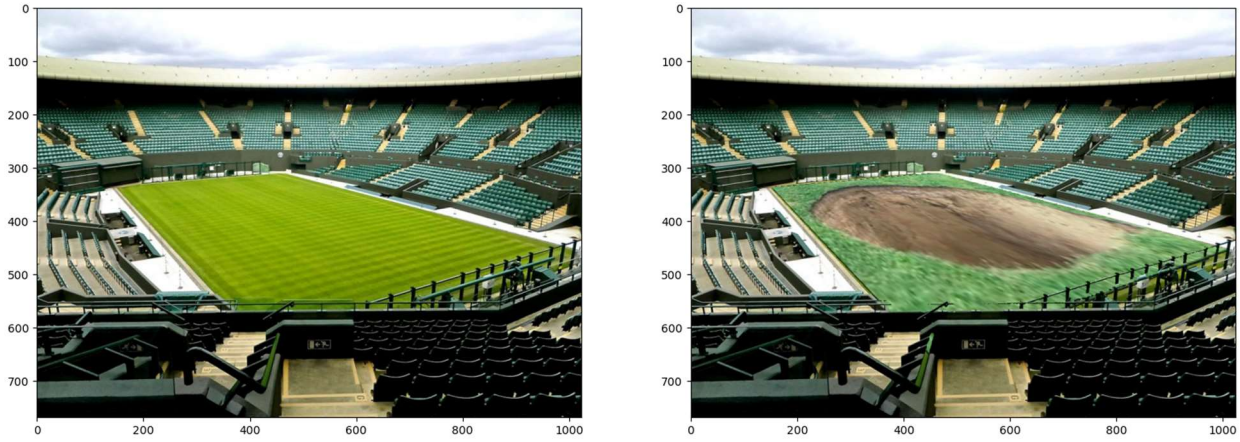
Region Replacement and Green Screening – green_screen()

The modifications made from the original code were minor since steps 1-4 were already done as stated previously, however all I had to do was find where the green screen was so that overlaying image 2 would only overlay on valid green pixels. The results are shown below after implementing the algorithm above-

Below, to the left we have image 1 and to the right we have image 2 projected on image 1. As we can see around the baseball hitters hands it doesn't seem as good as it can be, but further modification to the threshold when considering the green screen would cause the ad to overlay on the baseball player. Nonetheless, the results were sufficient to project image 2 on the green screen only and the errors were negligible.



Below, to the left we have image 1 (Wimbledon tennis court) and to the right we have image 2 (a giant hole) projected on image 1. To the bottom right of the generated image, we can see that image 2 does in fact obey the threshold and demonstrates a sufficient threshold for green screening to occur.



Conclusions

The algorithms that were proposed and demonstrated for problems 1 and 3 were shown to be sufficient in generating the target image respective of their tasks. What I learned from this project was how images can be used to be projected in a fitted region using a projective homography as well modifying the overlay of a projection to only display in a green region (green screening).

Appendix

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from skimage import transform as tf
from utils import *

def multi_warp(img,img2):

    fig, ax = plt.subplots(figsize=(12,10))
    ax.imshow(img)
    fig.suptitle('Original image - Select 8 points', fontsize=14)

    #Input the source points
    print("Click 8 source points")
    srcs=[]
    srcs.append(np.asarray(plt.ginput(n=4)))
    srcs.append(np.asarray(plt.ginput(n=4)))
```

```
    # The destination are the four corners of the image
    dest_rows,dest_cols = 300,200
    dest = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[dest_cols,dest_rows]])

    for src in srcs:

        '''
        # Display point correspondences -points
        fig, ax = plt.subplots(ncols=2, figsize=(8, 4))
        ax[0].imshow(img)
        ax[1].imshow(np.zeros((dest_rows,dest_cols)),cmap=plt.cm.gray)
        ax[0].set_title('Source points')
        ax[1].set_title('Destination points')
        color='rgby'
        for i in range(src.shape[0]):
            ax[0].plot(src[i, 0], src[i, 1], '*',color=color[i])
            ax[1].plot(dest[i, 0], dest[i, 1], '*',color=color[i])
        ax[1].set_xlim([-5, dest_cols+5])
        ax[1].set_ylim([dest_rows+5,-5])
        fig.suptitle('Point correspondences', fontsize=14)
        '''

        # Compute homography from points
        H0 = tf.ProjectiveTransform()
        H0.estimate(src, dest)

        # Find destination image
        warped = tf.warp(img, H0.inverse, output_shape=(np.amax(dest[:,1])+1,
np.amax(dest[:,0])+1))
```

```
'''
# Display source and destination images
fig, ax = plt.subplots(ncols=2, figsize=(8, 4))
ax[0].imshow(img, cmap=plt.cm.gray)
ax[1].imshow(warped, cmap=plt.cm.gray)
ax[0].set_title('Source image')
ax[1].set_title('Destination image')
fig.suptitle('Image transformation', fontsize=14)
'''

src2 = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[img2.shape[1],img2.shape[0]]])
dest2  = src   # Destination points are the source points in the first image

'''
# Display point correspondences
fig, ax = plt.subplots(ncols=2, figsize=(8, 4))
ax[0].imshow(img2)
ax[1].imshow(img)
color='rgby'
for i in range(src.shape[0]):
    ax[0].plot(src2[i, 0], src2[i, 1], '*',color=color[i])
    ax[1].plot(dest2[i, 0], dest2[i, 1], '*',color=color[i])
ax[0].set_xlim([-10, img2.shape[1]+10])
ax[0].set_ylim([img2.shape[0]+10,-10])
fig.suptitle('Point correspondences', fontsize=14)
'''
```

```
H1 = tf.ProjectiveTransform()
H1.estimate(src2, dest2)
warped = tf.warp(img2, H1.inverse, output_shape=(img.shape[0],img.shape[1]))

'''
fig, ax = plt.subplots(ncols=2, figsize=(8, 4))
ax[0].imshow(img2)
ax[1].imshow(warped)
ax[0].set_title('Source image')
ax[1].set_title('Destination image')
fig.suptitle('Transformed source image', fontsize=14)
'''

# Finding mask
# mask[r,c] = 0 if warped[r,c] = [0,0,0], otherwise mask[r,c] = 1
mask = np.expand_dims(np.sum(warped,axis=2)==0,axis=2).astype(np.int)
# Combining masked source and warped image
combined = warped + img*mask

#transform then transform again with the old image
img = combined

'''
fig, ax = plt.subplots(ncols=3, figsize=(12, 4))
ax[0].imshow(img*mask)
ax[1].imshow(warped)
ax[2].imshow(combined)
ax[0].set_title('Original * mask')
```

```
        ax[1].set_title('Destination')
        ax[2].set_title('Original * mask + destination')
        fig.suptitle('Final image creation', fontsize=14)
        '''

    return img

#img is image with green screen | img2 is image that we are inserting
def green_screen(img,img2):

    fig, ax = plt.subplots(figsize=(12,10))
    ax.imshow(img)
    fig.suptitle('Original image - Select 4 points', fontsize=14)

    src = np.asarray(plt.ginput(n=4))
    #src = np.asarray([[19,191], [1230,184] , [1234,372], [27,387] ])

    # The destination are the four corners of the image
    dest_rows,dest_cols = 300,200
    dest = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[dest_cols,dest_rows]])

    # Display point correspondences -points
    fig, ax = plt.subplots(ncols=2, figsize=(8, 4))
    ax[0].imshow(img)
    ax[1].imshow(np.zeros((dest_rows,dest_cols)),cmap=plt.cm.gray)
    ax[0].set_title('Source points')
    ax[1].set_title('Destination points')
    color='rgby'
```

```
    for i in range(src.shape[0]):
        ax[0].plot(src[i, 0], src[i, 1], '*',color=color[i])
        ax[1].plot(dest[i, 0], dest[i, 1], '*',color=color[i])
    ax[1].set_xlim([-5, dest_cols+5])
    ax[1].set_ylim([dest_rows+5,-5])
    fig.suptitle('Point correspondences', fontsize=14)

    # Compute homography from points
    H0 = tf.ProjectiveTransform()
    H0.estimate(src, dest)

    # Find destination image
    warped = tf.warp(img, H0.inverse, output_shape=(np.amax(dest[:,1])+1,
np.amax(dest[:,0])+1))

    '''
    # Display source and destination images
    img2 = mpimg.imread('vll_utep_720.jpg')
    img2 = img2/np.amax(img2)
    '''

    src2 = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[img2.shape[1],img2.shape[0]]])
    dest2  = src   # Destination points are the source points in the first image

    H1 = tf.ProjectiveTransform()
    H1.estimate(src2, dest2)
    warped = tf.warp(img2, H1.inverse, output_shape=(img.shape[0],img.shape[1]))

    '''
```

```
    Goal: Only overlay on greenscreen and not everything else
    '''


    # Finding mask
    # mask[r,c] = 0 if warped[r,c] = [0,0,0], otherwise mask[r,c] = 1 // 1 is background of mask and 0 is the overlayer where image will go
    mask = np.expand_dims(np.sum(warped,axis=2)==0,axis=2).astype(np.int) #mask shape (730,1296,1)


    #Calculate the green screen values | a 2D array booleans at each index in the image that are valid green screen vals
    gs  = np.asarray(color_index(img,1) > .200)#gs shape = (730, 1296)


    for i in range(mask.shape[0]):
        for j in range(mask.shape[1]):


            #Setting the mask and the ad to only go on the green screen
            if gs[i,j] == False:
                mask[i,j] = 1
                warped[i,j] = 0


    # Combining masked source and warped image
    #make sure that we overlay our add on the green screen only


    #(730,1296,3)
    combined = warped + img * mask


    #transform then transform again with the old image
    img = combined
```

```
    return img


if __name__ == "__main__":

    #------------------- USING REQUIRED IMAGES

    #1 projective warping
    img = mpimg.imread('mobile_billboard.jpg')
    img = img/np.amax(img)
    # Display source and destination images
    img2 = mpimg.imread('utep720.jpg')
    img2 = img2/np.amax(img2)
    warped_img = multi_warp(img,img2)
    fig, ax = plt.subplots(ncols=2, figsize=(12, 4))
    ax[0].imshow(img)
    ax[1].imshow(warped_img)

    #3 Green screening
    img = mpimg.imread('soto.jpg')
    img = img/np.amax(img)
    # Display source and destination images
    img2 = mpimg.imread('vll_utep_720.jpg')
    img2 = img2/np.amax(img2)
    warped_img = green_screen(img,img2)
    fig, bx = plt.subplots(ncols=2, figsize=(12, 4))
    bx[0].imshow(img)
```

```
bx[1].imshow(warped_img)

#-------------------- USING MY OWN IMAGES

#1 projective warping
img = mpimg.imread('timesquare.jpg')
img = img/np.amax(img)
# Display source and destination images
img2 = mpimg.imread('bernie.jpg')
img2 = img2/np.amax(img2)
#1 projective warping
warped_img = multi_warp(img,img2)
fig, ax = plt.subplots(ncols=2, figsize=(12, 4))
ax[0].imshow(img)
ax[1].imshow(warped_img)

#3 Green screening
img = mpimg.imread('wimbeldon.jpg')
img = img/np.amax(img)
# Display source and destination images
img2 = mpimg.imread('hole.jpg')
img2 = img2/np.amax(img2)
warped_img = green_screen(img,img2)
fig, bx = plt.subplots(ncols=2, figsize=(12, 4))
bx[0].imshow(img)
bx[1].imshow(warped_img)
```