CS 435 - Computational Photography

Final Project

Ameer Jalil

Contents

1 Hard Coding Correspondences

Let's start off by hard coding some point correspondences. Look at each image and choose four point correspondences.
Display the images side-by-side (as one image) with the point correspondences color coded as dots in the image.

Figure 1: Manual Correspondences

2 Compute Transformation Matrix, Project, and Blend!

Next, use the four points you identified in the previous part to compute the transformation matrix that maps one image to the other. You can determine which image you want to be the "base" image.

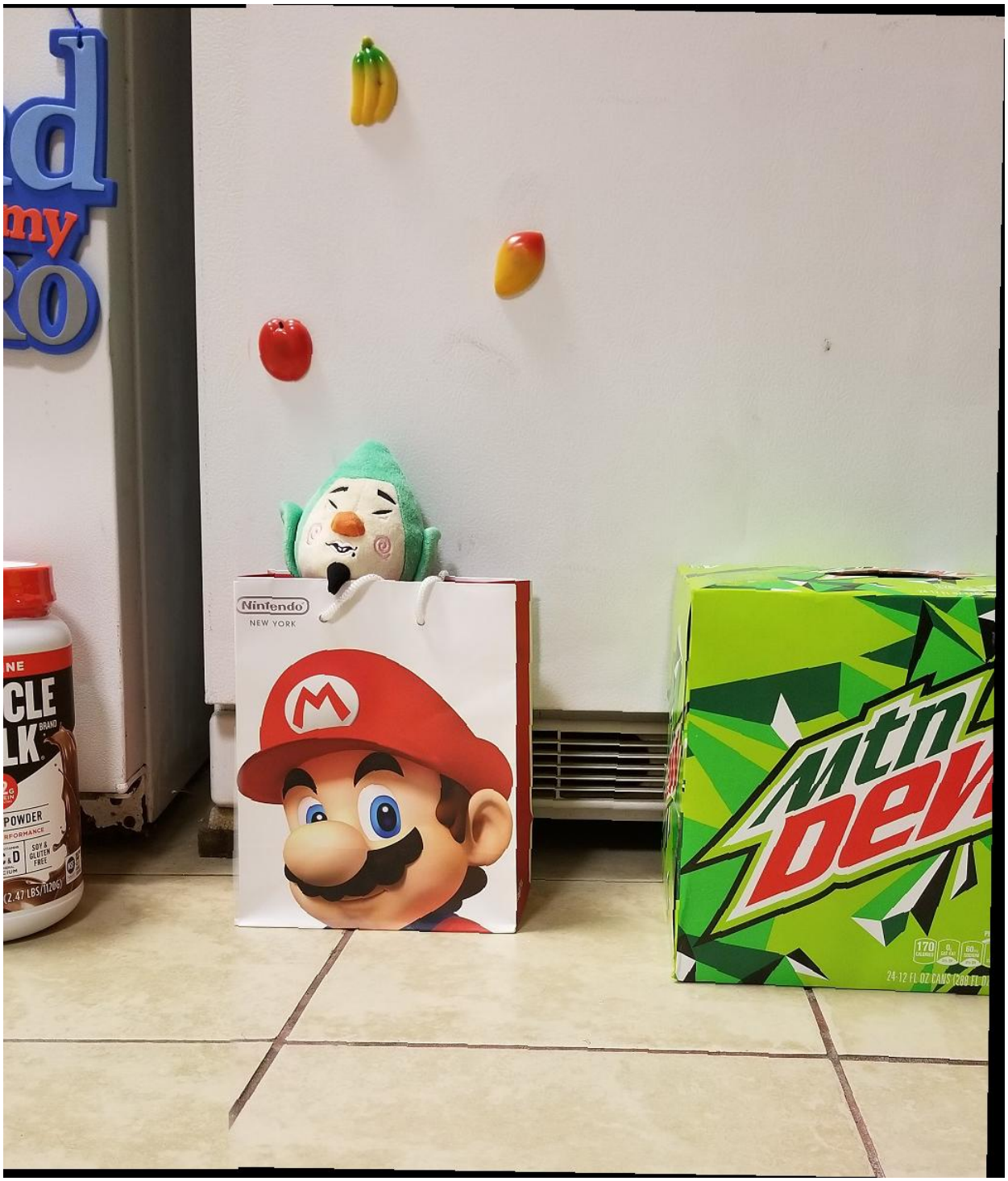

Figure 2: Stitched images using manual correspondences

3 Create Scale-Space Image Pyramids

Now on to the tough(er) stuff! We want to automate all this!
The first step is to automatically identify locations of interest. To do this we'll find the stable local maximas in scale-space for each image. And the first step of that is to create image pyramids!
Here are some hyperparameters we'll use to create our image pyramids:

- Find the extremas in *grayscale*.
- Create five scales per octave.
- The initial scale will have a standard deviation of $\sigma_0 = 1.6$.
- Each subsequent scale will have a value that is $k = \sqrt{2}$ times larger than the previous.
- Each Gaussian kernel will have a width and height that is three times the filter's σ value, i.e. $w = \lceil 3\sigma \rceil$.
- Create four octaves, each 1/4 of the size of the previous octave, obtained by subsampling every other row and column of the previous column (no interpolation).

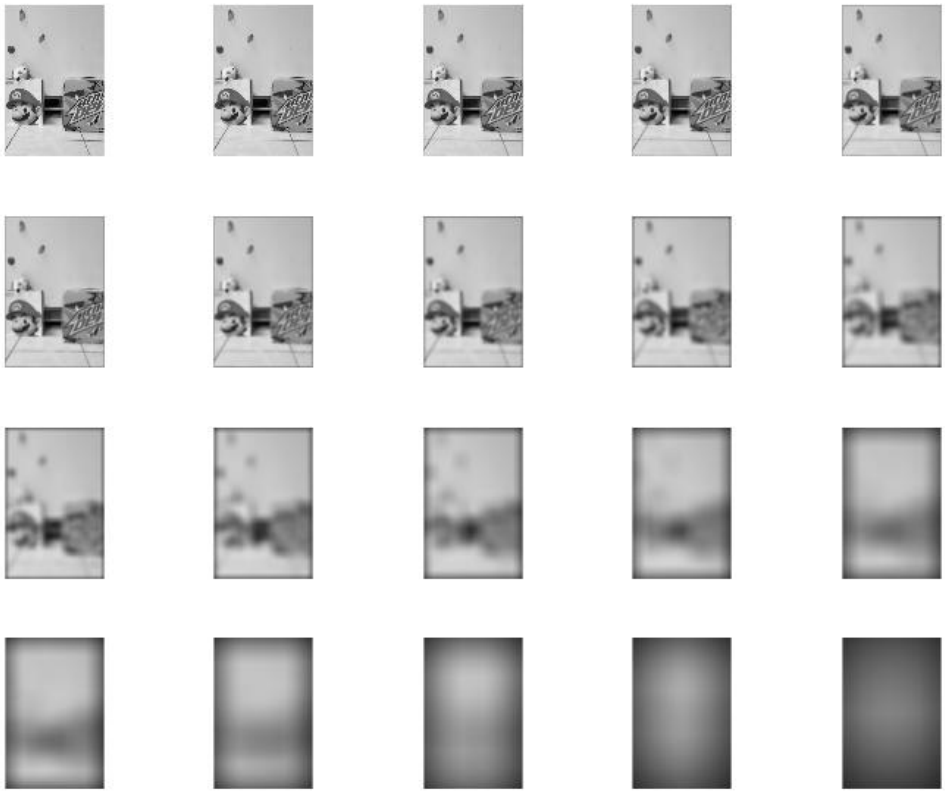

Figure 3: Right Image Pyramid

4 Finding the Local Extremas

Now for each octave of each image, locate the local maxima, as discussed in class. These locations then need to be in terms of the original image's size (i.e. the first octave), which can be done by multiplying their locations by 2^{n-1}, where again n is the current octave.

After identifying all the extrams, we want to remove the *unstable* ones, i.e. those that are edge pixels and/or in areas of low contrast. To do this:

- Find edge pixels use Matlab's *edge* function. Use that to eliminate extremas that are also edge pixels.
- We will also eliminate extremas that are too close to the border of the image.
- Finally, for each remaining extrema, compute the standard deviation of a patch around it. If this standard deviation is less than some threshold, then the patch has low contrast and thus should be eliminated from the extrema list.

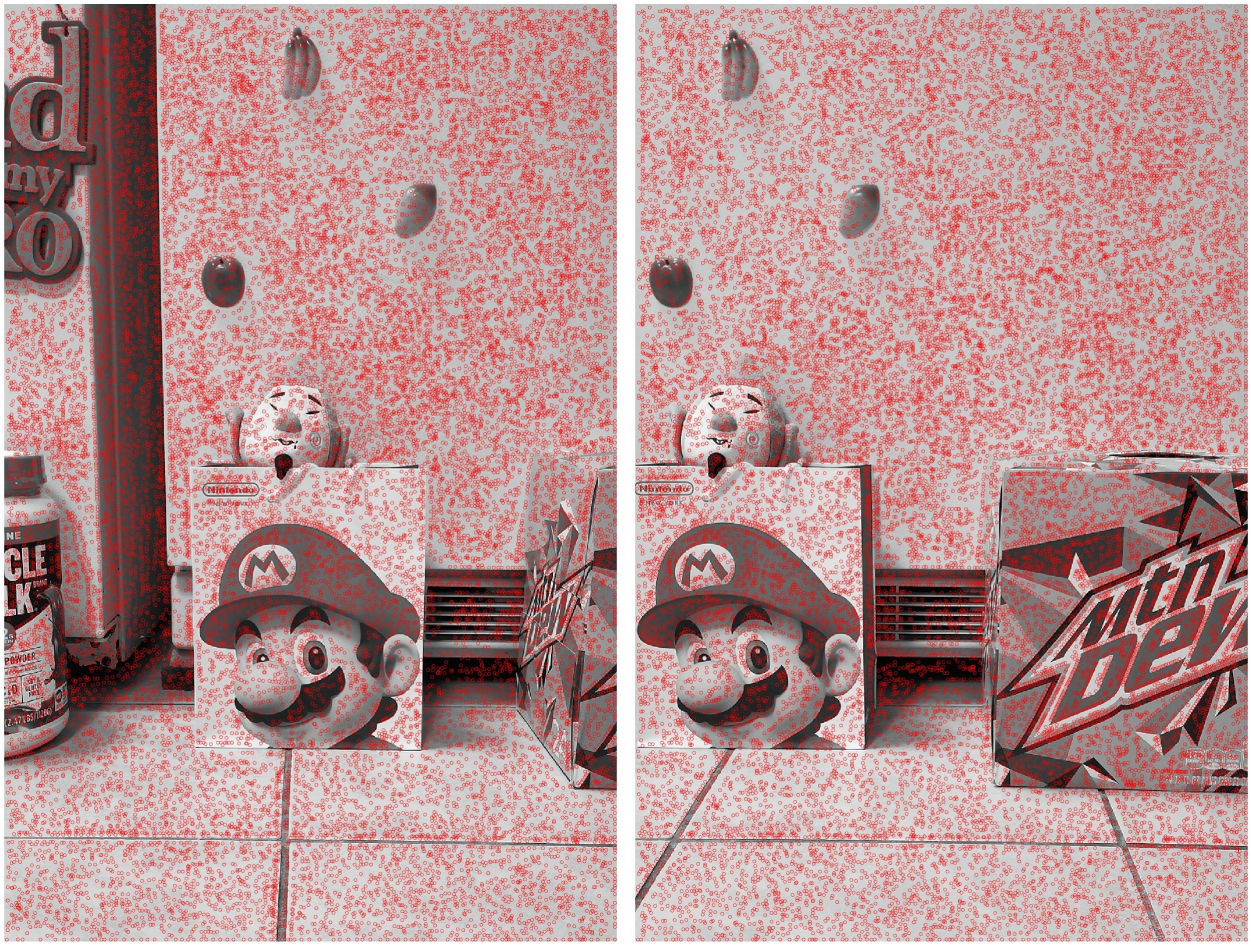

Figure 4: All extrema points

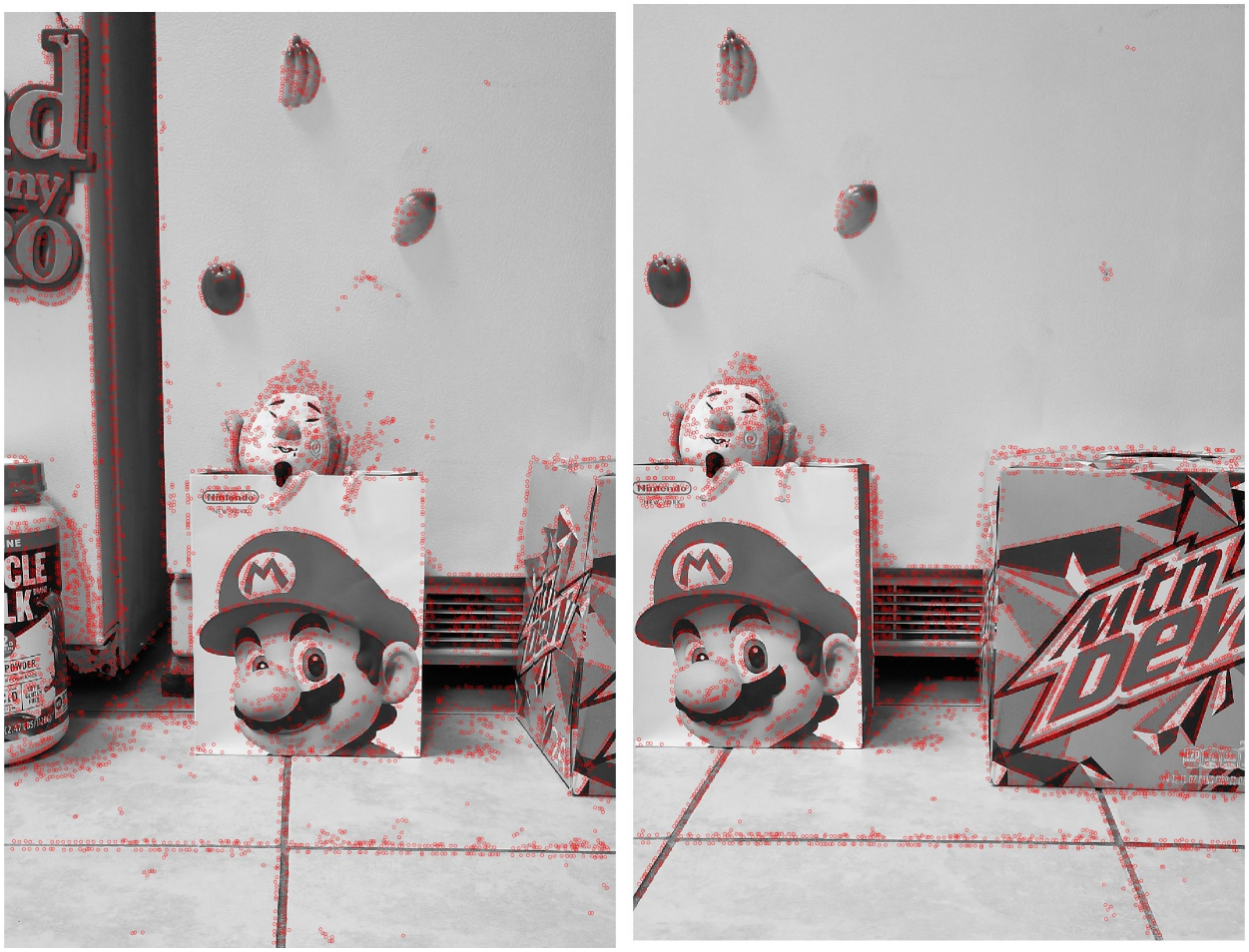

Figure 5: Pruned extrema points

5 Keypoint Description and Matching

For each remaining extrema/keypoint in each image, we'll want to extract a descriptor and then match the descriptors from one image to ones in the other.

5.1 Descriptors

For each keypoint in each image, The chosen descriptor was a histogram of a 9x9 patch around the keypoint in all 3 color channels.

5.2 Keypoint Correspondences

To match keypoints in both images, two assumptions were made

- Correspondences should have roughly the same y value.
- The camera was rotated and/or translated right to obtain the second image.

The keypoint matching strategy was as follows

1. For each keypoint in the first image, find the best match (using histogram intersection) in the second image that satisfies the aforementioned constraints. Call this set C_1.
2. For each keypoint in the second image, find the best match (using histogram intersection) in the first image that satisfies the aforementioned constraints. Call this set C_2.
3. For both C_1 and C_2, remove any matches with a similarity below 90%
4. Compute the set intersection of the two sets: $C = C_1 \cap C_2$

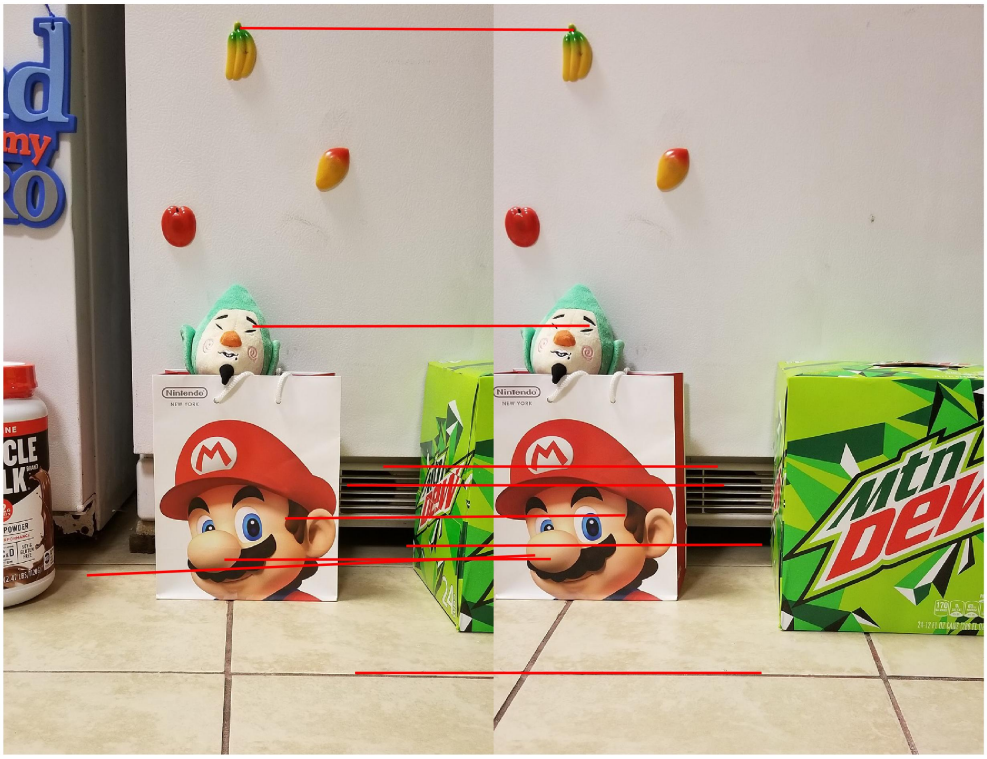

Figure 6: Some point correspondences

6 Find the Transformation Matrix via RANSAC and Stitch

Finally we want to use the keypoint correspondences to compute a transformation matrix that we can then use to auto-stitch our images.
However, as you may have noticed, many of the point correspondences might not be correct :(. So instead we'll use a *RANdom SAmpling Consensus* (RANSAC) strategy.
To Perform RANSAC for our panoramic stitching:

1. For experiments 1 through N (chosen as a function of the number of correspondences found)
   (a) Select four correspondences at random.
   (b) Compute the transformation matrix using these correspondences.
   (c) Using the discovered transformation matrix, count how many point correspondences (among all of them) would end up within a few pixels of one another after projection.
2. Keep the transformation matrix the resulting in the largest number of point correspondences (among all of them) that ended up within a few pixels of one another after projection.

Now use this transformation matrix to stitch the images!

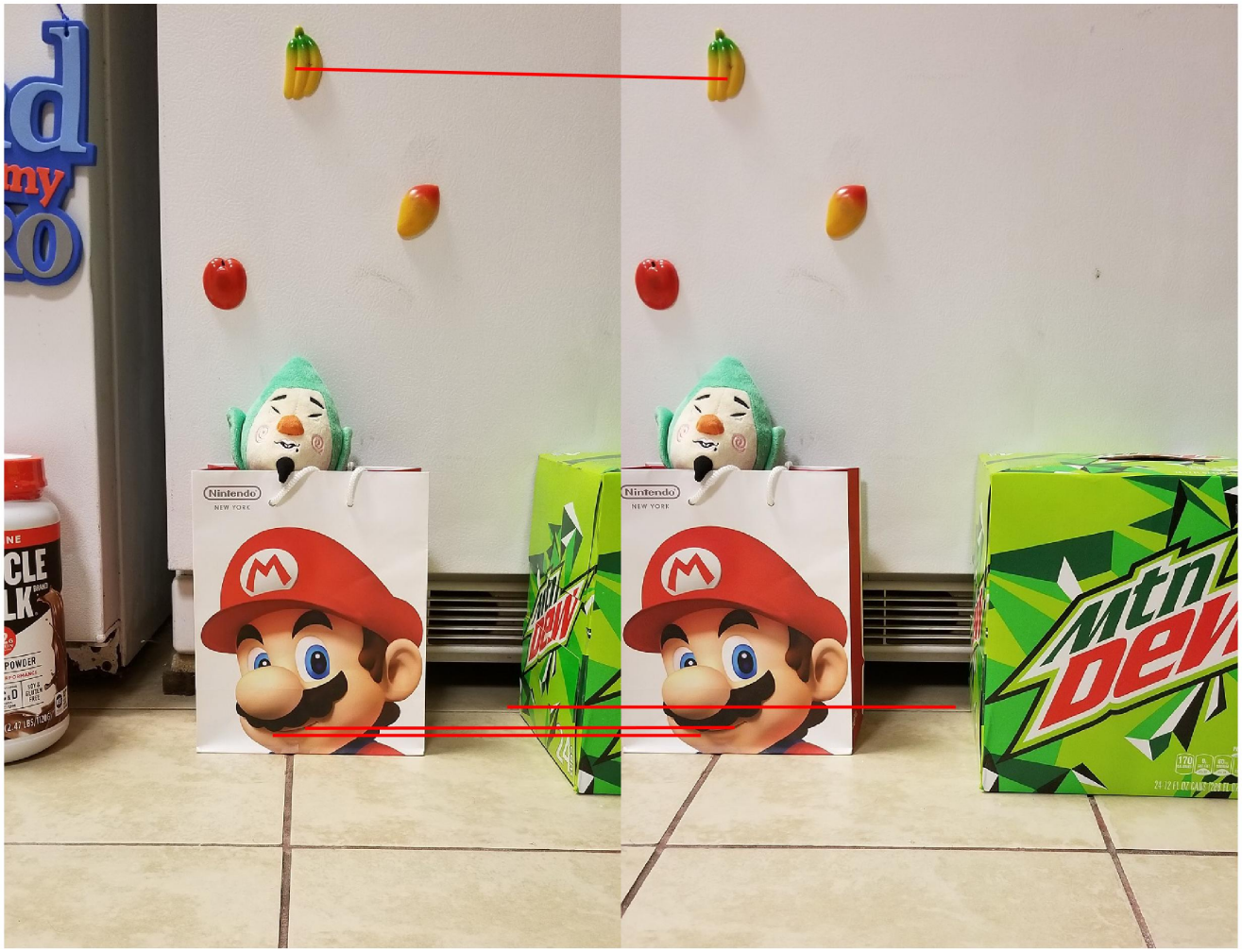

Figure 7: Chosen point correspondences for final transformation matrix

6.1 Final Image

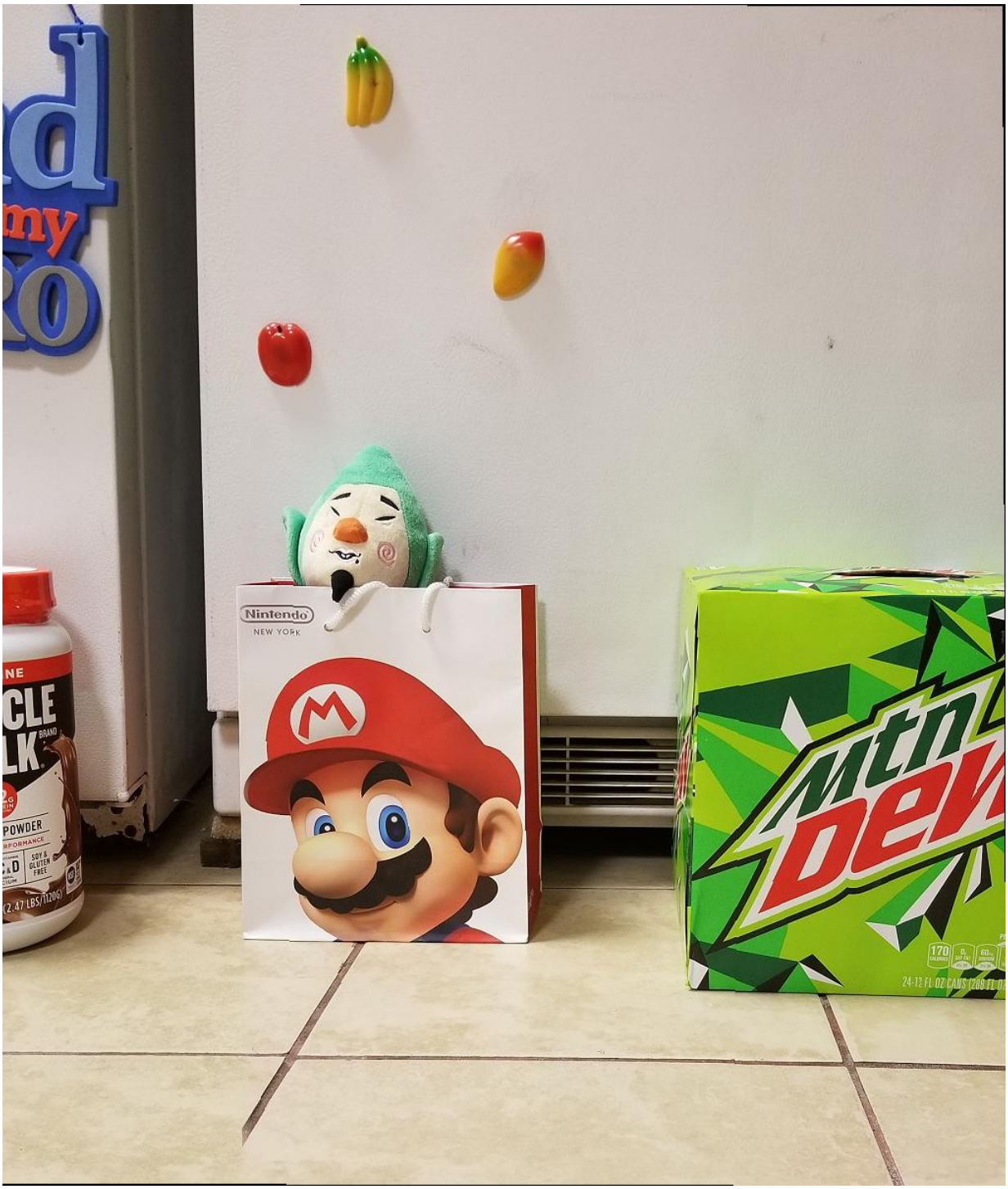

Figure 8: Final stitched image

7 5-Image Automatic Stitching

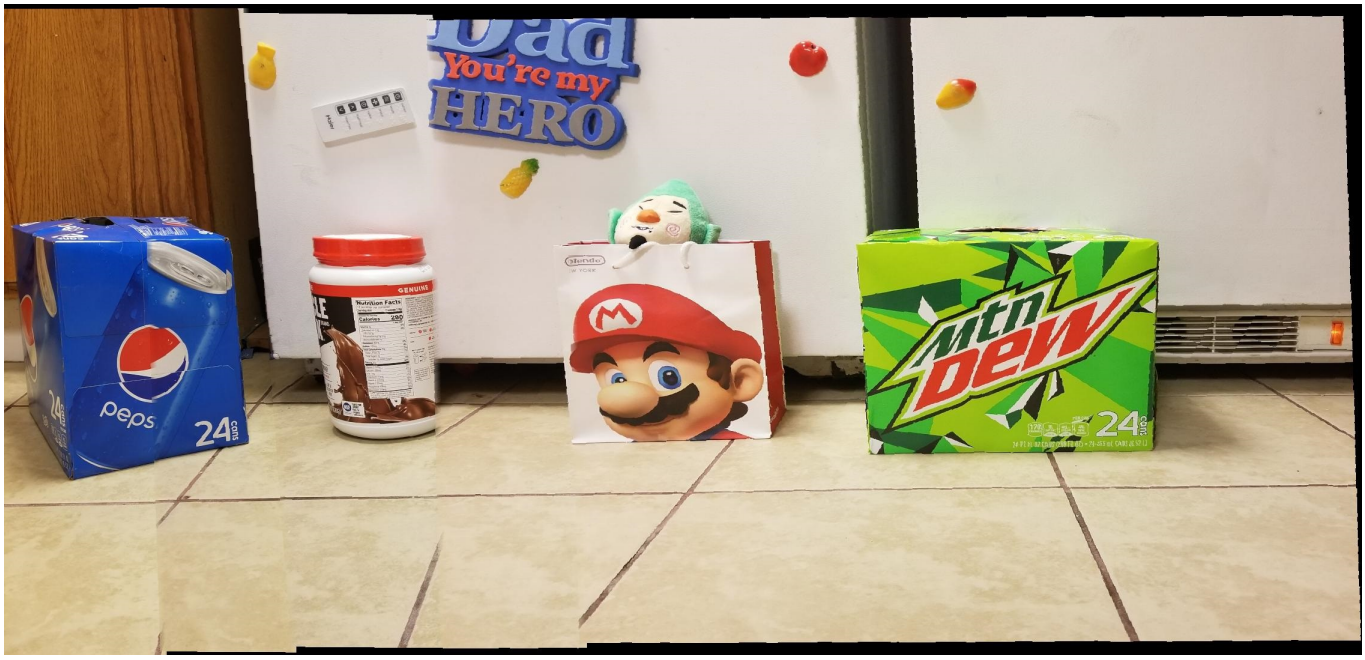

Figure 9: 5-Image Stitch

I attempted to pursue stitching 5 images to make a wide panoramic image using my automatic stitching algorithm. Being able to take 5 images with good overlap between adjacent images, while minimizing any slight variance in the images' rotation/elevation to get a good final result proved to be rather challenging!

Also, because of the nature of perspective images, the subjects in the image lose some of their intrinsic features: parallel lines no longer remain parallel, and the panning of the camera makes it seem as if the items are rotated. These factors also contributed to the imperfections found in the final image

I also wanted to attempt to maintain image 3 as the center and base image, and have the other images map onto it. It was non-trivial and non-intuitive to get this functionality without a rather large refactoring of my code. That could have resulted in a cleaner looking final image.