Comp Photography (Spring 2016)
HW 8

Stewart Boyd

Images for assignment 8

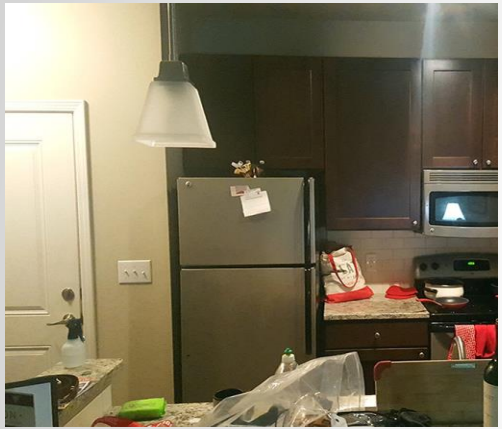


Image 1

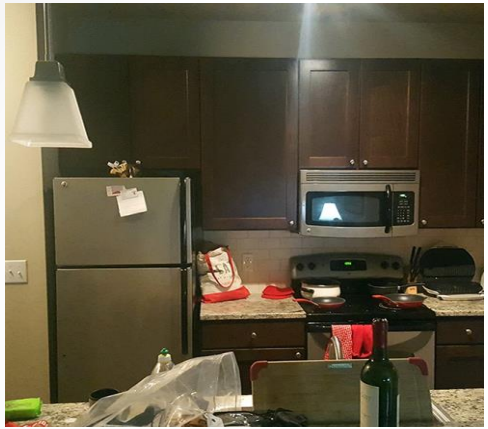
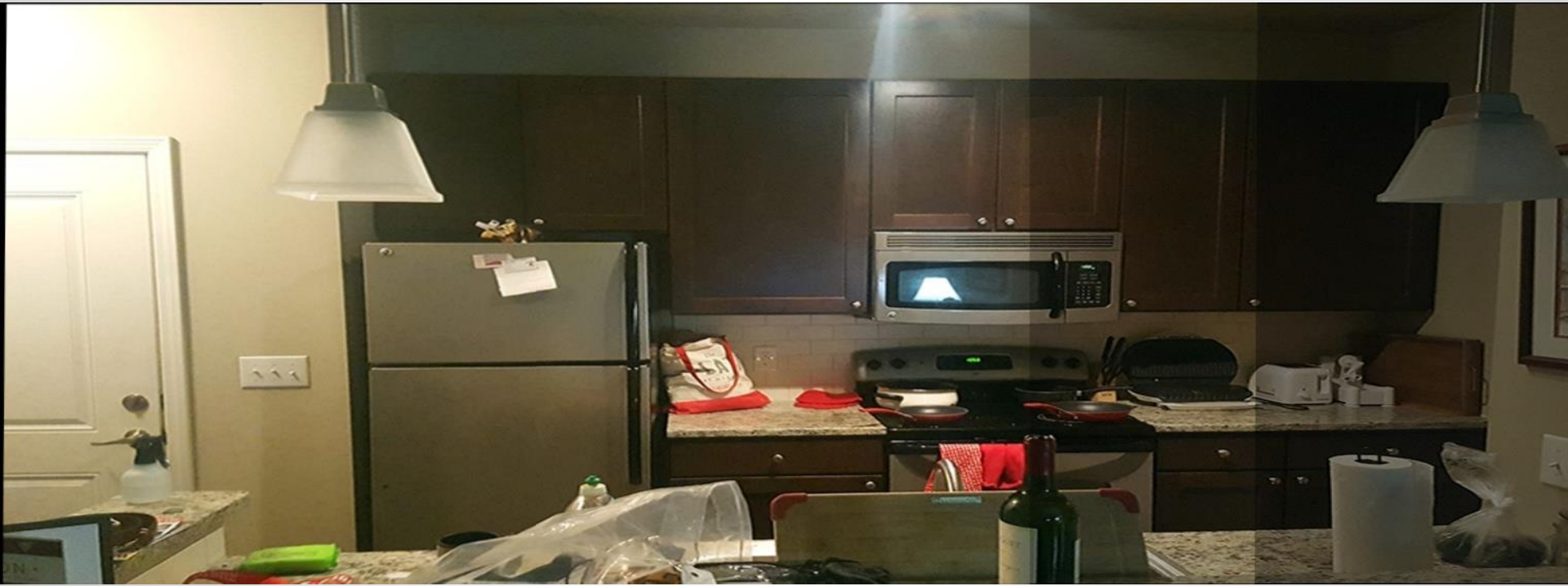


Image 2

Image 3

Panaroma Result



blendImagePair

My implementation of blendImagePair was simply an average of the warped_image and image_2 at the point given by argument point.

Because of overflow issues with int8 I cast the Ndarray's to int64s before the average operation.

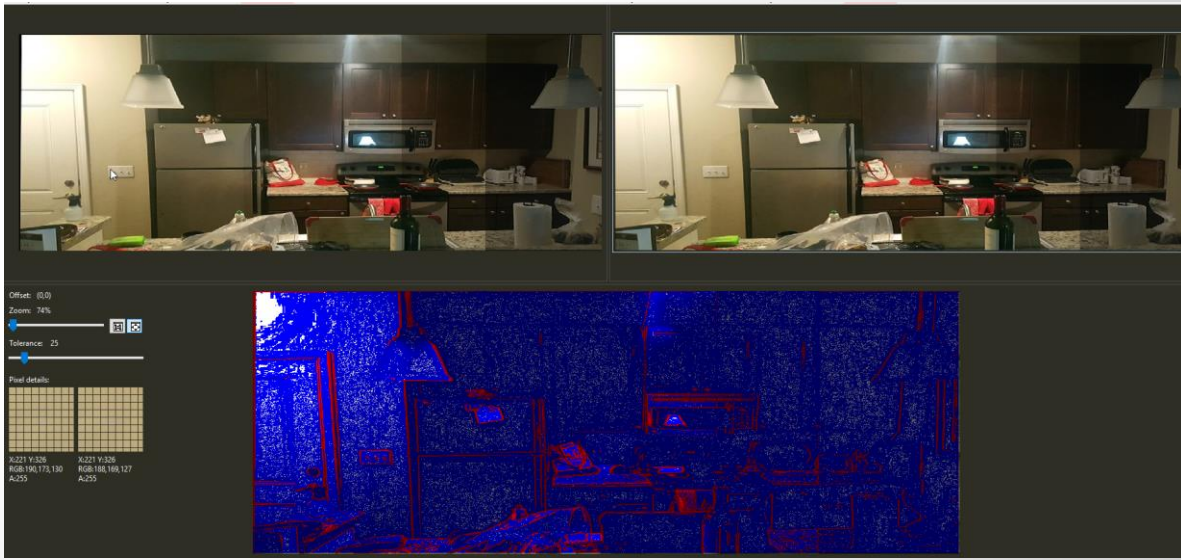
The code may be seen in the adjacent image

```
def blendImagePair(warped_image, image_2, point):
    """ This is the blending function. We provide a basic implementation of
    this function that we would like you to replace.

    This function takes in an image that has been warped and an image that needs
    to be inserted into the warped image. Lastly, it takes in a point where the
    new image will be inserted.

    The current method we provide is very simple, it pastes in the image at the
    point. We want you to replace this and blend between the images.

    We want you to be creative. The most common implementation would be to take
    the average between image 1 and image 2 only for the pixels that overlap.
    That is just a starting point / suggestion but you are encouraged to use
    other approaches.

    Args:
        warped_image (numpy.ndarray): The image provided by cv2.warpPerspective.
        image_2 (numpy.ndarray): The image to insert into the warped image.
        point (numpy.ndarray): The point (x, y) to insert the image at.

    Returns:
        image: The warped image with image_2 blended into it.
    """
    output_image = np.copy(warped_image)
    # REPLACE THIS WITH YOUR BLENDING CODE.
    im2r = image_2.shape[0]
    im2c = image_2.shape[1]
    output_image[point[1]:point[1] + im2r, point[0]:point[0] + im2c] = ((output_image[point[1]:point[1] + im2r,
                                                                         point[0]:point[0] + im2c]).astype(np.int64) +
                                                                        image_2.astype(np.int64)) / 2
```

Sensitivity to Number of Matches

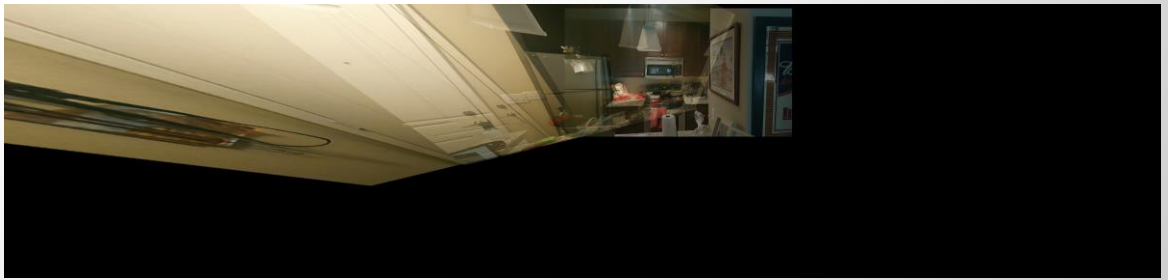
The images I ultimately used for the panorama were not sensititive to increasing the number of matching features used (increased from 10). The image on the left is the stitched together image using three features whilst that on the right is using 1000.

The blue and red image is the diffed result. Red indicates larger deltas between the intensity of a pixel in the left image versus the right. This image makes it clear that the result was a just a slight translational shift.



Sensitivity to Number of Matches (Cont)

However, that's not to say the overall algorithm is not sensitive to the number of matches required for stitching. I've included here, another outputted panaroma using a different set of pictures than that shown on slide 2. The top image is with 20 features requested, and the bottom is a 1000. The quality of matches diminishes as the request grows and has lead to significantly distorted panaroma output images



What type of Panorama?

I took a planar panorama image. The reason being, simply, that the stitched together image would maintain its rectangular shape and thus wouldn’t lead to introduced blank space in the final stitched image.

Happy with Result?

I was happy with the overall result. The image isn’t perfect, by any means. The outputted image has been warped enough that the aspect ratio of objects has definitely been skewed. Other than that, however, the final stitched together image accurately blends that of its 3 image constituents.

multiplying by -x_min and -y_min

In the warpImagePair method the homograph is multiplied by the translation matrix as seen to the right. This affine transformation acts to translate image 1 by xmin and ymin. Without the transformation the stitched together images are offset conside (as evidenced in image to right wh transformation factor is used).

```
[[1, 0, -1 * x_min],
 [0, 1, -1 * y_min],
 [0, 0, 1]]
```