ECS 174 Spring 2020 (05/11) Problem Set #2
Bingwei Wang 914683636
Nan Chen 915218152

Part 1.

(1) We believe that the grouping algorithm graph-cuts, would be the most appropriate grouping algorithm to recover the model parameter hypotheses from the continuous vote space. The reason is that each edge of it has its own affinity weight, and pixels are linked in pairs. Additionally, graph-cuts connect shapes on edges with high affinity while mean shift and k-means does not, which works well when shapes are defined by boundary points in Hough Transform.

But here comes a problem which we find mean-shifts might have better performance. Graph-cuts use a pixel-based energy function to compute the similarity of neighboring pixels, while in a continuous space we cannot count the number of points or their neighbors, so this would not work. For mean-shift, it is non-parametric and does not segment the vote space. We do not need to discretize of the vote space in the mean-shift algorithm because the algorithm itself already does it.

We do not choose k-means because in continuous vote space, it is hard(impossible) to select parameter k.

(2) After using K-means to cluster those points into two groups, we will have two center points for outputs in two clusters. Each center point will be randomly positioned at the beginning. After that, new positions of two center points will be updated by calculating the least squared Euclidean distance from feature points in its own cluster. The process will keep repeated until feature points are divided equally in two groups, belonging to those two center points, which means half points will be close to one center point, and half will be close to the other center point.

(3) Variables and Pseudocode:

```
variables:
    blobs: the set of the blobs.
    areas: the set of the areas of blobs.
    k: the number of groups
    clusters: the set of groups, each contains some blobs, size(clusters) = k.
    centroids: the list of means.

pseudo_code:
group_blobs(blobs,k)
    areas = get_areas(blobs,size(blobs))
    clusters = k_means(blobs,areas,k)
    return clusters
endfunc

get_areas(blobs,n)
    for blob = 1:n
        areas[blob] = 0
        for each pixel in blob
            areas[blob]++
        endfor
    endfor
    return areas
endfunc

k_means(blobs,areas,k)
    centriods = []
    cluster = zeros(k)
    cluster[each] = random blobs in the range
    // Loop until converge or fixed iterations
    while
        for i = 1:k
            centroids[i] = rand(min(areas),max(areas))
        endfor
        for each blob
            // Use diff to determine
            find nearest centroid[]
            assign the blob to that cluster
        endfor
        for i = 1:k
            new centroid = mean of all blob to that cluster
        endfor
    endwhile
    return clusters
endfunc
```

□

Part 2.

(a) The code is written in file **get_correspondences.m**. The function has two inputs of two images and the number of points the user wants to pick. The outputs are two 2 x N point matrices of selected points.

(b) The code is written in file **computeH.m**.

(c) The code is written in file **warpImage.m**.

(d) Images:

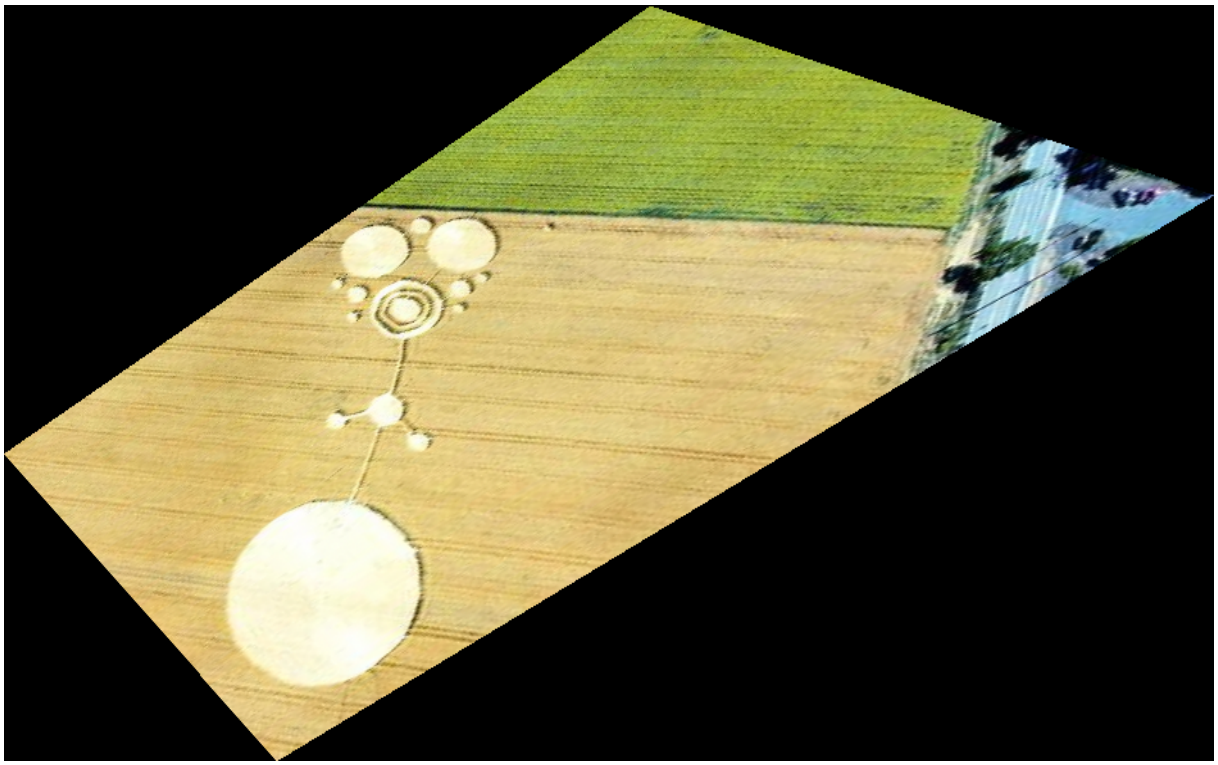

Figure 1: crop_warp

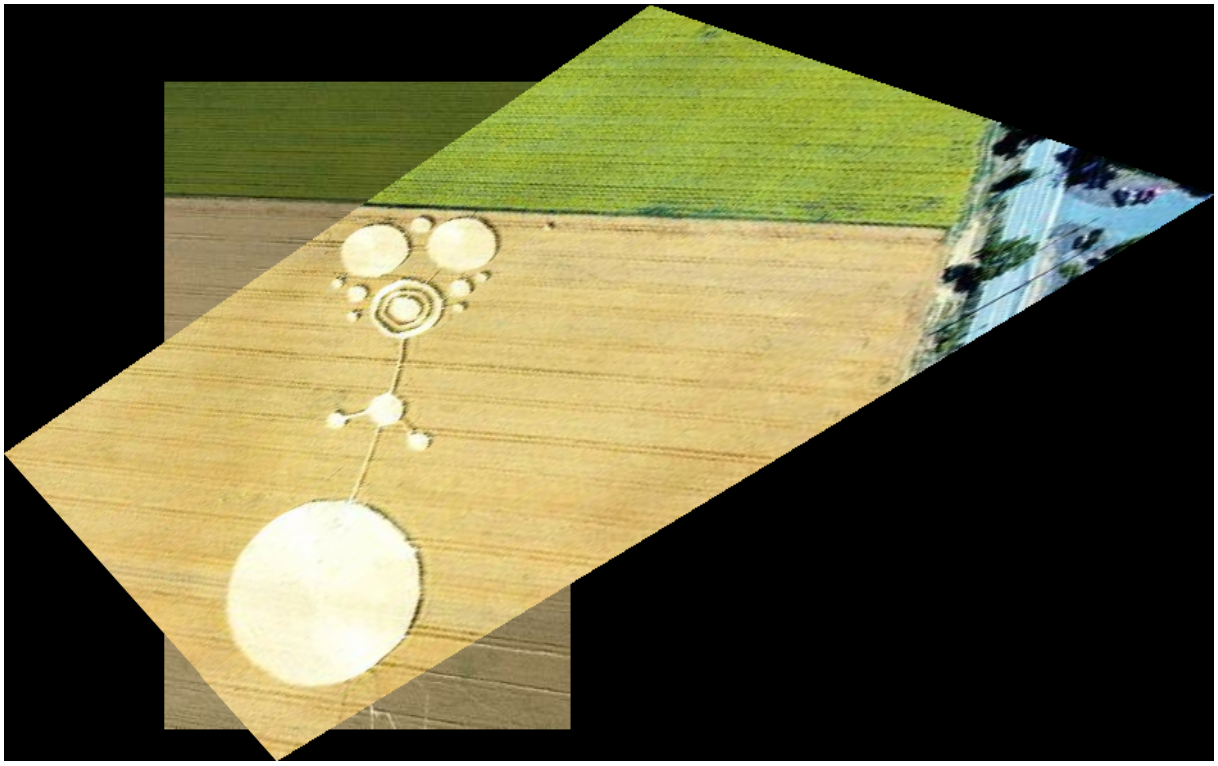

Figure 2: crop_merge

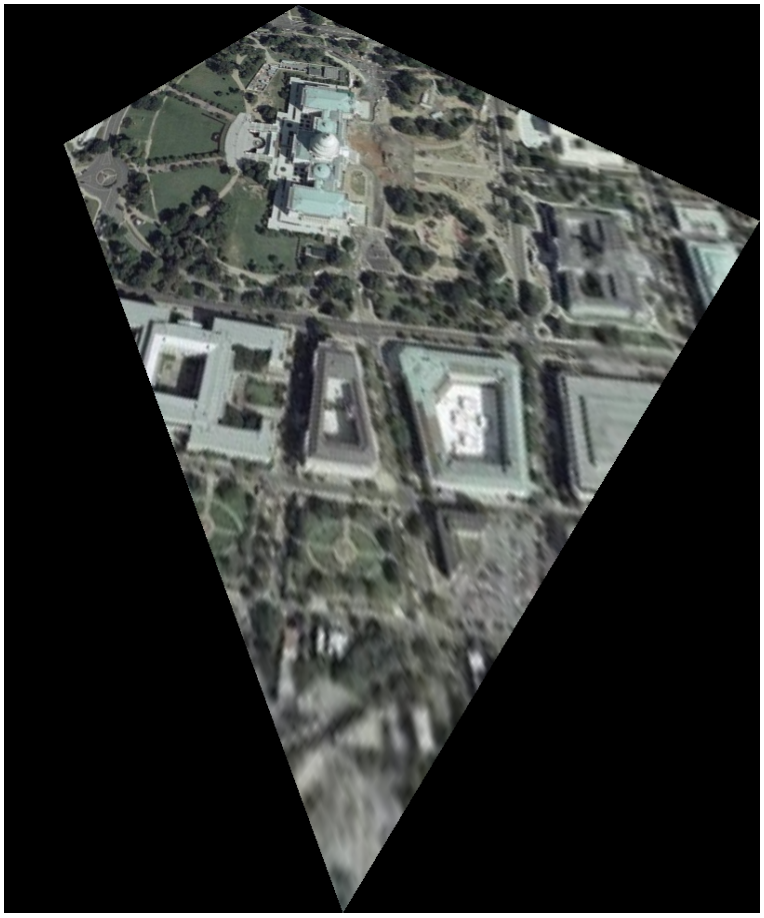

Figure 3: wdc_warp

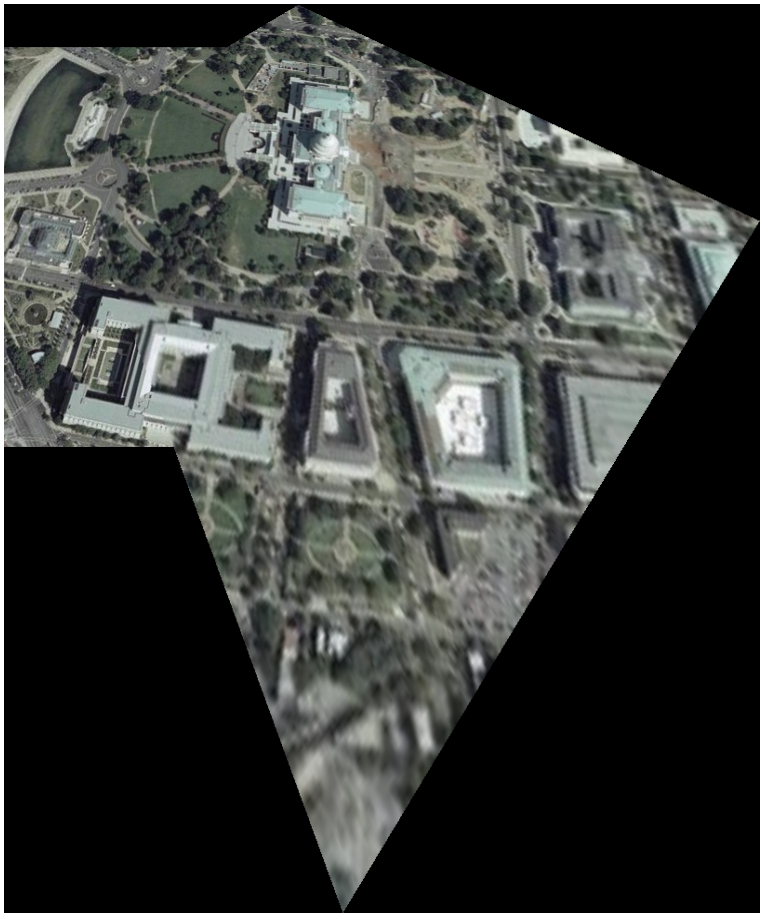

Figure 4: wdc_merge

The data points1 and points2 are saved in the file **points.m**.

(e) Images:

Figure 5: town_source_1

Figure 6: town_source_2

Figure 7: town_mosic

Source: https://www.bilibili.com/bangumi/play/ep264278.

Another example (can be omitted):

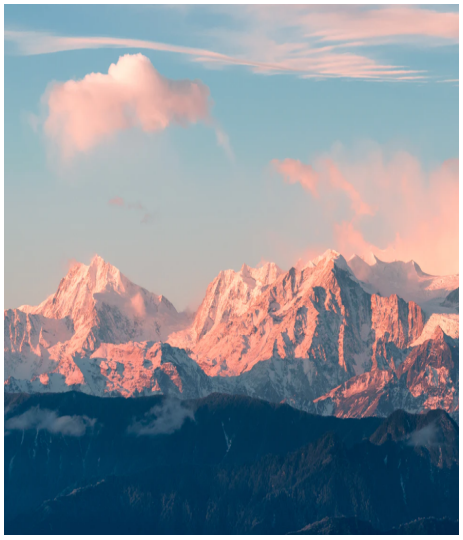

Figure 8: mountain_source_1

Figure 9: mountain_source_2

Figure 10: mountain_mosic

Source: https://unsplash.com/photos/Y8lCoTRgHPE.

(f) Images:

Figure 11: plate

Figure 12: car

Figure 13: car_mosic

Source: ScreenShots from Game Forza Horizon4.
Take the license plate picture, and mosic it on the car. □

Part 3.

(a) Images:

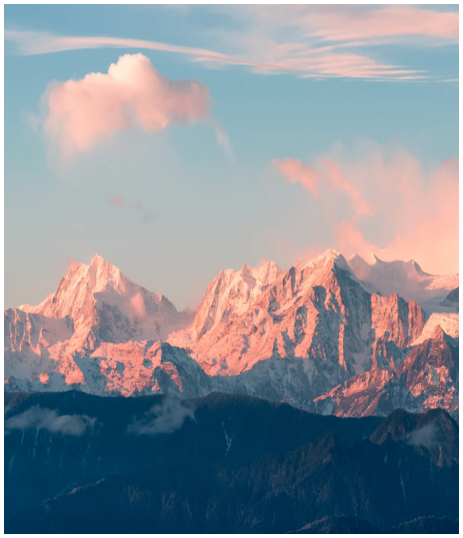

Figure 14: mountain_source_1

Figure 15: mountain_source_2

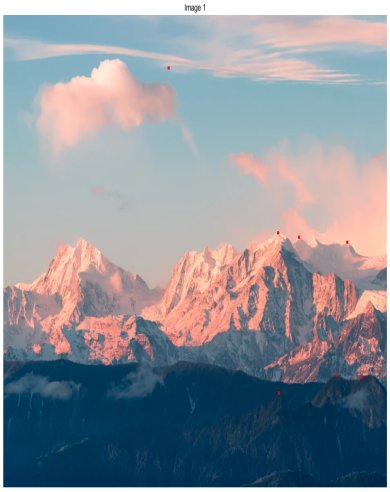

Figure 16: mountain_select_1

Figure 17: mountain_select_2

Figure 18: mountain_mosic_original

Figure 19: mountain_mosic_RANSAC

Source: https://unsplash.com/photos/Y8lCoTRgHPE.

Code: **RANSAC_script.m** and **RANSAC.m**.

Explanation: We implemented a new function **bestH = RANSAC(t1, t2)** to calculate the new H value using RANSAC algorithm. From Figure 16 and Figure 17, we can see there is one bad point selected. Thus, in Figure 18, the image is "broken" and no longer seems "stitch". In Figure 19, we use RANSAC method, and that can eliminate the bad point; therefore, gives us the correct stitch.

Implementation: The user can use the script **RANSAC_script.m** and change the image names to get results with RANSAC algorithm.

(b) Images:

Figure 20: tiles_source

Figure 21: tiles_fronto

Source: https://www.wayfair.com/home-improvement/pdp/msi-bianco-dolomite-marble-mosaic-tile-mvp4112.html?piid=49665325.

Code: **fronto_script.m** and **get_correspondences_fronto.m**.

Explanation: We revised the **get_correspondences()** function to only get the four points of one image. Then, we use these four points as the first input points to compute H, and we selected the four corners of the image as the second input points to compute H. Lastly, as usual, we use the image twice and H to get the results.

Implementation: The user can use the script **fronto_script.m** and change the image name to get results. □