ID:80585887
CS 5363
Dr. Olac Fuentes

I. Problem Description

For Lab 3 our task is to perform three different manipulation of images. First, we are asked to implement the warping of an image to “land” or replace some area of a target image. Secondly, we are asked to implement through warping the stitching of a set of three images that basically are sections of the same area. Lastly, we had to implement an algorithm that basically used a green background to indicate the area that should be filled with pixels of a source image.

II. Algorithms Implemented

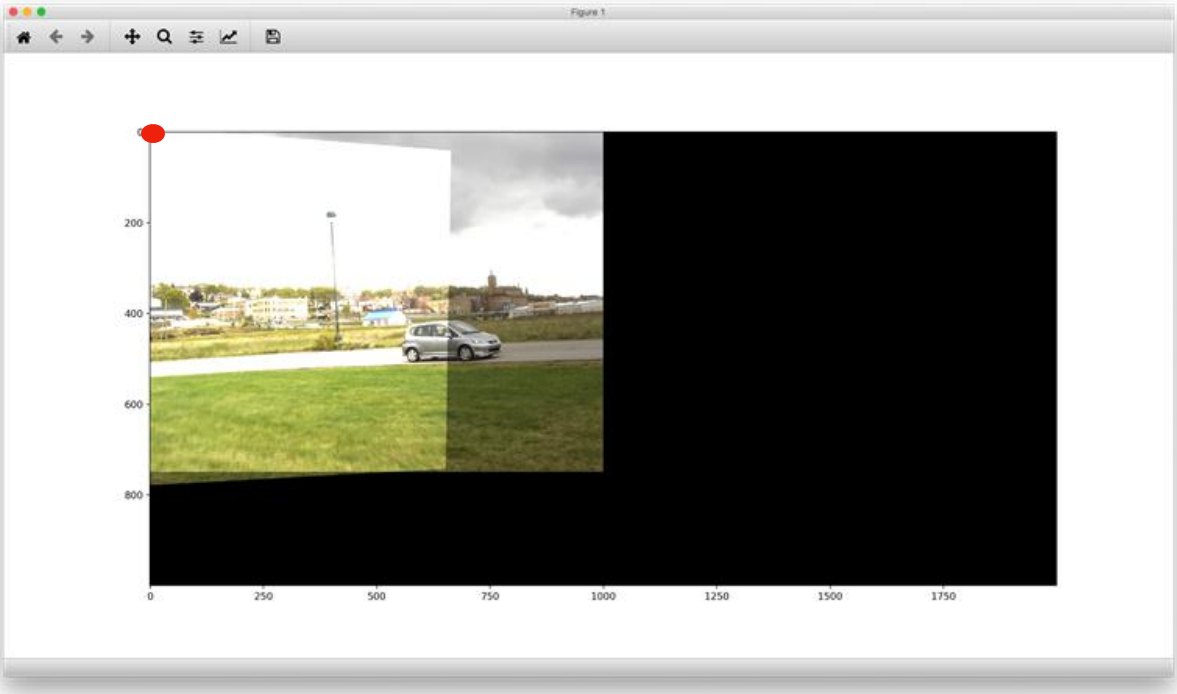
Problem 1

For this problem I basically used the code presented by Dr. Fuentes, this code takes the points from the image (i.e. 2 sets of 4 points) and we assume this points are the destination and the src are the points of the corners of UTEP's logo. So, to actually place the logo in its appropriate place we apply the inverse of the computed Homography, that way we make the points of UTEP's logo to map to those we wish to fill. In addition, we force the resulting image after the homography is applied to be of the size of the target images (i.e. the casino image). Then, we apply a mask of zeros to the area where we wish to map UTEP's logo through a mask that contains 1's in the coordinates that are not in the target area and 0's where the target area is located which essentially makes the area to be blacked out. Finally, we simply add the resulting, warped UTEP logo to the masked target image and that way the images blend nicely.

```
src2 = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[img2.shape[1],i
warped,H1 = warp(img2,src2,dest2,shape=(img.shape[0],img.shape[1]))
dest = src_2
warped2,H2 = warp(img2,src2,dest,shape=(img.shape[0],img.shape[1]))
warped += warped2
mask = np.expand_dims(np.sum(warped,axis=2)==0,axis=2).astype(np.int)
```

Problem 2

For this problem I first experimented with the warping of the left hand image and the center image. So, basically I obtained the following result:



From the previous result I experimented with a bunch of things, for example setting the displacement in the homographies, check the displacement and move the image by some value on X and some value on Y so I could negate the effects of the displacement and have images that basically are aligned to the Y axis, etc. Unfortunately, none of this worked. So, the idea that worked for me was to realign the origin, which means move the images by a specific offset that even when the homography is applied the image remains fully visible. The important aspect of doing this is that the property that was shown in the previous image is kept, which means that there is a point at which if taken as the origin and an image is placed from there a perfect alignment would exist between the image placed and the section of the warped image that already exists from the “realigned origin point”. This points seen as the red dot in the previous image.

By following the previously described idea of realigning the origin point I was able to stitch the images, now the only challenge that persisted was to find the bounding coordinates of the resulting image. For this I used the library function “warp_coords” that given the homography and the target size I would receive back the coordinates of all points after warped, for example at coords[0,offset,offset] I could see the coordinate row-wise where the coordinate would end. Logically, coords[1,offset,offset] indicated the place where the point would end column wise. This helped to find the coordinates that bounded the resulting stitched image because now the only thing missing was to find the biggest and smallest column and row, between the four points of the stitched image. This is done through the “smart” use of the offset to help with the calculations, the codes is:

```
result,H1 = warp(space,img_p,img2_p)
coords = tf.warp_coords(H1, space.shape[:-1])
min_r = min(int(coords[0,offset,offset]),int(coords[0,offset+img.shape[0],offset]+1),offset,offset+img2.shape[0])
max_r = max(int(coords[0,offset,offset]),int(coords[0,offset+img.shape[0],offset]+1),offset,offset+img2.shape[0])
min_c = min(int(coords[1,offset,offset]),int(coords[1,offset+img.shape[0],offset]+1),offset,offset+img2.shape[1])
max_c = max(int(coords[1,offset,offset]),int(coords[1,offset+img.shape[0],offset]+1),offset,offset+img2.shape[1])
```

**In order to get the two pairs of stitching the process is repeated twice.*

Problem 3

For this problem I basically had the idea of warping, as done in problem 1, in order to modify the size of the src image to a desired size. Then by mapping to the correct size all that was needed was to analyze the selected area by the user to determine the “degree of green” for every pixel (i.e. the fraction that green represents of the entire addition of intensities among all channels). So, basically we take the target area and do the division “green_channel of pixel i,j/addition of intensities of all channels in pixel i,j ”, then the next step was to apply a mask to the banner/src image to set those pixels where the fraction of green is not higher than 0.5 to zero which essentially applies a mask. Lastly, the only thing needed is to add the masked src image to the area bounded by the coordinates selected and observe the results.

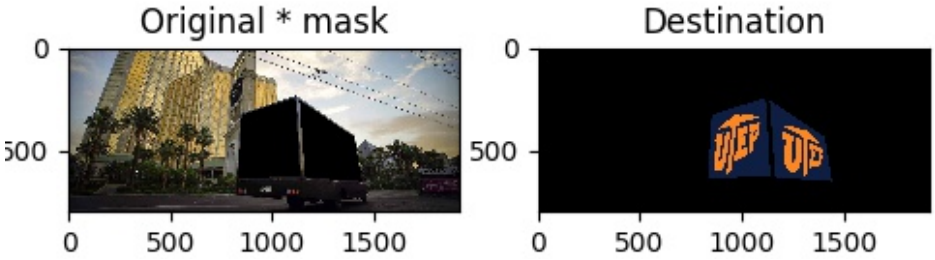
III. Experimental Results

Problem 1:

- Original Image

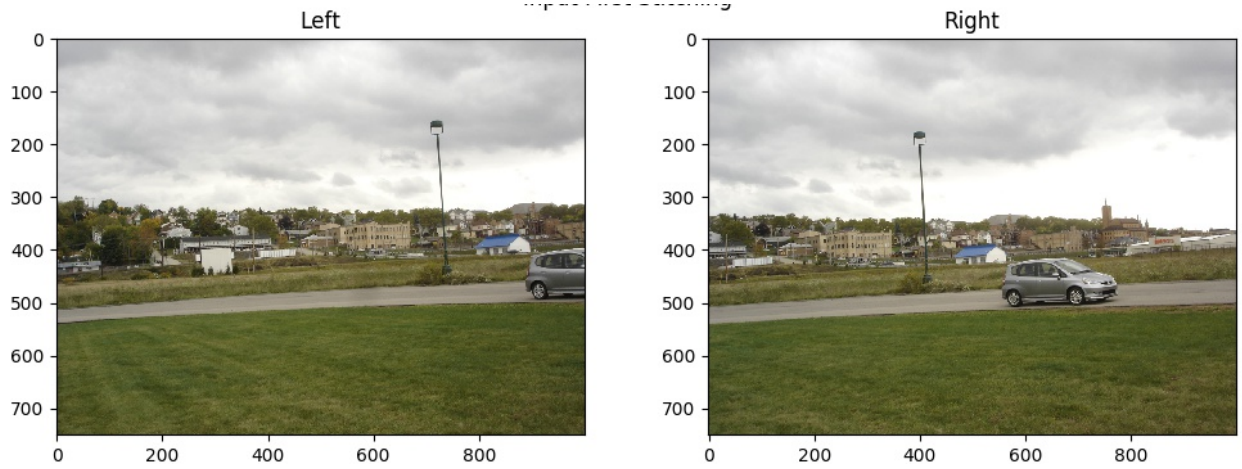
- Resulting Image

- Transitions

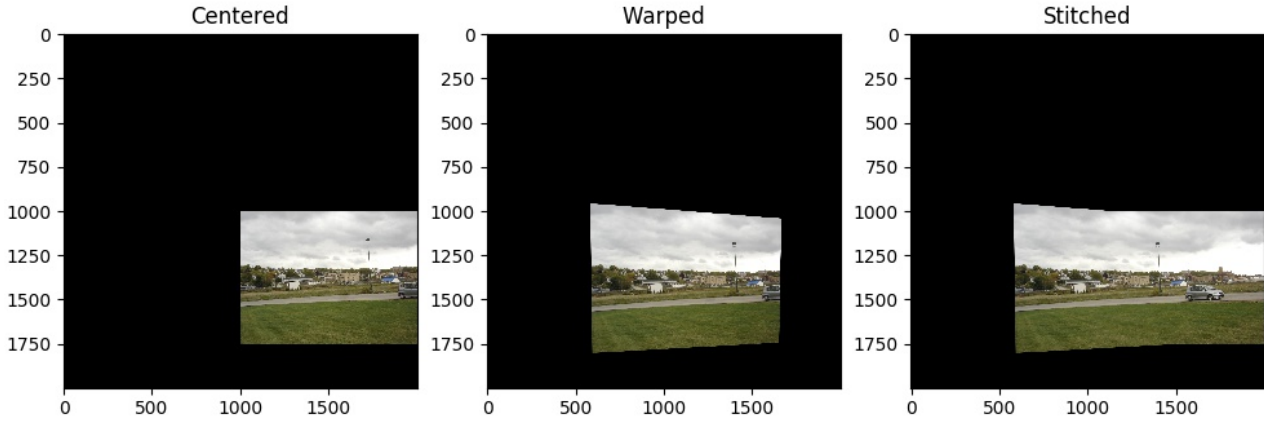


Problem 2

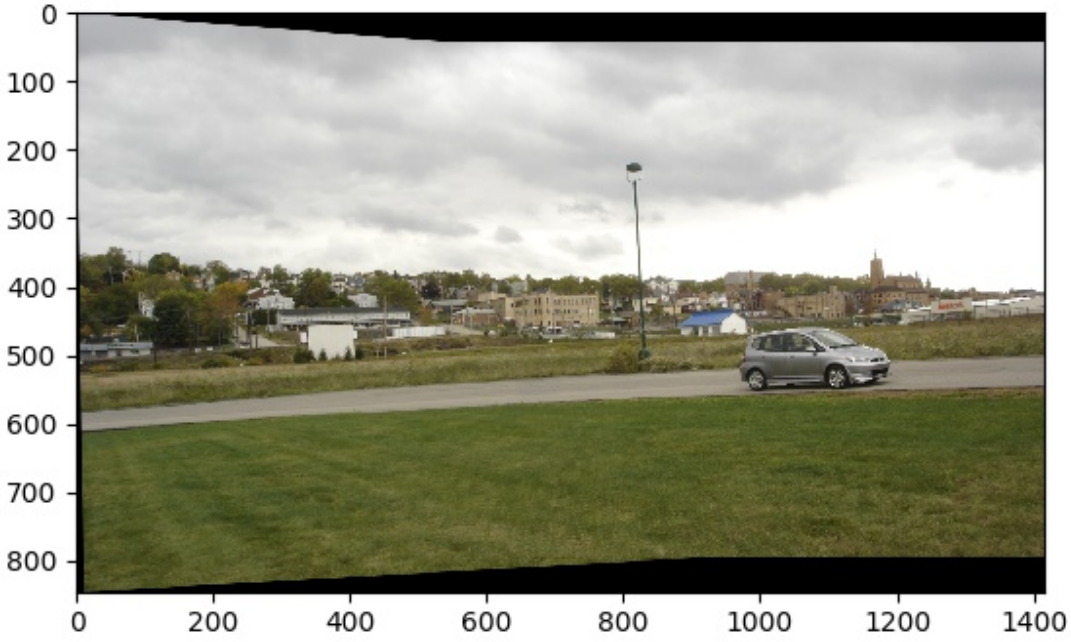
- First Stitching Input



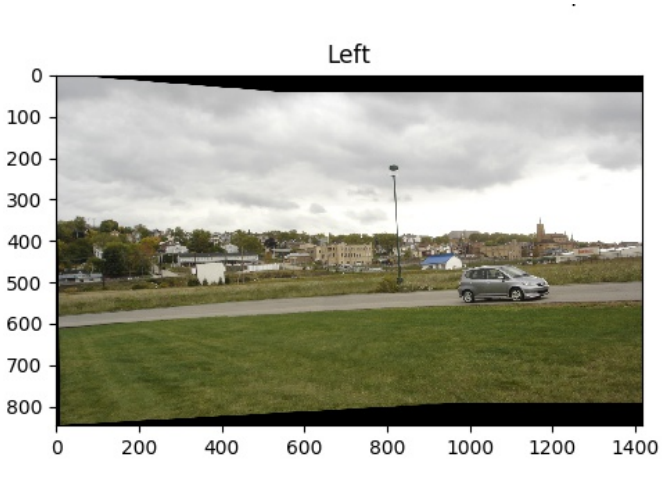
- First Warping Output



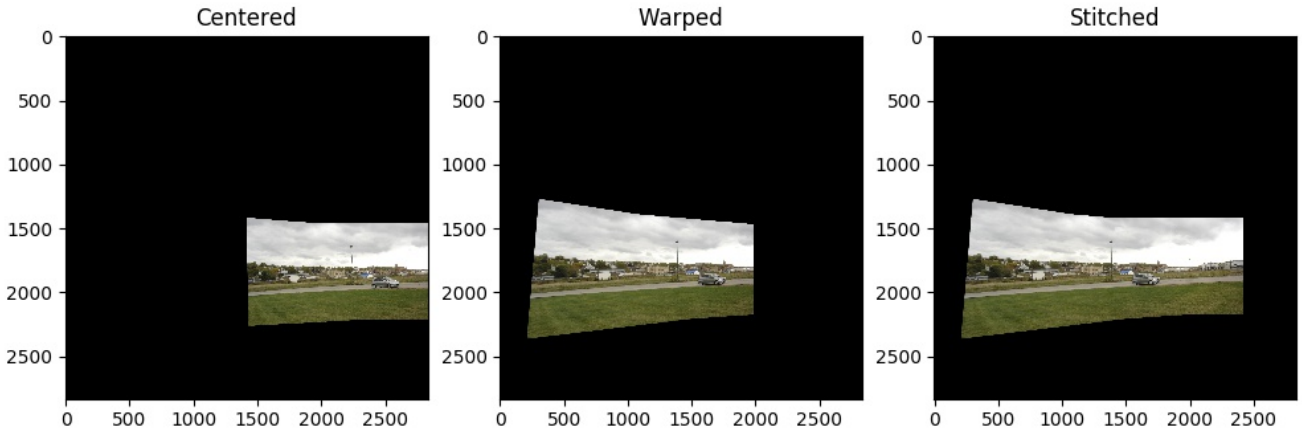
- First Stitching Result



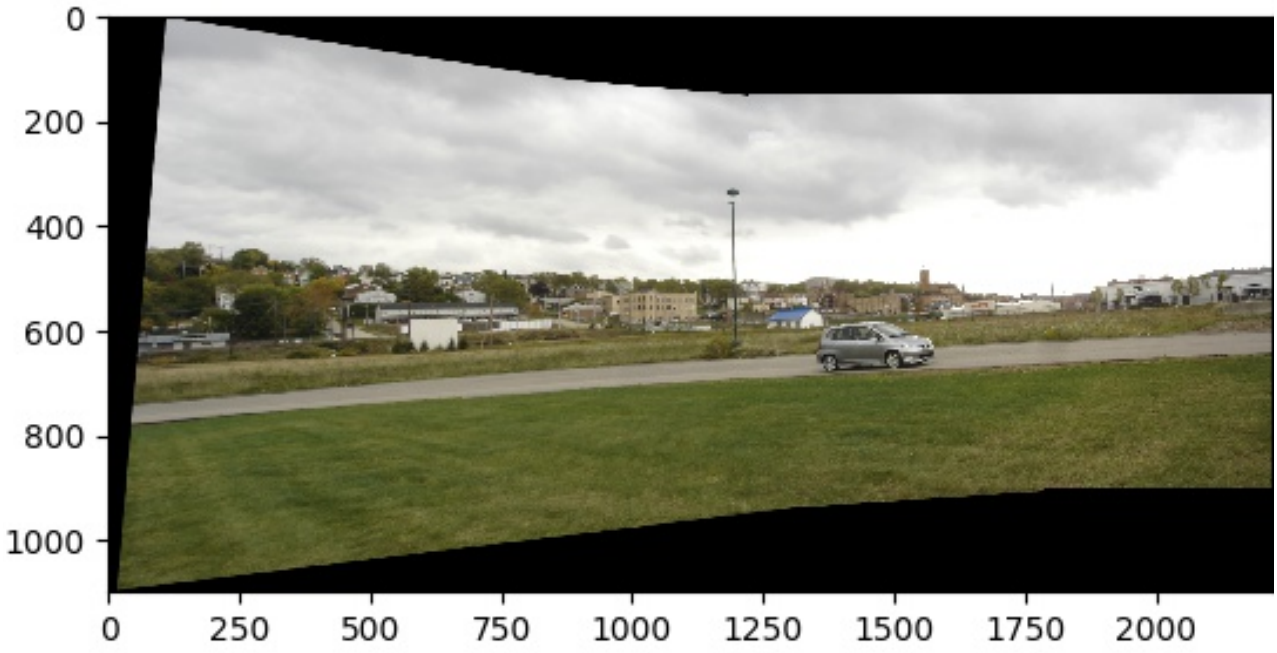
- Second Stitching Input



- Second Warping Output

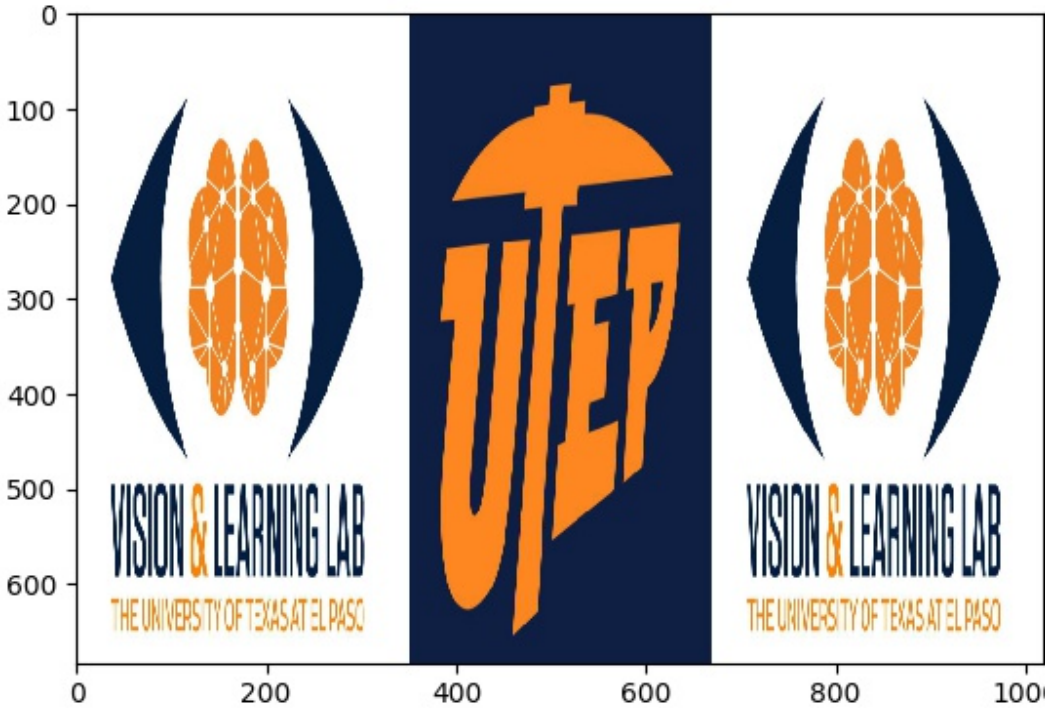


- Second Stitching Result



Problem 3

- Warped Banner

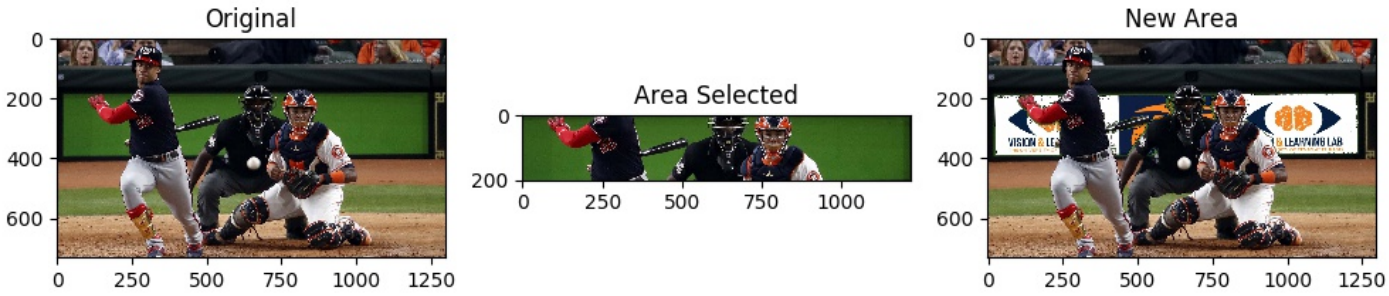


- Transition

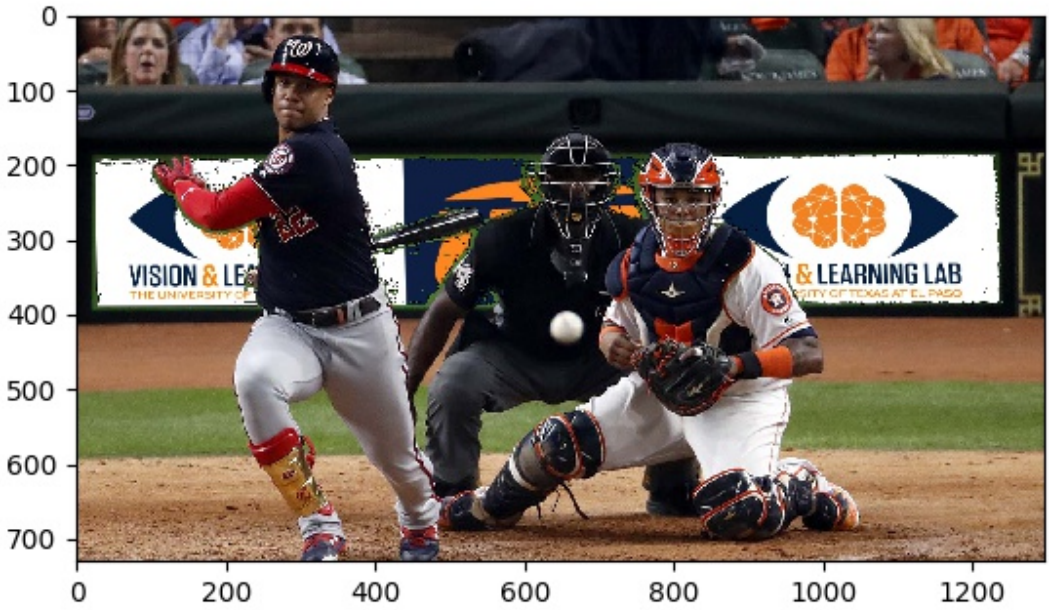
- Resulting Image

- Warped Banner

- Transition



- Resulting



IV. Discussion of Results

As seen in the results all the algorithms worked nicely, particularly the stitching looks quite good. We can see that in the case of algorithm 3 the masking work well but still there are areas of the green screening with the baseball picture that are not painted well, theoretically this is because the threshold as in the pixel level shading could dramatically change the share of green of some pixels. We can also observe for problem 3 that the gaussian blurring works well and we can see that the banner blends better than if placed as it originally looked. With the uniform green screening as shown in the alternative results for problem 3 it can be seen that the algorithm works really well as long as the background is uniform.

For problem 2, it is clear that taking the maximum intensity at every pixel on the aligned area does help to correct the possible effect of having different illumination, in fact as a result the fully stitched image looks natural. Naturally, the results are point dependent as both the results presented in the description of the lab and those presented in this report don't exactly have the same shape but do have the same look.

For problem 1 it is observable how the results are those desired and really nothing else has to be done.

V. Appendix

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 21 22:51:21 2021

@author: oscargalindo
"""

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

```
from scipy import signal

def correlate2d_scipy(image,filt):
    if len(image.shape) == 2:
        return signal.correlate2d(image, filt,mode='valid')
    else:
        r,c = filt.shape
        img = np.zeros((image.shape[0]-r + 1, image.shape[1]-c + 1, 3))
        for channel in range(image.shape[2]):
            img[...,channel] = signal.correlate2d(image[...,channel], filt,mode='valid')
        return (img/255).astype(np.float32)

def gaussian_filter(size, sigma):
    d = ((np.arange(size) - (size-1)/2)**2).reshape(-1,1)
    f = np.exp(-(d+d.T)/2/sigma/sigma)
    return f/np.sum(f)

def show(img,title=''):
    f,a = plt.subplots()
    f.suptitle(title)
    a.imshow(img)
    if title:
        f.savefig(title+".jpg")
    plt.show()

def show_multiple(images,title='',titles=[],figsize=(12,4)):
    f,a = plt.subplots(ncols=len(images),figsize=figsize)
    f.suptitle(title)
    for i in range(len(images)):
        a[i].imshow(images[i])
        if i < len(titles):
            a[i].set_title(titles[i])
    f.savefig(title+".jpg")
    plt.show()

def load_image(name,as_float=True):
    img = mpimg.imread(name)
    if as_float:
        return img/np.amax(img)
    else:
        return img

def get_points(num_points):
    return np.asarray(plt.ginput(n=num_points,timeout=0))

def warp(img,src,dst,shape = None):
```

```
    H = tf.ProjectiveTransform()
    H.estimate(src,dst)
    result = None
    if not shape:
        result = tf.warp(img, H.inverse)
    else:
        result = tf.warp(img, H.inverse,output_shape=shape)
    return result,H

def problem1():
    plt.close('all')
    #Image Loading
    img = load_image('mobile_billboard.jpg')
    img2 = load_image('utep720.jpg')
    ##############
    #Get Points
    show(img,'Original Image')
    print("Click four source points")
    src,src_2 = get_points(4),get_points(4)
    ##############

    #Computations
    dest2  = src
    src2 = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[img2.shape[1],img2.shape[0]]])
    warped,H1 = warp(img2,src2,dest2,shape=(img.shape[0],img.shape[1]))
    dest = src_2
    warped2,H2 = warp(img2,src2,dest,shape=(img.shape[0],img.shape[1]))
    warped += warped2
    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
    ##############

    #Display
    show_multiple([img*mask,warped,warped2,combined],'Problem1',
                ['Original * mask','Destination','Destination2','Original * mask +
destination'],
                )
    show(combined,'Resulting Image')
    ##############

def problem2(automatic=True):
    automatic = True
    plt.close('all')
```

```
    #Display Images for 1st stitching
    img = load_image('WF01.jpg')
    img2 = load_image('WF02.jpg')
    show_multiple([img,img2],'Input First Stitching',['Left','Right'])
    ##################################

    #Define Points First Stitching
    if not automatic:
        print("Click correspondences")
        src = get_points(12)
        img_p = src[::2]
        img2_p = src[1::2]
    else:
        img_p = np.array([[604.67338234, 379.29681609],
                [666.55126252, 344.34289904],
                [875.82964543, 374.10043988],
                [372.07164323, 356.73509445],
                [898.66898351, 317.52105153],
                [922.26814464, 443.32220864]])

        img2_p = np.array([[282.1093994 , 395.77670716],
                [344.90893857, 361.19145371],
                [545.91019538, 393.43328446],
                [ 34.77490072, 369.40850059],
                [567.27070332, 339.62639147],
                [587.22722512, 459.11012778]])
    ##################################

    #Change origin and reflect changes on points
    offset = max(img.shape[0],img.shape[1])
    space = np.zeros((offset*2,offset*2,3))
    space[offset:offset+img.shape[0],offset:offset+img.shape[1],:] = img
    img_p += offset
    img2_p += offset
    ##################################

    #Apply Homography and calculate limits of resulting image
    result,H1 = warp(space,img_p,img2_p)
    coords = tf.warp_coords(H1, space.shape[:-1])
    min_r = min(int(coords[0,offset,offset]),int(coords[0,offset+img.shape[0],offset]
+1),offset,offset+img2.shape[0])
    max_r = max(int(coords[0,offset,offset]),int(coords[0,offset+img.shape[0],offset]
+1),offset,offset+img2.shape[0])
    min_c = min(int(coords[1,offset,offset]),int(coords[1,offset+img.shape[0],offset]
+1),offset,offset+img2.shape[1])
```

```
    max_c = max(int(coords[1,offset,offset]),int(coords[1,offset+img.shape[0],offset]
+1),offset,offset+img2.shape[1])
    ##################################

    #Display results of first stitching
    r = result.copy()
    result[offset:offset+img2.shape[0],offset:offset+img2.shape[1],:] =
np.maximum(img2,result[offset:offset+img2.shape[0],offset:offset+img2.shape[1],:])
    first_stitch = result[min_r:max_r,min_c:max_c]
    show_multiple([space,r,result],'First Warping',['Centered','Warped','Stitched'])
    show(first_stitch,'Results First Stitching')
    ##################################

    #Load and show images for second stitching
    img3 = load_image('WF03.jpg')
    show_multiple([first_stitch,img3],"Input Second Stitching",["Left","Right"])
    ##################################

    #Define Points Second Stitching
    if not automatic:
        print("Click correspondences")
        src = get_points(12)
        imgc_p = src[::2]
        img3_p = src[1::2]
    else:
        imgc_p = np.array([[ 974.7361245,502.30896434],
         [1186.07022588,382.99609307],
         [1249.54672906,420.65623014],
         [ 961.37444229,437.46662036],
         [1097.49030633,430.51680009],
         [1365.52645099,423.94189488]])

        img3_p = np.array([[135.56565561,478.24584038],
         [353.01205177,359.24339772],
         [413.66546917,396.3644444 ],
         [123.1317045,410.39270247],
         [263.8693006,404.9737237 ],
         [520.57390025,400.63621346]])
    ##################################

    #Change origin and reflect changes on points
    offset = max(first_stitch.shape[0],first_stitch.shape[1])
    spaced = np.zeros((offset*2,offset*2,3))
    spaced[offset:offset+first_stitch.shape[0],offset:offset+first_stitch.shape[1],:] =
first_stitch
    imgc_p+=offset
```

```
    img3_p+=offset
    ##################################

    #Apply Homography and calculate limits of resulting image
    result,H2 = warp(spaced,imgc_p,img3_p)
    coords = tf.warp_coords(H2, spaced.shape[:-1])
    min_r = min(int(coords[0,offset,offset]),int(coords[0,offset+first_stitch.shape[0],offset]
+1),offset,offset+img3.shape[0])
    max_r =
max(int(coords[0,offset,offset]),int(coords[0,offset+first_stitch.shape[0],offset]
+1),offset,offset+img3.shape[0])
    min_c = min(int(coords[1,offset,offset]),int(coords[1,offset+first_stitch.shape[0],offset]
+1),offset,offset+img3.shape[1])
    max_c =
max(int(coords[1,offset,offset]),int(coords[1,offset+first_stitch.shape[0],offset]
+1),offset,offset+img3.shape[1])
    ##################################

    #Display results of first stitching
    r = result.copy()
    result[offset:offset+img3.shape[0],offset:offset+img3.shape[1]] =
np.maximum(img3,result[offset:offset+img3.shape[0],offset:offset+img3.shape[1]])
    show_multiple([spaced,r,result],'Second Warping',['Centered','Warped','Stitched'])
    second_stitch = result[min_r:max_r,min_c:max_c,:]
    show(second_stitch,'Results Second Stitching')
    ##################################

def problem3():
    plt.close('all')
    #Display and Get Points
    baseball = load_image('soto.jpg',as_float=False)
    vll_banner = load_image('vll_utep_720.jpg',as_float=False)
    vll_banner = correlate2d_scipy(vll_banner, gaussian_filter(10, sigma=0.3))
    show(baseball)

    print("Click four corners of Green Area")
    src = get_points(4)
    print(src)
    ########################

    #Warp Banner
    cols = int(np.amax(src[:,0])-np.amin(src[:,0]))
    rows = int(np.amax(src[:,1])-np.amin(src[:,1]))
    s = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) *
np.array([vll_banner.shape[1],vll_banner.shape[0]])
    dest = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[cols,rows]])
```

```
    w,H0 = warp(vll_banner,s,dest,shape=(rows,cols))
    show(w,'Warped Banner')
    ##########################

    #Show modified banner by mask and with color detection
    minr,minc = np.amin(src[:,1]).astype(int),np.amin(src[:,0]).astype(int)
    area = baseball[minr:minr+rows,minc:minc+cols,:]
    green_share = (area[:,:,1]/(np.sum(area, axis=2)+0.0000000001)).astype(np.float32)
    w[green_share < 0.5] = 0
    show(w)
    ##########################

    #Expand the banner to take all area of the image and combine with original
    src2 = np.array([[0, 0], [1, 0], [1, 1], [0, 1]]) * np.array([[w.shape[1],w.shape[0]]])
    warped2,H1 = warp(w,src2,src,shape=(baseball.shape[0],baseball.shape[1]))
    mask = np.expand_dims(np.sum(warped2,axis=2)==0,axis=2).astype(np.int)
    baseball_mod = (baseball*mask + warped2*255).astype(np.uint8)
    ##########################

    #Display Results
    show_multiple([baseball,area,baseball_mod],
                  'Transition',['Original','Area Selected','New Area'],
                  )
    show(baseball_mod,'Problem 3')
    ##########################

if __name__=="__main__":
    problem1()
    problem2()
    problem3()
```

VI. Conclusion

Overall I think this lab was quite challenging, but really good to understand homographies through trial and error displacements, warping, manipulation of sizes through warping, etc. Through this I observed the difficulty of manipulating images and the remarkable work that sophisticated pieces of software do to stitch images. In addition, I learned to think outside of the box a little more, since the solution to problem 2 came about from not trying to exploit the disadvantage that the warping made on the original image by moving it and disappearing some of the points, but

instead use it to my advantage. Also, I saw that it is better to apply linear changes to the data that is used to create warps and homographies than it is to deal with inverse homographies and the respective displacements. Lastly, I obviously learned about the software packages that may be used to create the operations needed to solve this lab as well as other problems. In conclusion, the lab was entertaining, challenging, and enlightening.

Statement of Academic Honesty:

“I certify that this project is entirely my own work. I
wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”