ECS 174: Computer Vision, Spring 2019 Problem Set 2

Gabriel Brusman, Wyatt Robertson

914060880, 913920853
University of California
Davis, CA 95616

1. Short Answer

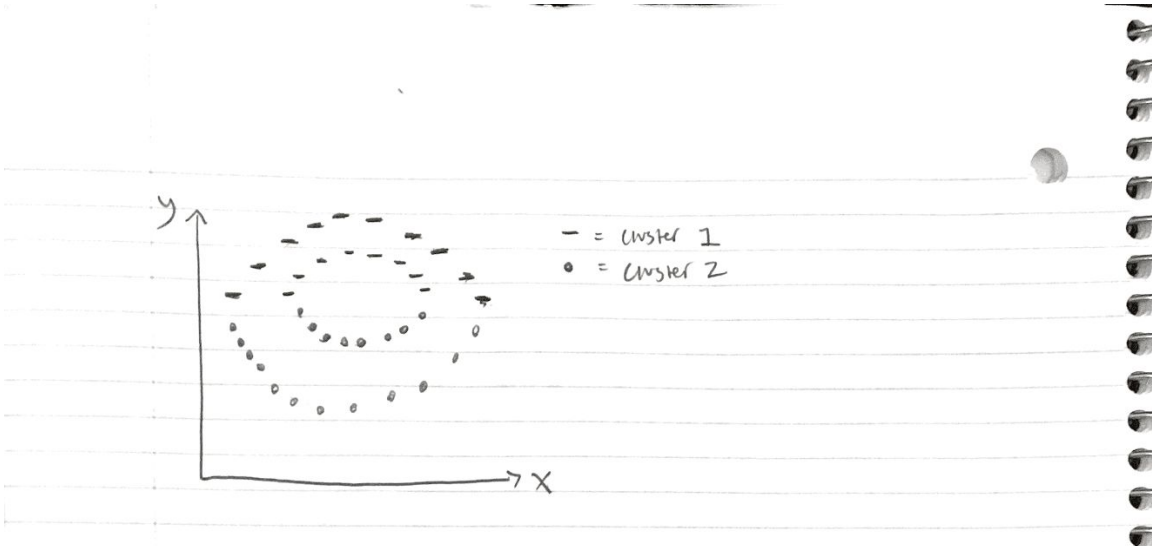
1.1 When using the Hough Transform, we often discretize the parameter space to collect votes in an accumulator array. Alternatively, suppose we maintain a continuous vote space. Which grouping algorithm (among k-means, mean-shift, or graph-cuts) would be appropriate to recover the model parameter hypotheses from the continuous vote space? Briefly describe and explain.

The best grouping algorithm for this situation is the mean-shift algorithm. We can eliminate k-means as the correct choice since it takes in the number of clusters you want as an input, so the algorithm naturally discretizes the vote space. Graph-cuts breaks the space into segments, which is also a natural discretization. Additionally, graph cuts uses a pixel-based energy function to compute the similarity of neighboring pixels, but in a continuous space there will be an infinite number of points, each with an infinite number of neighbors, so this would not work. Mean-shift is non-parametric, and does not segment the vote space in any way. It simply finds features, initializes windows around them, and continuously shifts towards the local maxima for each window. There is no discretization of the vote space in the mean-shift algorithm.

1.2 Consider Figure 2 below. Each small dot denotes an edge point extracted from an image. Say we are going to use k-means to cluster these points' positions into k=2 groups. That is, we will run k-means where the feature inputs are the (x,y) coordinates of all the small dots. What is a likely clustering assignment that would result? Briefly explain your answer.

Since k = 2, the algorithm will make 2 clusters. K-means assigns each observation to the cluster whose mean has the least squared Euclidean Distance, and then updates the new center of each cluster. Then each observation will belong to the cluster with the nearest mean. Note that each concentric circle will have the same mean. Thus, with 2 points/clusters, the resulting clusters will be two nested/concentric semicircles.

E.g.



1.3 Suppose we have access to multiple foreground ‘blobs’ within a binary image. Write pseudocode showing how to group the blobs according to the similarity of their area (# of pixels) into some specified number of groups. Define clearly any variables you introduce.

PSEUDOCODE:

```
//MAIN FUNCTION
group_blobs_based_on_area(img, k):
        areas = find_connected_components(img)
        k_means(areas, k)  // this will do the actual clustering of the blobs.
endfunc


 find_connected_components(img):
        labels = {}  //empty dictionary or map
        areas = []  // empty list
        component_label = 1


        for y <- 1 to height(img):
                for x <- 1 to width(img):
                        if img[x,y] == 1:  // if the pixel is part of a blob
                                labels[(x,y)] = component_label
                                areas[component_label] = 1
                                recursively_label(x, y, component_label, img)
                                component_label++
                        endif
                endfor
```

```
        endfor
        return areas
endfunc


recursively_label(x, y, component_label, img, labels):
        for each neighbor position (x', y') of img(x,y):
                if img(x',y') == 1 and (x', y') not in labels:
                        labels[(x', y')] = component_label
                        areas[component_label]++
                        recursively_label(x', y', component_label, img, labels)
                endif
        endfor
endfunc


k_means(areas, k):
        centroids = []
        //initialize clusters to random points in the range of the dataset
        for i <- 1 to k:
                centroids[i] = random(min(areas), max(areas))
        endfor

        clusters, changed = assignToClusters(centroids, areas)
        //keep updating centroids until there is no change in cluster assignments
        while changed == True
                centroids = updateClusterCenters(centroids, clusters)
                clusters, changed = assignToClusters(centroids, areas)
        endwhile
endfunc

assignToClusters(centroids, areas):
        //assign each point in the dataset to a cluster
        clusters = {} //dictionary or map that holds lists of data points (each list represents a cluster)
        changed = False
        for i <- 1 to size(areas):
                smallestDist = inf
                closestCentroid = 0
                for j <- 1 to size(centroids):
                        if (dataset[i] - centroids[j])^2 < smallestDist:
                                smallestDist = (areas[i] - centroids[j])^2
                                closestCentroid = j
                                changed = True
                        endif
                endfor
```

```
                clusters[closestCentroid].append(i)
        endfor
        return clusters, changed
endfunc


updateClusterCenters(centroids, clusters)
        //find updated cluster centers
        for i <- 1 to size(clusters):
                sum = 0
                for j <-1 to size(clusters[i]):
                        sum += clusters[i][j]
                endfor
                centroids[i] = sum / size(clusters[i])
        endfor
        return centroids
endfunc
```

VARIABLES:

labels: a map whose keys are coordinate tuples (x,y), and whose values are integers that correspond to which connected component (blob) a pixel belongs to.

areas: a list of integers where each integer is the size of a connected component (blob).

component_label: an integer that labels a blob. The idea is to update component_label after we finish labeling every pixel in a blob. This way each blob will have its own label.

img: the binary image.

k: the number of groups we want in our output.

centroids: a list of size k that contains the centroid for each cluster

clusters: a map whose keys are indices (representing cluster labels), and whose values are lists containing the data points belonging to the cluster.

changed: a boolean value that represents whether or not the cluster assignments of the data points have changed.

smallestDist: a numerical value that represents the smallest euclidean distance of a data point to a centroid.

closestCentroid: a numerical value representing the label of the centroid a data point is closest to.

2. Programming Problems

2.1 Getting Correspondences

Matlab's *ginput* function was used to collect the coordinates of user selected points on two images of the same subject at different perspectives called correspondences.

2.2 Computing Homography Matrix

Correspondences collected from the user were used to estimate a 3x3 homography matrix which was used to warp the coordinates corresponding to the same points of interest in the input image to the reference image.

The homography matrix was estimated by setting up a least squares problem with a matrix **A**, which is defined below:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A} \quad \mathbf{h} \quad \mathbf{0}$$

$$\mathbf{2n \times 9} \quad \mathbf{9} \quad \mathbf{2n}$$

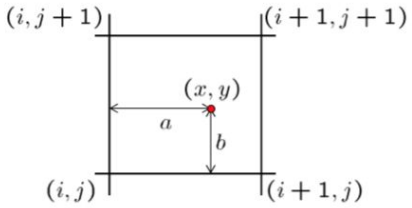
The eigenvector with the smallest associated eigenvalue of $\mathbf{A^TA}$ was taken to be the solution for **h**, which was then reshaped to a 3x3 matrix **H**. According to the prompt, estimation can sometimes perform better when coordinates are scaled between 0 and 2, however we did not scale our coordinates since our estimates looked acceptable.

2.3 Warping Between Image Planes

In this section, we take an input image, a reference image, and a 3x3 homography matrix H, and return a warped version of input image according to **H**, and the warped image merged together with the reference image.

To avoid holes, we used an inverse warp, which essentially works by mapping new coordinates in the warped space back to the original space using the inverted homography matrix. Sampling

was performed in all points in the destination bounding box from the corresponding pixels in the source image using bilinear interpolation. Specifically, intensities for each RGB channel's pixels were chosen using the following formula given in lecture:



$$
\begin{aligned}
f(x,y) = \quad & (1-a)(1-b) \quad f[i,j] \\
& +a(1-b) \quad f[i+1,j] \\
& \quad +ab \quad f[i+1,j+1] \\
& +(1-a)b \quad f[i,j+1]
\end{aligned}
$$

2.4 Example Applications

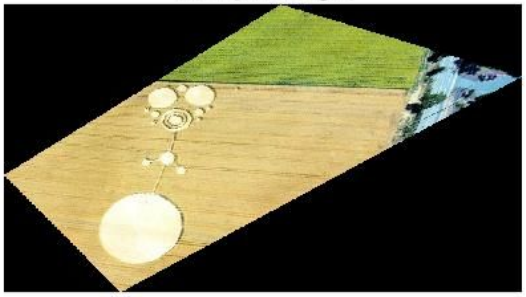
In this section we will show the reader two applications of the technologies described above. In the first example, we warp and merge the following images using predefined correspondences given in the prompt. Below are the input and reference images manipulated in the first example.

Input Image

Reference Image

Notice that the transformed image has different dimensions and shape than the original image.

Warp Image

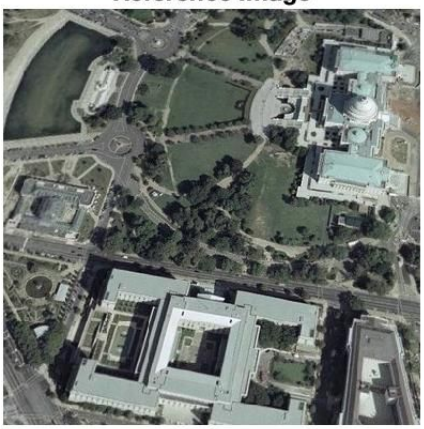


Merge Image

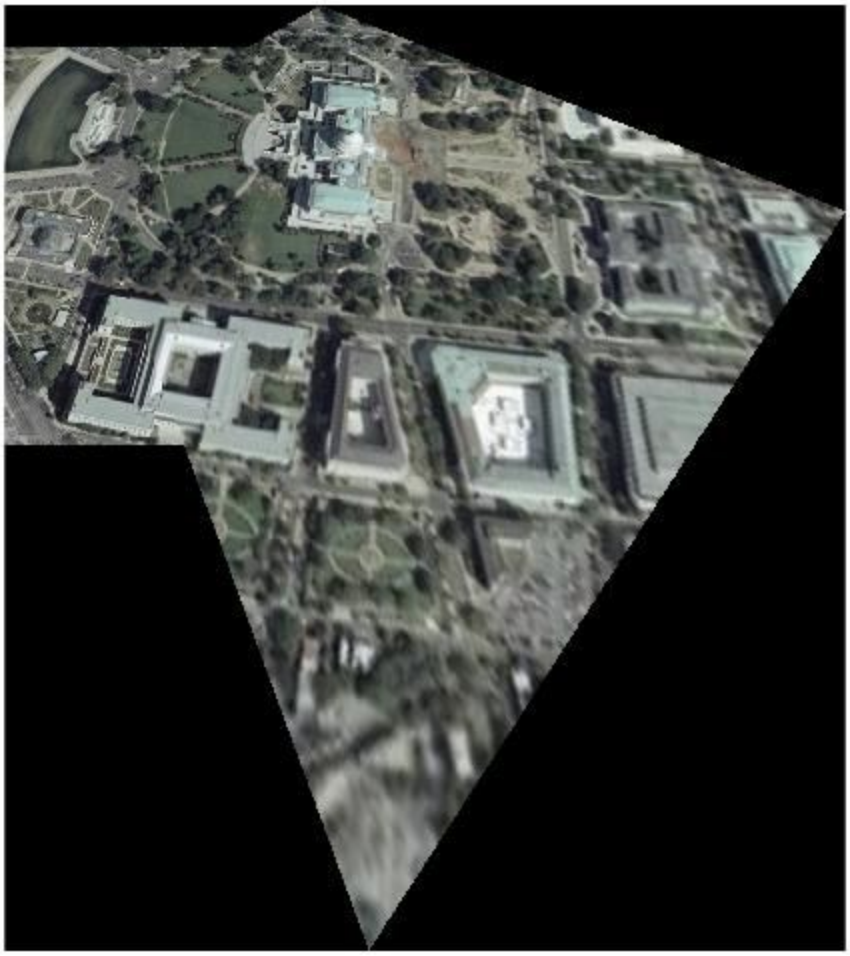
In the next example, we warp and merge the following images using our own chosen correspondences.

Input Image

Reference Image



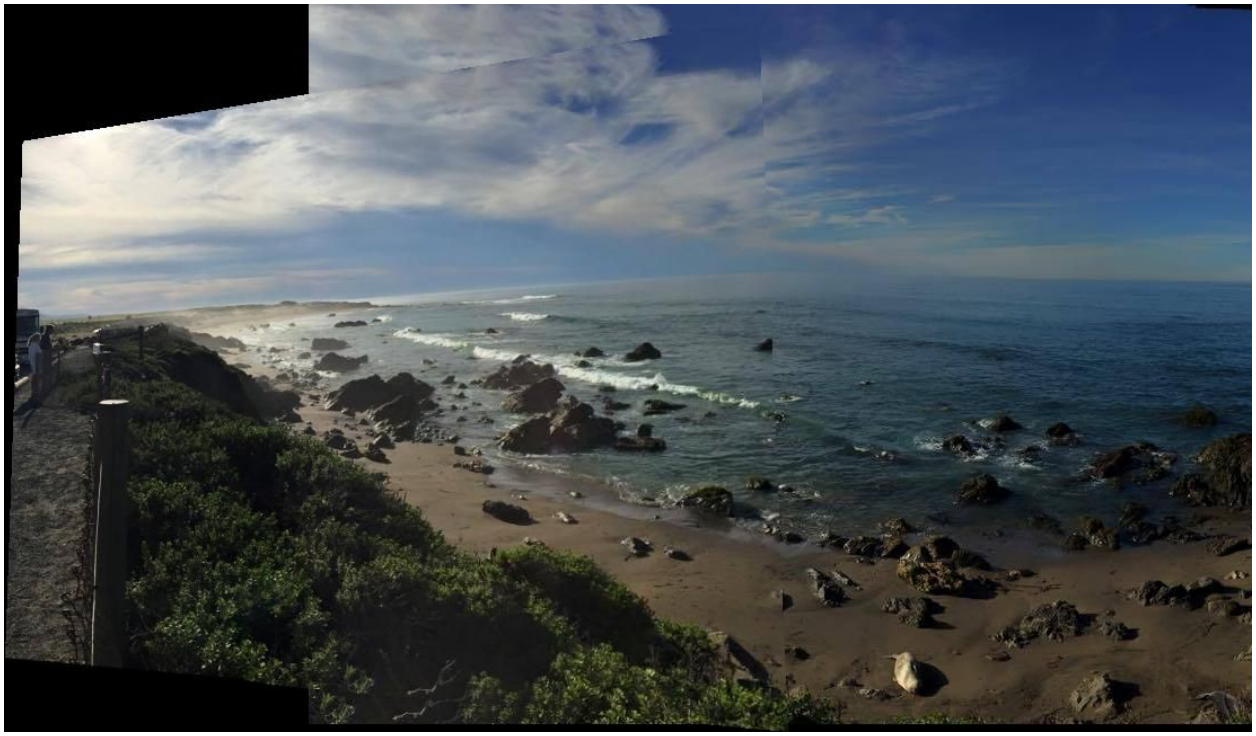
Twenty points were chosen around the perimeter of the grass field and largest building, as well as the two buildings that can be seen at the bottom of the reference image, and the two images were stitched together.



2.5 Additional Original Application

A mosaic was constructed using four original images of a beach through merging. In this example, merging appeared to perform well with the exception of the fourth image, where there is a disconnect in skyline with image three. This difference is most likely explained by the fact that image three and four were taken with the largest change in perspective in all four photos, and/or by imprecise inputs given by the user.

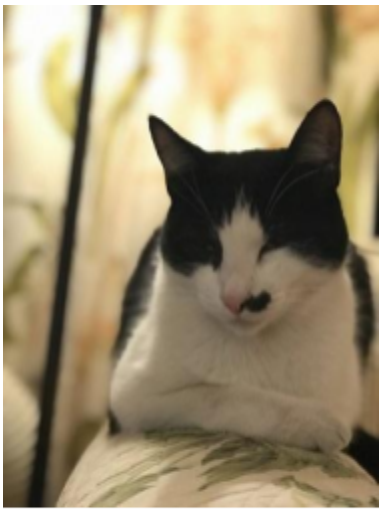
Here is a combination of two images that yielded better results. Points were chosen along the rocks where the water meets the beach. The seam between the two images is still slightly visible due to imperfect alignment of the clouds, but the photos appeared to have merged relatively well.



2.6 Warping Into Image Frames

In this section, we warp an input image into the frame of a reference image. This was accomplished by choosing points around the corner of the object we wanted to warp and choosing the corresponding points to be the corners of any frame.

In this example, we take an original photo of cat and warp it into a blank space on a billboard image.



Original Image

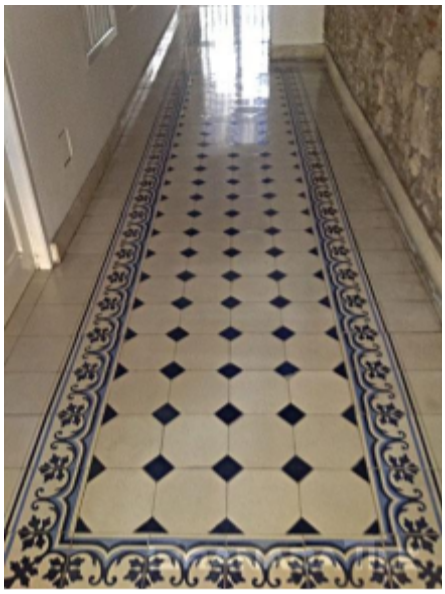
Frame Image

Resulting Image

Picture source:
https://www.marketingdonut.co.uk/media-advertising/billboards-and-outdoor-advertising/why-traditional-billboards-are-still-delivering-results

3. Extra Credit: Fronto-parallel View

For extra credit, we created an example application of image warping called a fronto-parallel view on a planar surface. **H** was solved for using the four corners of the planar surface and user chosen coordinates for the output image. Below you can see the output of what this photo might look like from a bird's eye view.



Original Image

Frontoparallel View

Picture source: https://www.molonytile.com/create-unique-designs-patterns-cement-tile/

4. Sources Cited

1. Yong Jae Lee. “Image warping and stitching”. University of California, Davis. 2 May, 2019.
2. Billboard Image. https://www.marketingdonut.co.uk/media-advertising/billboards-and-outdoor-advertising/why-traditional-billboards-are-still-delivering-results
3. Tile Image. https://www.molonytile.com/create-unique-designs-patterns-cement-tile/