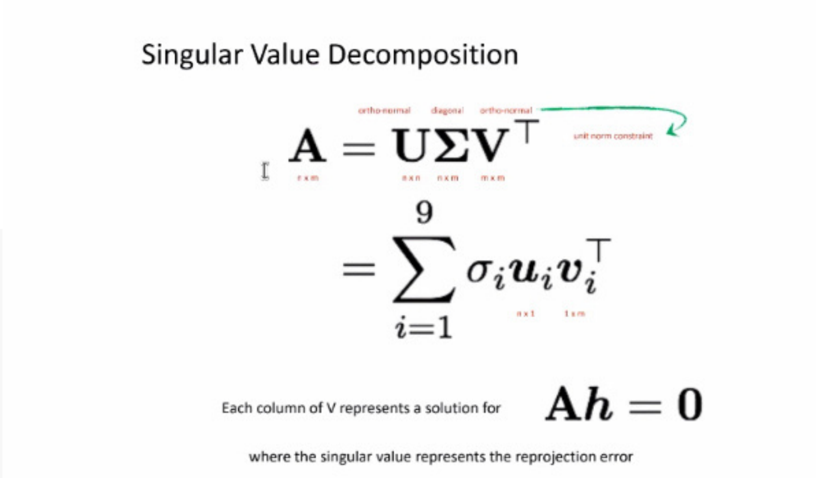
IMAGE MOSAICS

NAIRA YASSER 5299
RANEEM AHMED 5363
HOSSAM ELKADY 5446
LOGYN MEDHAT 5386

INTRODUCTION

1-IMAGE MOSAICS

In this part of the assignment, we will implement an image stitcher that uses image warping and homographies to automatically create an image mosaic. We will focus on the case where we have two input images that should form the mosaic, where we warp one image into the plane of the second image and display the combined views. This problem will give some practice manipulating homogeneous coordinates, computing homography matrices, and performing image warps. For simplicity, we will specify corresponding pairs of points manually using mouse clicks.




```python
def getPoints(image):
    img1 = Image.open(image)
    plt.figure(1,figsize=(20,20))
    plt.imshow(img1)
    print("Please click")
    x = plt.ginput(8)
    return x

pts_dst = getPoints(fileB)
pts_src = getPoints(fileA)

plt.close()


corrs = np.concatenate((pts_src, pts_dst), axis=1)
z = np.reshape(corrs, ((6, 4)))

aList = []
for corrr in z:
    p1 = np.matrix([corrr.item(0), corrr.item(1), 1])
    p2 = np.matrix([corrr.item(2), corrr.item(3), 1])

    a2 = [0, 0, 0, -p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2),
          p2.item(1) * p1.item(0), p2.item(1) * p1.item(1), p2.item(1) * p1.item(2)]
    a1 = [-p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2), 0, 0, 0,
          p2.item(0) * p1.item(0), p2.item(0) * p1.item(1), p2.item(0) * p1.item(2)]
    aList.append(a1)
    aList.append(a2)

matrixA = np.matrix(aList)

# svd composition
u, s, v = np.linalg.svd(matrixA)

# reshape the min singular value into a 3 by 3 matrix
h3 = np.reshape(v[8], (3, 3))

# normalize and now we have h
h3 = (1 / h3.item(8)) * h3

print('shvdH', h3)
```

first of all will get the correspondences vy this function then we will produce homography matrix using Singular value decomposition

WARPING FUNCTION:

A function that warps an image using the homography matrix H.**The image** to be warped and the **homography matrix** used to warp the image is given to the function **"def warp_image(image, H)"** then it returns first element: the warped images.
second element: the minimum u coordinate in coordinate space not image space
this means this could be a negative number, in other words
this is the amount of translation in the u direction.
third element: minimum v coordinate i.e. the translation in v direction.

```python
def warp_image(image, H):

    H_inv = np.linalg.inv(H)
    H_inv = H_inv / H_inv[2,2]
    # u == x
    # v == y


    orig_u_range = np.arange(image.shape[1])
    orig_v_range = np.arange(image.shape[0])

    _, transformed_image, = transform_grid(orig_u_range , orig_v_range, H)

    min_u=int(np.min(transformed_image[:,0]))
    max_u=int(np.max(transformed_image[:,0]))
    min_v=int(np.min(transformed_image[:,1]))
    max_v=int(np.max(transformed_image[:,1]))

    mapped_u_range = np.arange(min_u, max_u)
    mapped_v_range = np.arange(min_v, max_v)



    target_image = np.zeros((max_v-min_v, max_u-min_u,3))


    transformed_points, inv_transformed_image = transform_grid(mapped_u_range, mapped_v_range, H_inv)

    def fill_channel(target, channel, batch_size=64):
        I_cont = RectBivariateSpline(orig_v_range, orig_u_range, image[:,:,channel])

        n_iters =int( len(inv_transformed_image) / batch_size )

        for i in range(n_iters + 1):
            start = i * batch_size
            end = (i+1) * batch_size

            mapped_u_batch = inv_transformed_image[start:end, 0].ravel()
            mapped_v_batch = inv_transformed_image[start:end, 1].ravel()

            u_batch = transformed_points[start:end, 0].ravel()
            v_batch = transformed_points[start:end, 1].ravel()

            target[v_batch-min_v, u_batch-min_u, channel] = I_cont(mapped_v_batch, mapped_u_batch, grid=False)

    fill_channel(target_image, 0)
    fill_channel(target_image, 1)
    fill_channel(target_image, 2)
    #cv2_imshow(target_image)

    return target_image, min_u, min_v
```

TRANSFORMING POINTS:

Create a matrix the same size of the original image and place the transformed point

```
def transform_grid(u_range, v_range, H):

    grid_u, grid_v = np.meshgrid( u_range, v_range )

    u_flat = np.expand_dims(np.ndarray.flatten(grid_u), 1)
    v_flat = np.expand_dims(np.ndarray.flatten(grid_v), 1)
    points = np.concatenate([u_flat, v_flat],1)

    return points, transform_points(points, H)
```

Transforming all the points by adding 1 to the point matrix and multiplying it with the H matrix to produce the new points

```
def transform_points(points, H):

    ones = np.ones((points.shape[0], 1))
    points = np.concatenate([points, ones], 1)

    mapped_points = np.dot(points, H.T)
    mapped_points[:,:-1] /= np.expand_dims(mapped_points[:,-1],1)
    mapped_points = mapped_points[:,:-1]

    return mapped_points
```

STITCHING:

image stitching or photo stitching is the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image. Commonly performed through the use of computer software, most approaches to image stitching require nearly exact overlaps between images and identical exposures to produce seamless results, although some stitching algorithms actually benefit from differently exposed images by doing high-dynamic-range imaging in regions of overlap

STEPS:

Create a new image of same size as warped image +original image then we fill the matrix by the original image pixels and then put the pixels of warped image pixels and floor the image for brightness difference

```
def stitch(img2path,img1path,h):
    image=0
    image2 = cv2.imread(img2path)
    image1 = cv2.imread(img1path)
    warpped_image_1, min_u, min_v = warp_image(image2, np.array(h))

    res = np.zeros((warpped_image_1.shape[0] + image1.shape[0],
                    warpped_image_1.shape[1] + image1.shape[1], 3))

    shift_u_1 = min_u if min_u>0 else 0
    shift_v_1 = min_v if min_v>0 else 0



    shift_u_2 = -min_u if min_u<0 else 0
    shift_v_2 = -min_v if min_v<0 else 0
    res[shift_v_2:image1.shape[0] + shift_v_2      ,    shift_u_2:image1.shape[1] + shift_u_2, :] = image1


    res[shift_v_1:warpped_image_1.shape[0]+shift_v_1     , shift_u_1:warpped_image_1.shape[1]+shift_u_1-140 , :] = warpped_image_1[:,0:warpped_image_1.shape[1]-140,


    res = res[0:np.maximum(image2.shape[0], image1.shape[0]+100),
              0:np.maximum(image2.shape[1], image1.shape[1])+500]

    cv2.imwrite('final.png',res)
    #cv2_imshow(res)
    im = Image.open(r"final.png")
    im1 = im.crop((30, warpped_image_1.shape[1]-image1.shape[1], res.shape[1], res.shape[0]))
    im1.save('test.png')
    final=cv2.imread('test.png')
    final=cv2.medianBlur(final,3)
    cv2_imshow(final)
```

FINAL IMAGE:

BONUS:

Instead of stitching 2 images we stitched 3 images that overlap with each other.

- 1st we Stitched 2 images together following the previous steps.
- then we got correspondences between the output of stitching the 2 first images and the third image
- Finally we stitched the output with the third image creating the final output