Hin Chan
913645769

I. Short Answer Problems:

1. Mean-shift would probably be the best of the three. Since we want to work with a continuous space, being able to identify cluster without specifying how many expected clusters is especially important. K-means will need the user to specify k, which is how many clusters the user want the algorithm to form. Group-cut will need the user to specify assoc, which is the size of cluster. Since Hough Transform's vote is randomly scattered across the continuous space, there is no way for us to know how many clusters (k) there are or how big each cluster (assoc) is going to be. Whereas, mean-shift can automatically generate however many clusters using just a window size, which is used to find a center of a cluster. This type of behavior is more ideal when dealing with the continuous space; therefore, mean-shift is the better choice of the three here.

2. Since k-means determine the distance of a point to the cluster center using the Euclidean distance, the resulting clusters will often look like spherical clusters. In this case, the resulting clusters will look like 2 half circle.

3. Professor said in OH that blob's information (area and location) is known. But if it is not, we can detect the blobs by looping through the image and easily find it since it is a binary image (0 vs 1). We can also get a rough estimate the area of the blob by multiplying its width and height (obtained by looping through pixel near the blob and find max/min and do max - min), but it is not accurate. Alternatively, we can loop through the image and count the number of pixels a blob is consists of. Either way, we will just assume that we know the area of each blob and have it in an array, called blob_area.

```
Function group_blob (blob_area[], num_group) {
    sort the array blob_area

    while (true) {     // k-mean
            randomly pick num_group cluster center location

            each blob compares its area with each cluster center's area and see
            which cluster center it is closer to (and become part of that cluster)

            each cluster center finds the centroid and set that to be its center

            terminate if all of the cluster centers did not move
    }
}
```

II. Programming:

a. Getting correspondences:

Usage:

```
>> [pt1, pt2] = get_correspondences(im1, im2);
```

*Select 4+ points for im1, then press <Enter>.

*Select 4+ points for im2, then press <Enter>. Correspondences for im1 is plotted on im1 in case user needs it for reference.

Note:

- The correspondences should match, meaning the im1's 1st correspondences should be the same targeting object as im2's 1st correspondences.

b. Computing the homography parameters:

Usage:

```
>> H = computeH(pt1, pt2);
```

c. Warping between image planes:

Usage:

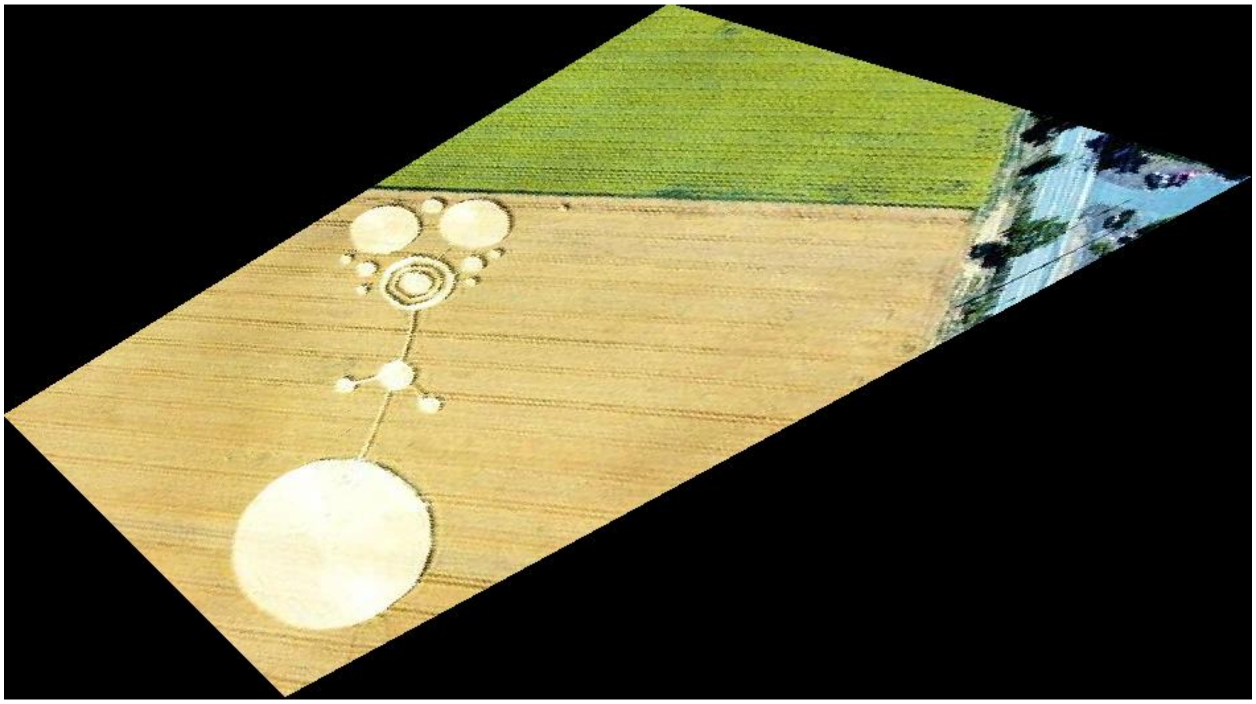
```
>> [warp_result, merge_result] = warpImage(im1, im2, H);
```

Note:

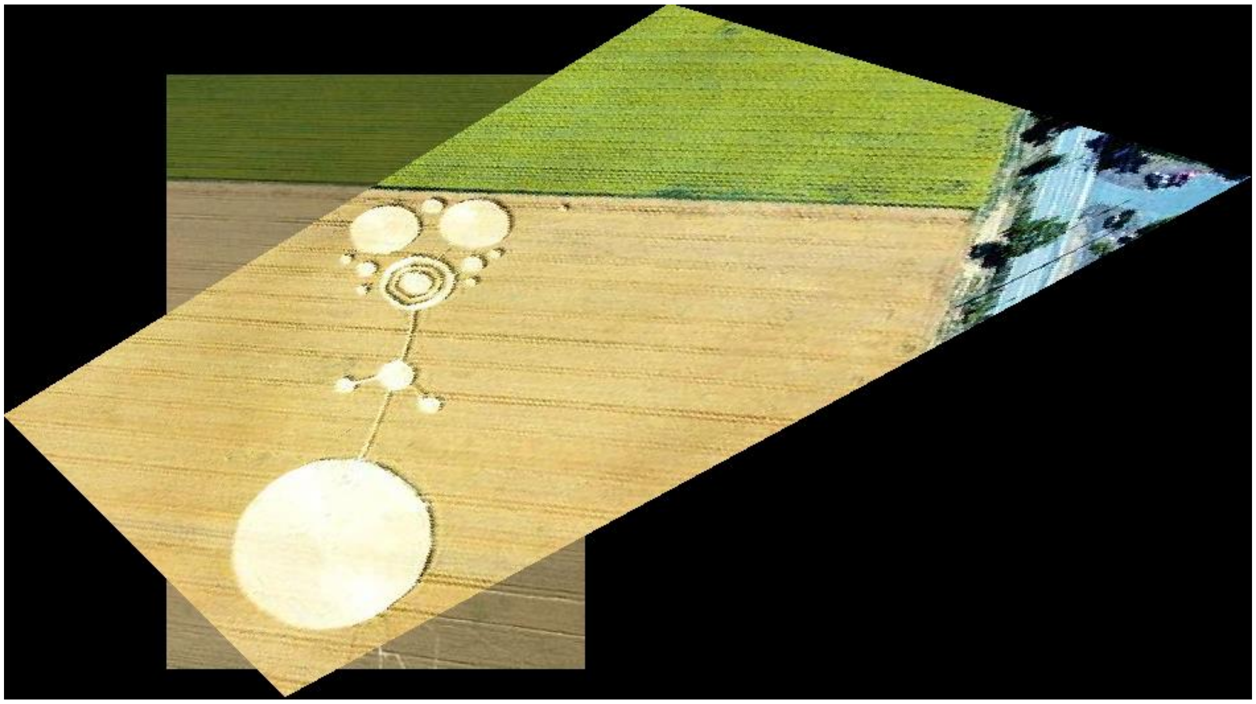
- H should be in the same order, pt1 should be correspondence of im1 and pt2 should be correspondence of im2.Our warp_result is the same size as merge_result. Therefore, there is often black pixel paddings around the warp_result.

Pair 1:

Warp:

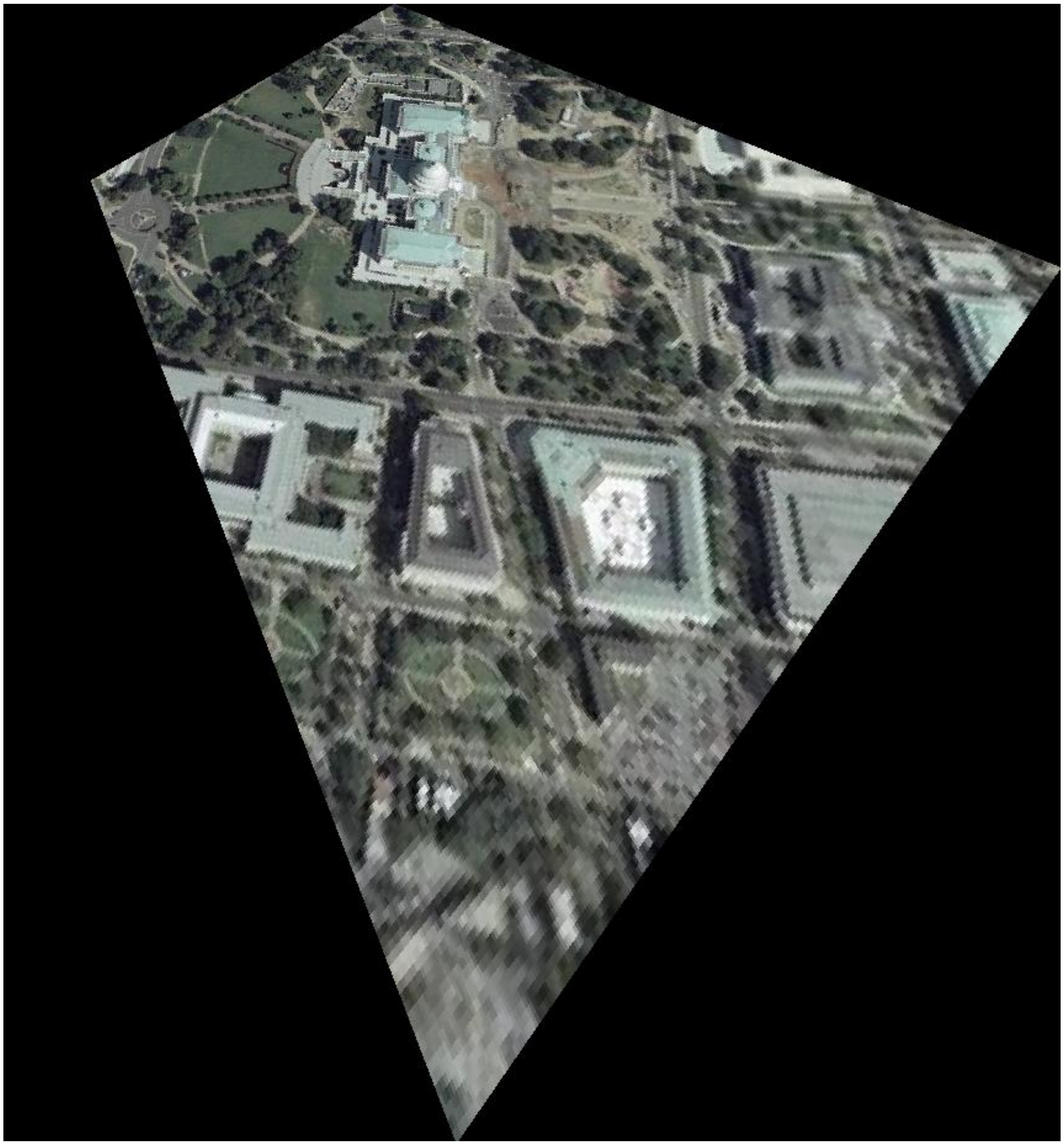


Merge:

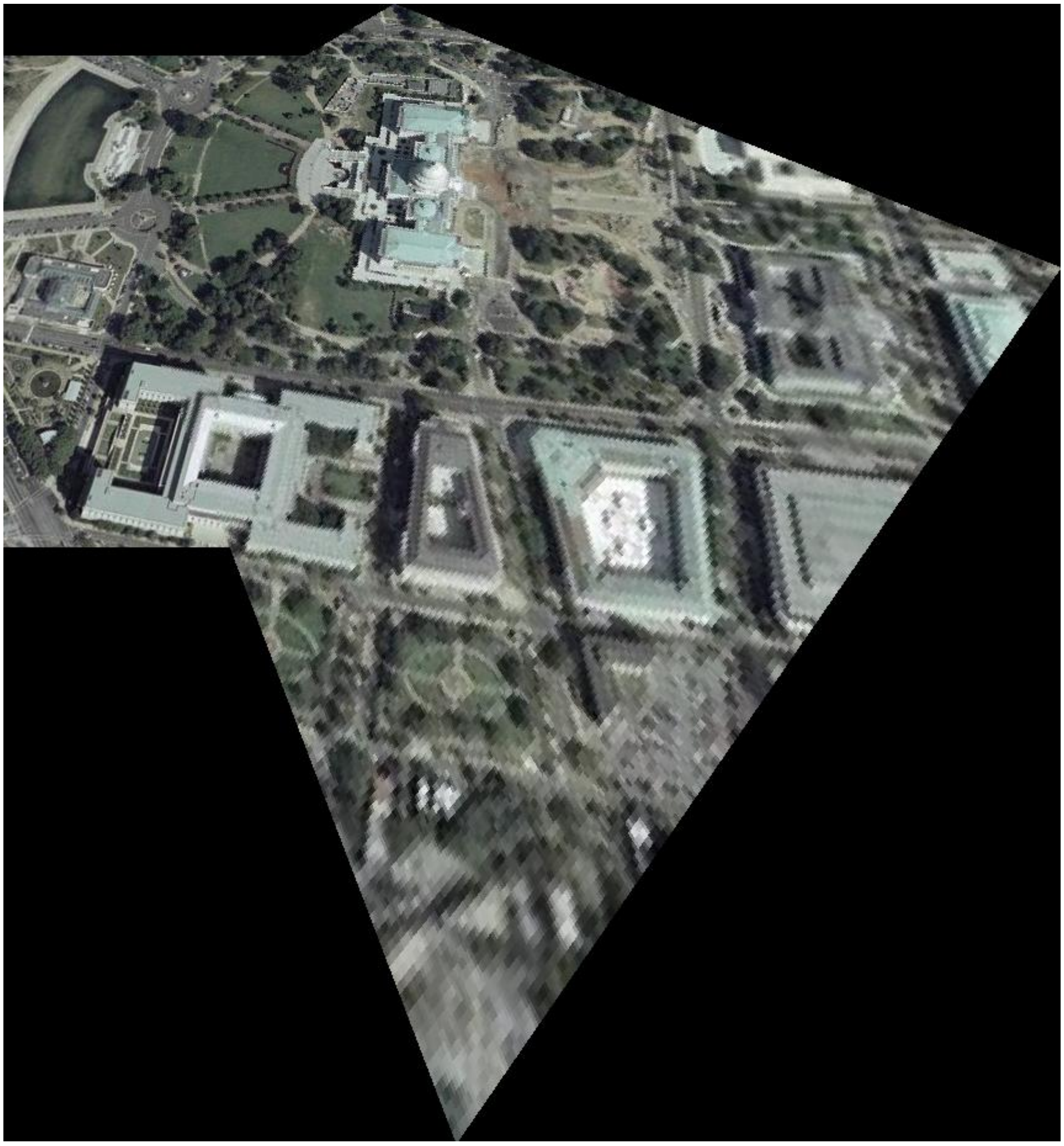


Pair 2

Warp:

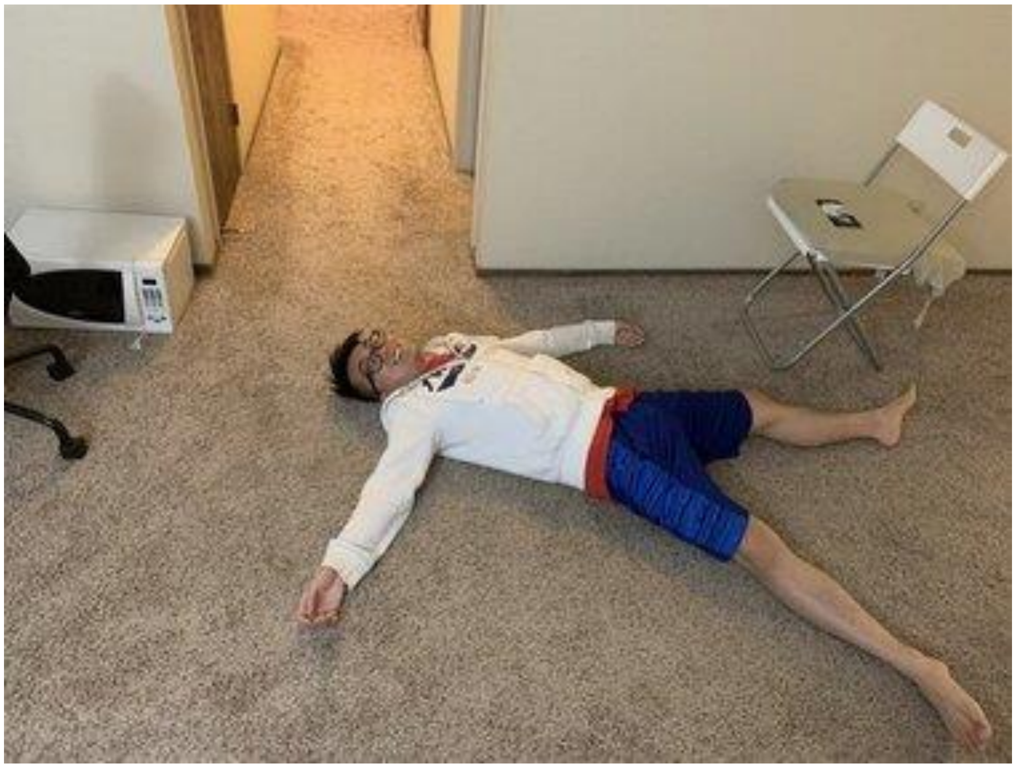


Merge:

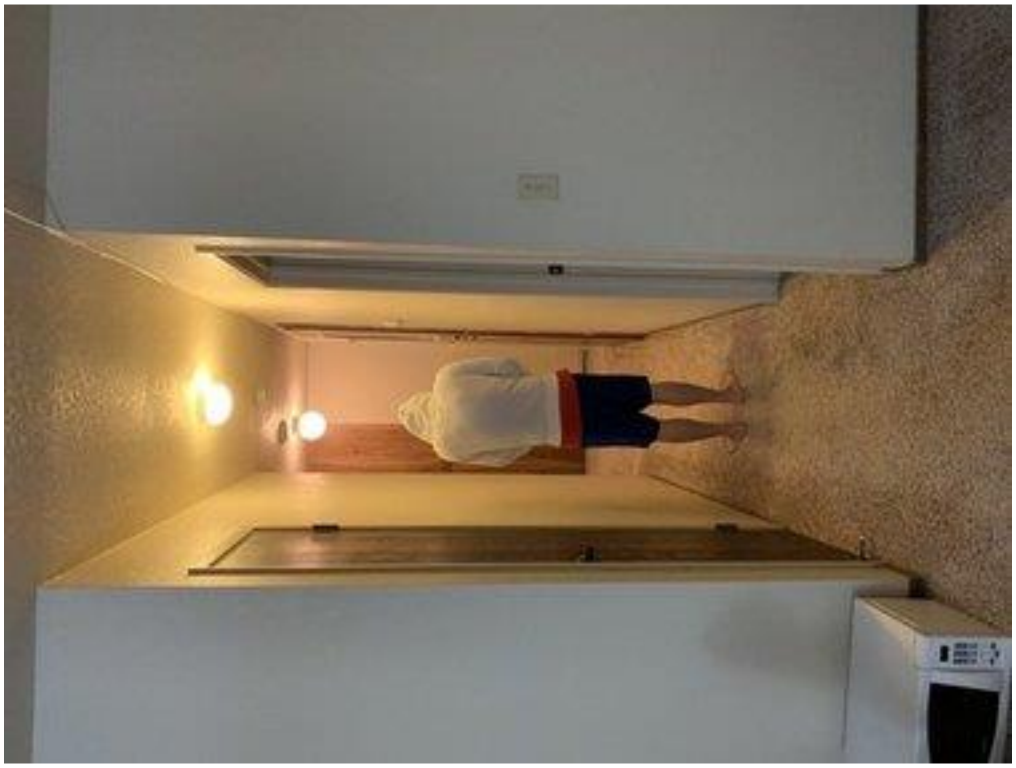


Additional Example:

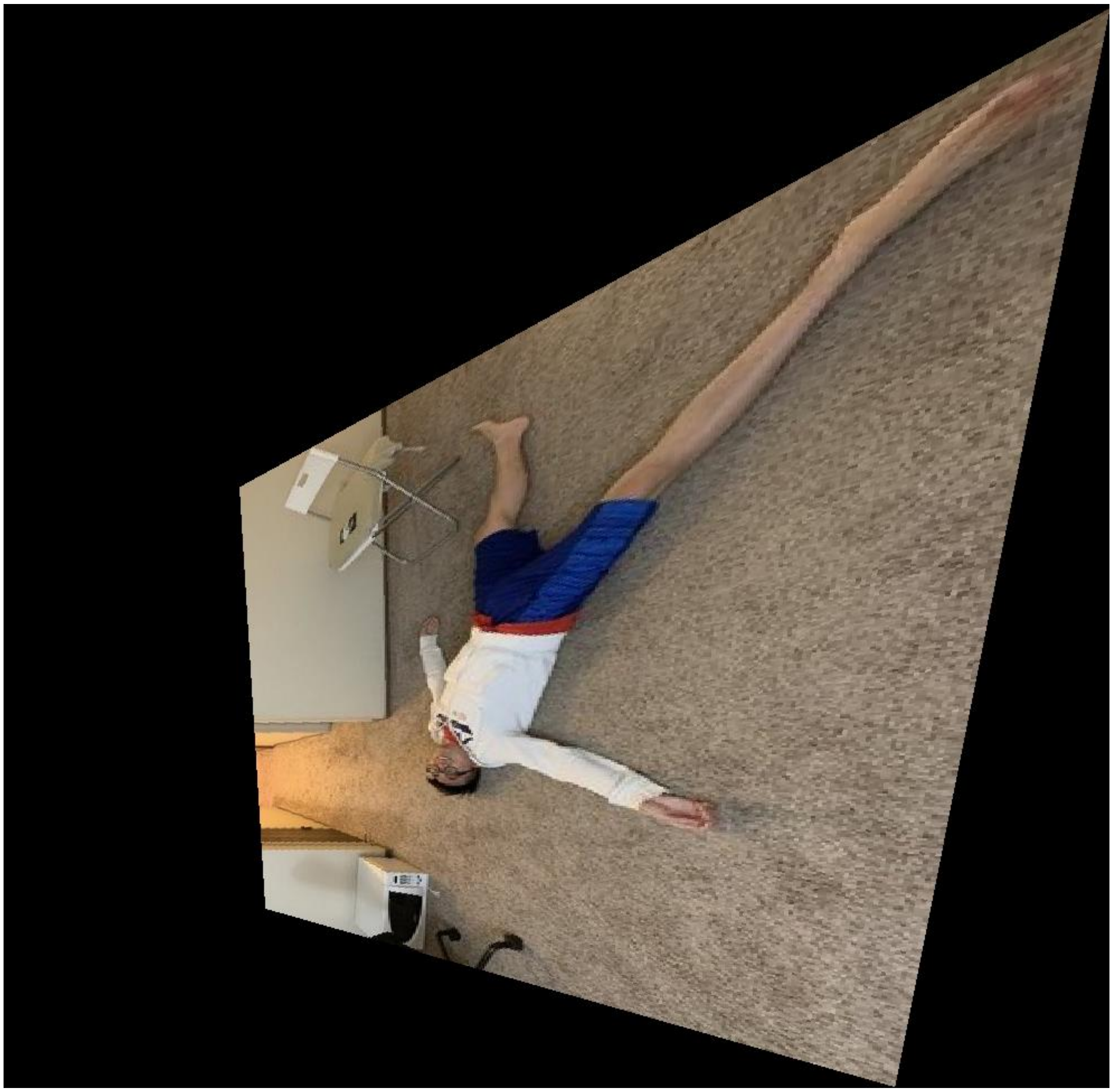
Input image:



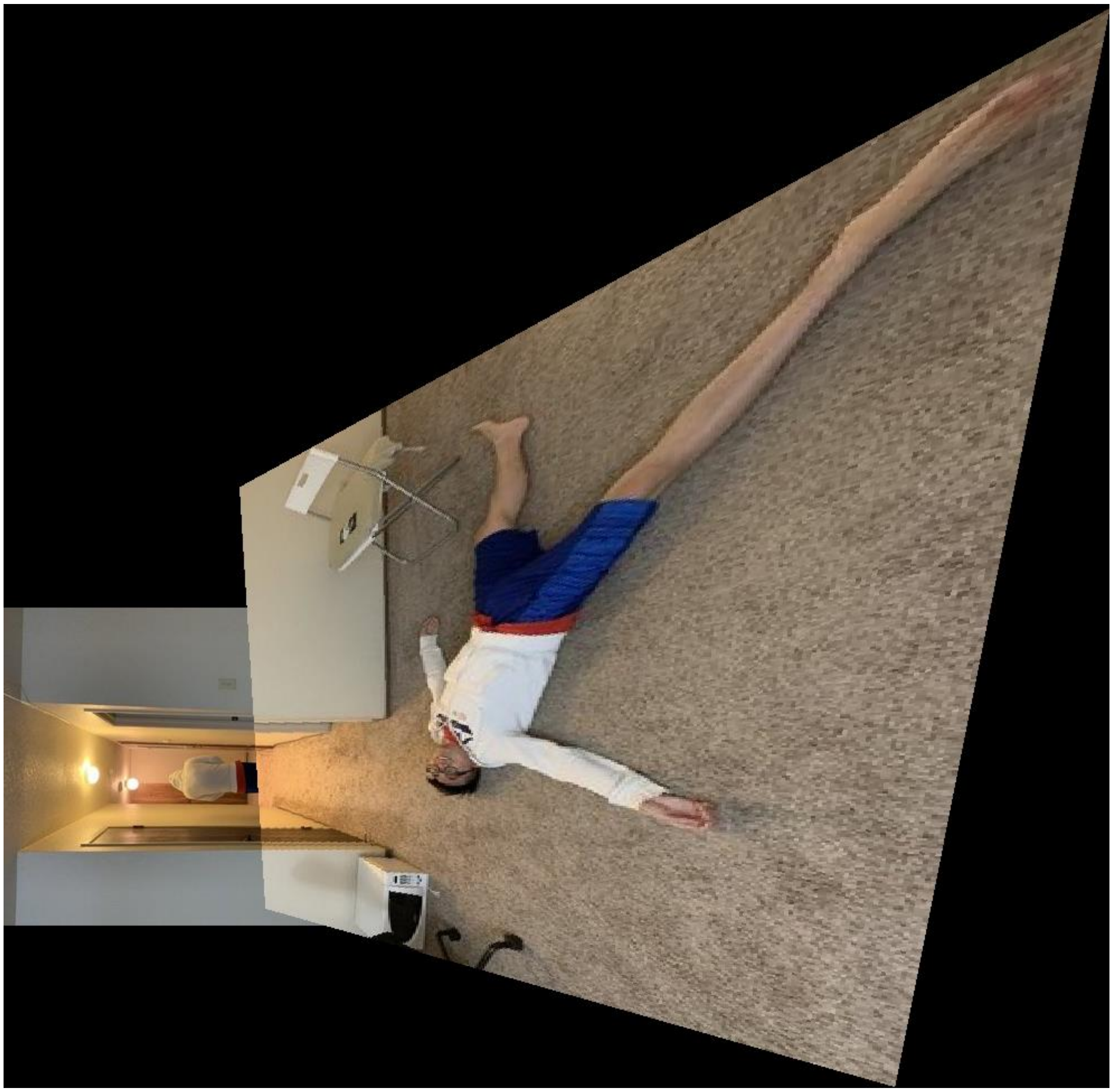
Reference image:



Warp:

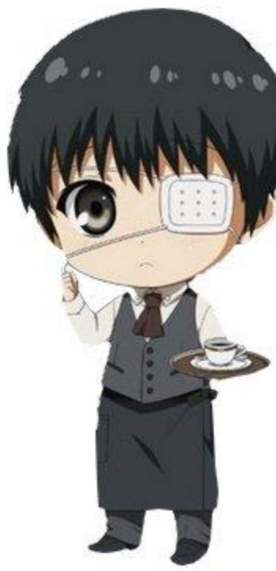


Merge:

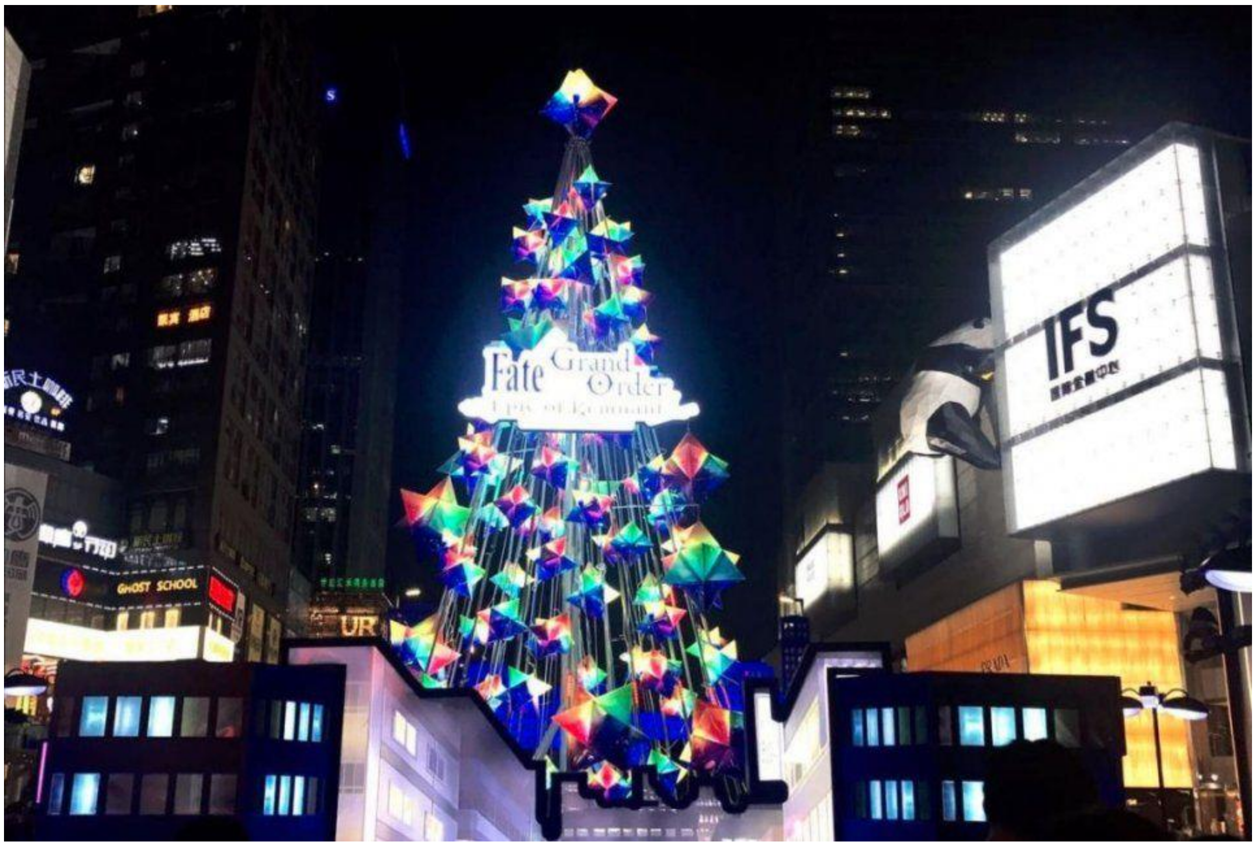


Frame:

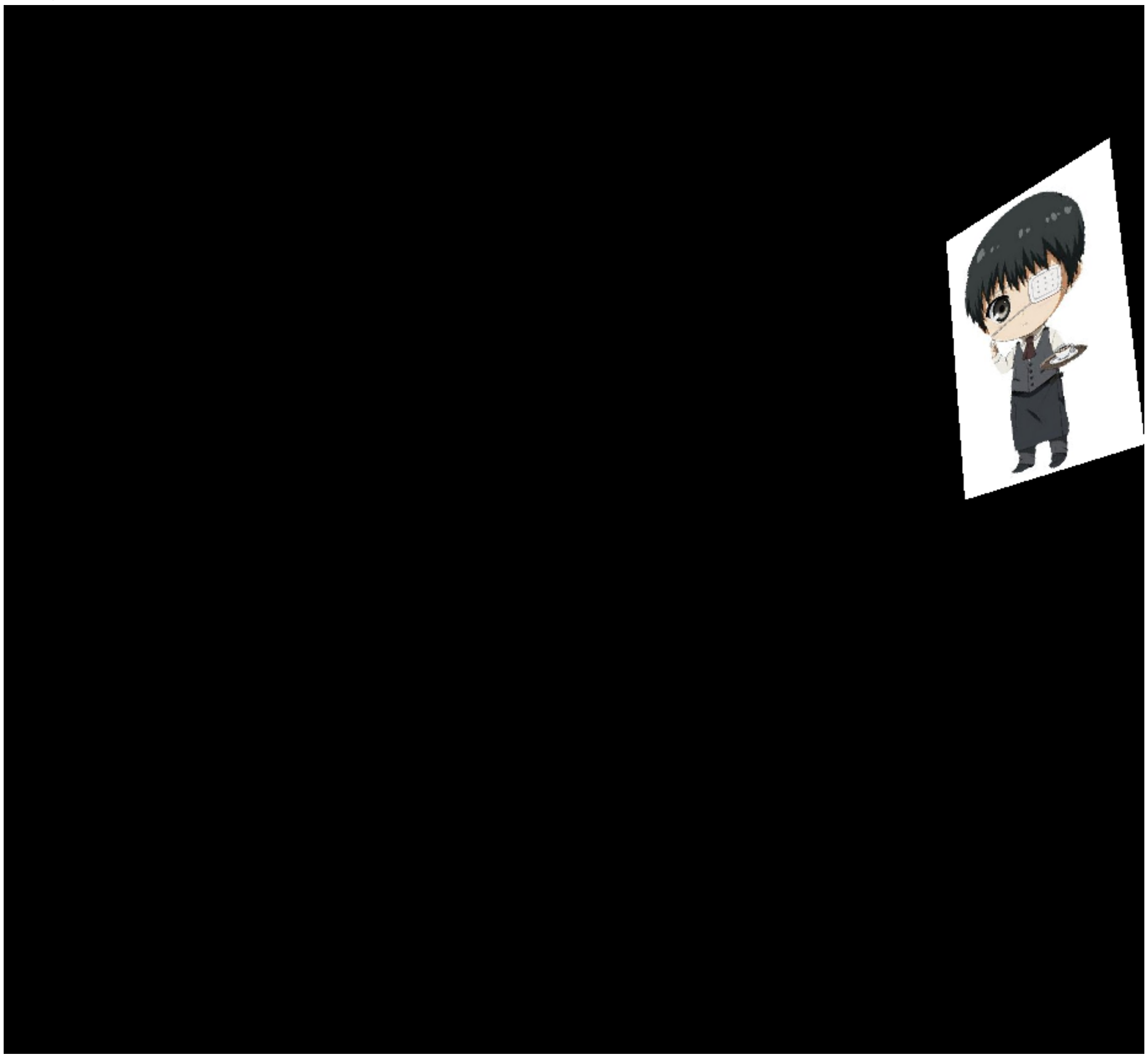
Input image:



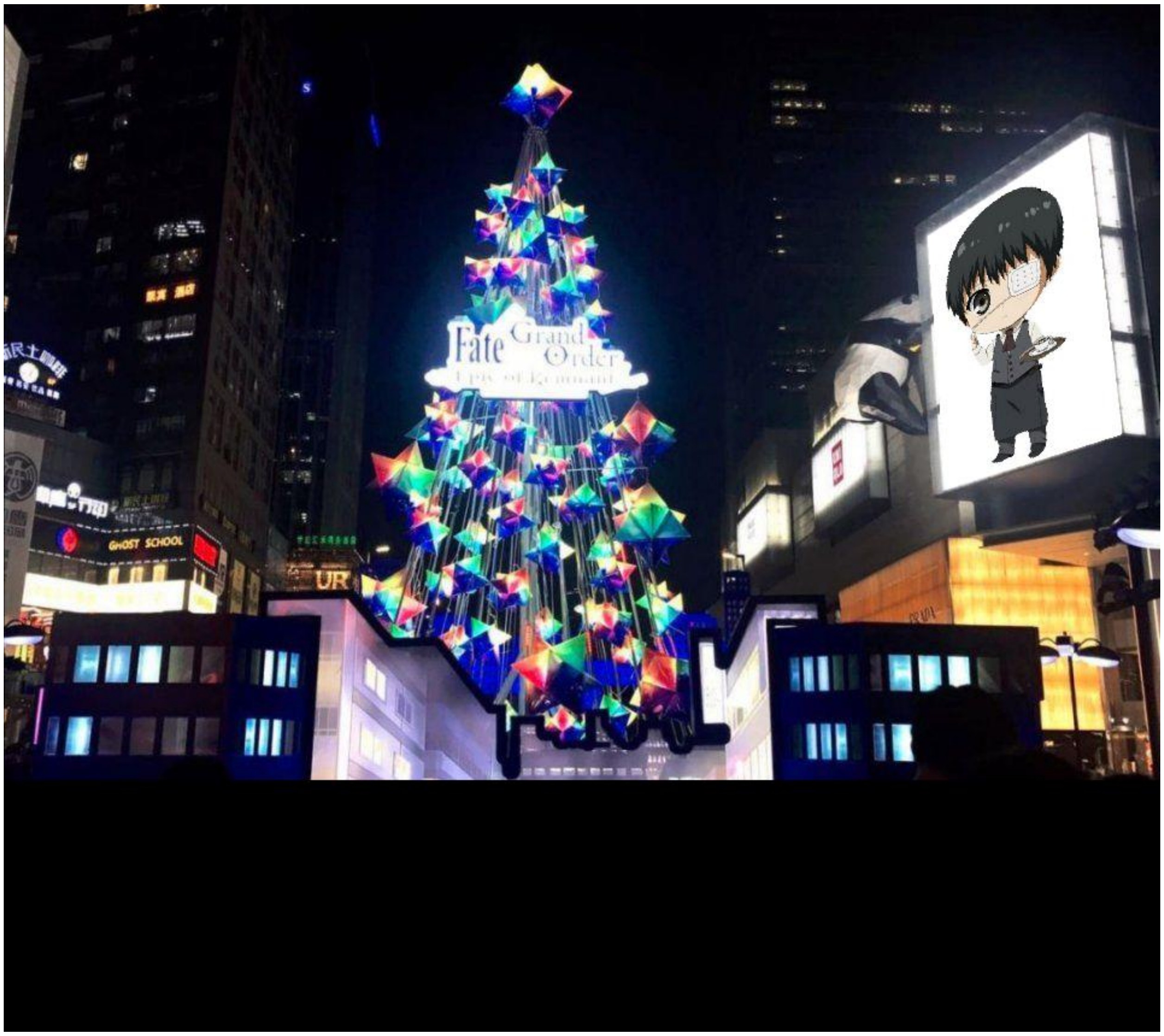
Reference image:



Warp:

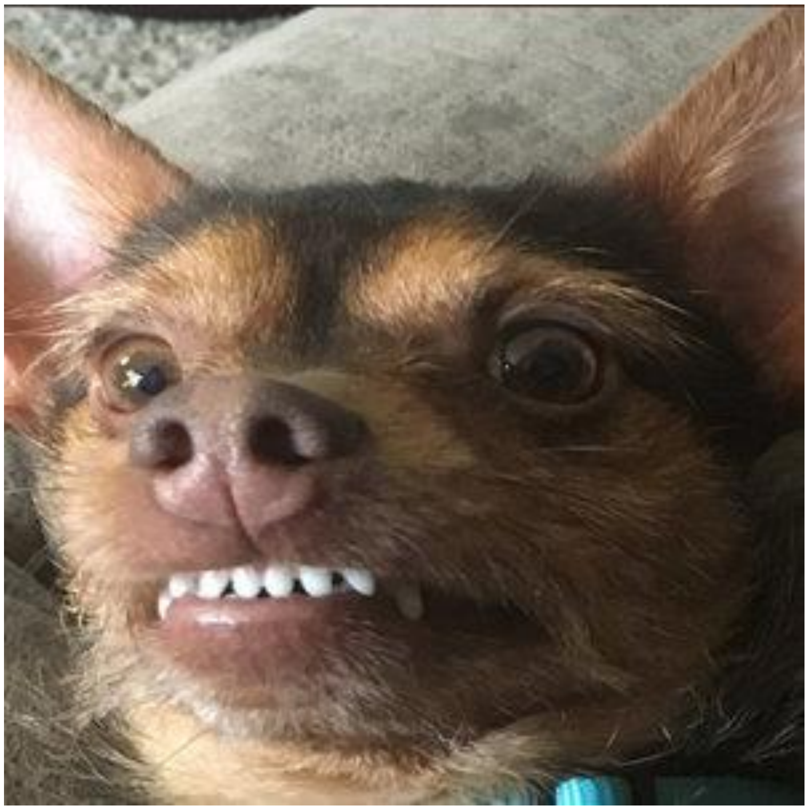


Merge:

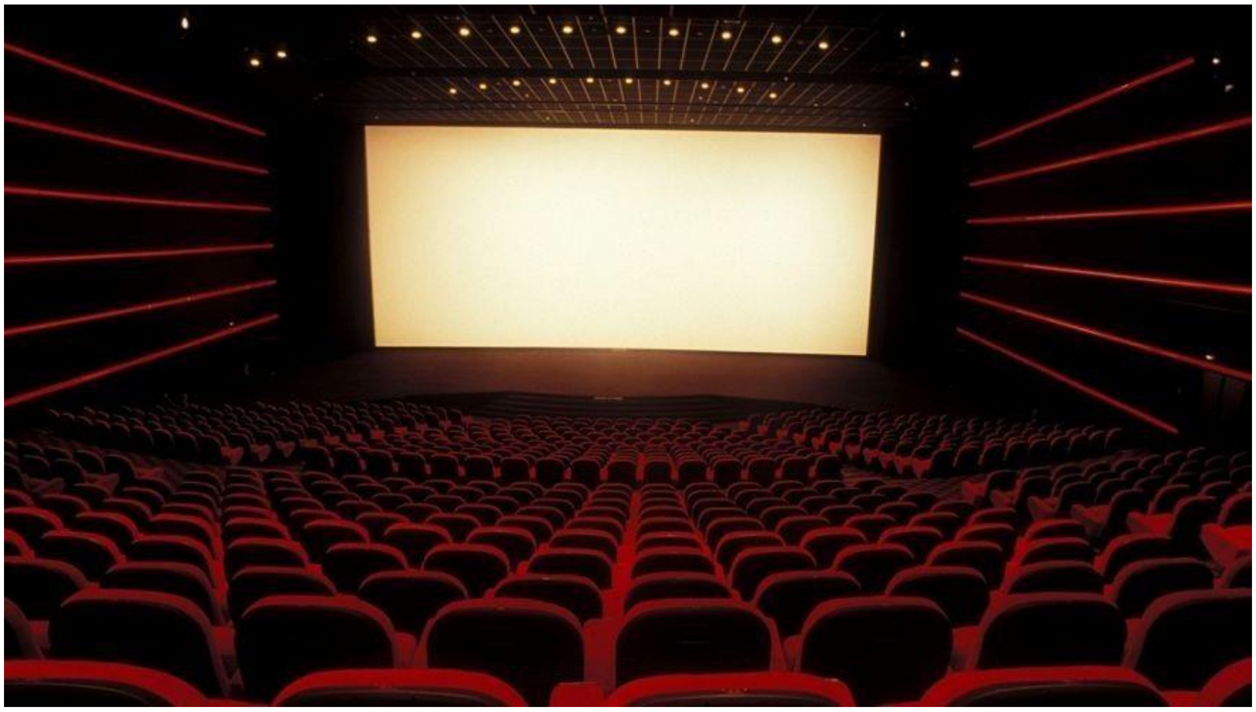


Frame 2nd example:

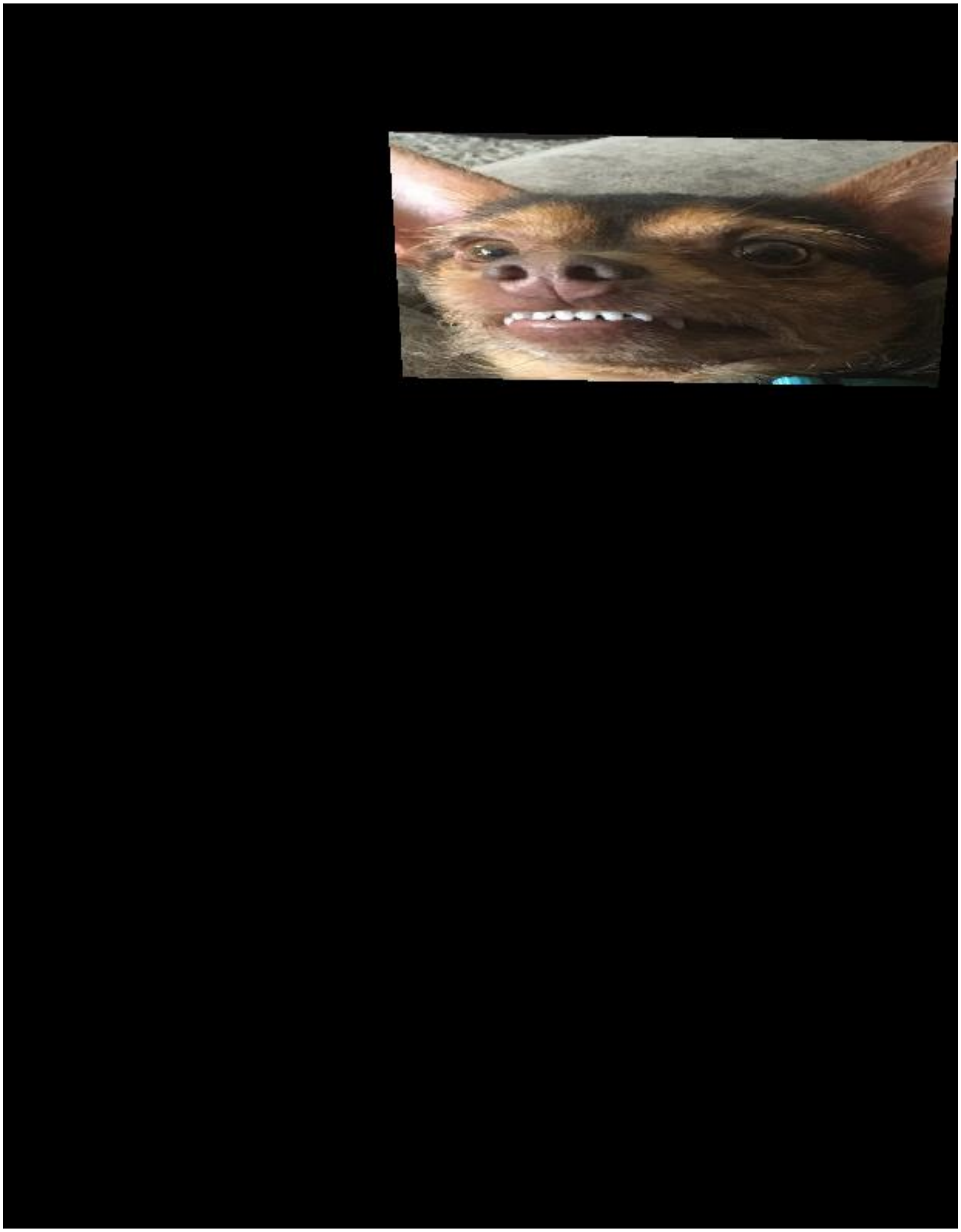
Input image:



Reference image:



Warp:



Merge: