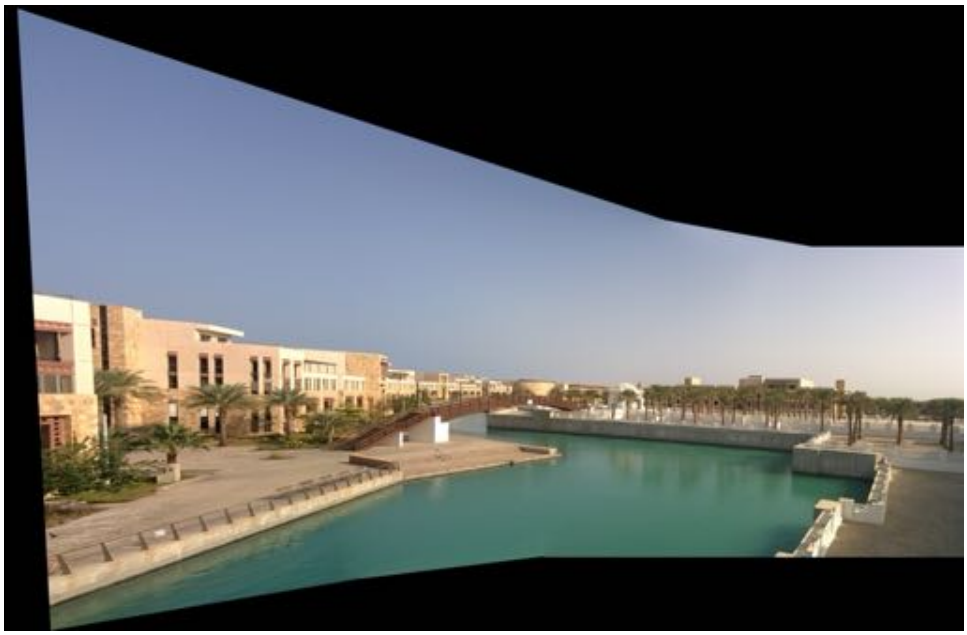
Assignment 3 - Panoramas

Andrew Parmar
CS6475 Fall 2022
aparmar32@gatech.edu

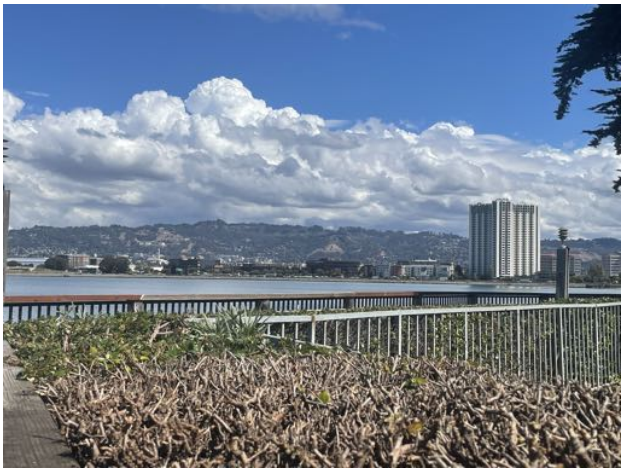


(a) Results: Test Images (UNCROPPED!

(b) Results: My Images

Fig. 1: Panorama Results (UNCROPPED!)

(a) Berkeley, CA from the boardwalk 1



(b) Berkeley, CA from the boardwalk 2

(c) Berkeley, CA from the boardwalk 3

Fig. 2: Original Input Images

I. Setup and Panorama Type

Image	Exposure	Aperture	ISO
1	1/5291	1.6	32
2	1/6494	1.6	32
3	1/5291	1.6	32

1) **Describe the scene that you captured. Be specific and provide details.**
This is a scene of the city of Berkeley in California taken from a boardwalk right outside my building. This was taken a day after the area went through quite a bit of rain so there were a lot of clouds present at the time.

I tried other shots on this boardwalk too, but most angles captured too much water. This specific angle provided a good mix of objects in the distance as well as closer up. There was also a good range of warm and cool colors in the scene with the water, sky, boardwalk, plants, and buildings. I chose this scene specifically for the buildings, hoping they would provide good features to align to.

2) **Describe the camera movement that you used when you took your pictures.**
I used a mini-tripod for these image captures, placing the tripod on a wooden sturdy wooden railing at the site. Mounting the phone in a landscape position, I used one hand to hold the stand and with the other I rotated the ball joint connected to the clamp grip holding my phone. I made sure my rotations went from left to right, making sure to rotate over one-third of the previous shot. I used the grid outline on my display to help with this estimation. To capture the images, I used a Bluetooth remote the prevent any stabilization issues. I continued this process for 4 shorts in total. I ended up using the three consecutive best frames for my experiment and report. I also had my phone settings set to auto, which is why the exposure values vary between the images.

3) **What kind of projection does your panorama use for the combined image canvas?**
My panorama warp step used a combination of affine and perspective projection. We can see from the final image that the leftmost image has a tapering effect going from left to right. Also the light post at the end of the wood plank has a shearing effect. The horizontal lines on the lamp stay horizontal but the pst itself is slanted. There is close to no rotational component or else the lamp shade itself would have been tilted. As the images were captured on a tripod with only rotations in one axis there is little to no rotations happening between the images so features would be at the same angle relative to each other from one image to the next.

II. Discussion Questions

1) **warpCanvas(): Why do you need to multiply x_min and y_min by –1 in the warping function?**
We receive the x_min and y_min values from the getBoundingCorners function, which internally, applies the homography on the first image's corners. A negative x_min and y_min implies that the perspective transformation step in getBoundingCorners has moved the top corner up and to the left. Multiplying these values by -1 in the translation matrix ensures the that the perspective transform's effect is countered and the warped image continues to stay in frame. So negative x_min and y_min, produced a positive translation in the x and y directions, i.e bottom right. Similarly, if x_min and y_min were non-zero and positive, the -1 multiplication would ensure that the translation matrix moved the warped image to the top left. The -1 then is ensuring the entire warped image is retained in the frame.

2) **Discuss your implementation for blendImagePair().**

Overview: For my blendImagePair implementation, I went with a simple approach of using alpha blending and therefore ended up using the functions that we had to implement as part of our code submission. Below I describe the steps in the implementation (for completion, I'm also describing the steps that were provided in the base template.)

Given two images, image-1 and image-2, we first use $findMatchesBetweenImages$ to retrieve features for both images, and a list of matches. Using the two sets of key-points and the list of matches, we use $findHomography()$ which is a wrapper around $cv2.findHomography()$. Internally, $findHomography$ assembles two lists of coordinates, by using the matches list and indexing into the key-points for images 1 and 2. Passing this into the $cv2.findHomography$ gives us the homography matrix that would transform the key-points in image-1 to key-points in image-2. Next, we use this homography, M, to determine the total bounding region of the image that would be produced if image-1 were transformed and combined with image-2. This is done using $getImageCorner$, and $getBoundingCorners$. $getBouningCorners$ return tuples describing the min x and y and max x and y coordinates of the image after the transform. These coordinates, image-1, and the homography matrix are passed into the $warpImage$ function. The mix and max coordinate are used to define the shape of the resulting canvas, and the homography is used to transform image-1 using $cv2.warpPerspective$. $warpCanvas()$ returns the large canvas but with only image-1.

We create a second blank canvas and overlap image-2 on top using the min and max x/y coordinates to place the image. This can be thought of as a pure translation transformation. A third blank canvas is also created. This will be the canvas used for our final output image.

The next section of the code is to determine how to blend the warped image-1 (right-img) and translated image-2 (left-img) together onto the output image. To do so we first create binary masks for the right-img and left-img. These masks are used with bitwise operators to determine the canvas area where only the right image pixels are preset and where only the left images are present. The area where both left and right are present is the overlap region. For our blending steps, we will use the left-region mask to directly apply all the pixels from the left image, similarly, we will use the right-region mask to apply all the pixels from the right image.
The overlap region is where we want to perform alpha blending. We use the $findDistanceToMask$ helper function to determine the distance from each boolean-False pixel in the region mask to the nearest boolean-True pixel. We do this for both the left and right region masks to get two distance masks. Using these distance masks, we can now calculate the ratio of how far a pixel is from the right mask by using the formula $r_distance/(left_distance + right_distance)$. This alpha ratio will be higher for pixels that are closer to the left region mask than the right.
With the alpha-weights generated, we iterate through each color channel, i, and calculate the total values for the overlap region using the following formula:

$$output[:,:,i][overlap\_mask] = alpha[overlap\_mask] * left\_img[:,:,i][overlap\_mask] \\ + (1 - alpha\_weights[overlap\_mask]) * right\_img[:,:,i][overlap\_mask]$$

This output image is returned and passed in as image-1 for the next iteration along with image-3.

III. Project Retrospective

Consider the following questions with regard to developing your custom blend function.

1) **What worked well?**

   When capturing my images, I'm glad I didn't just rely on one scene. Having multiple sets of images to put through my pipeline was key. Some of the image sets did not produce good results but having options helped. The image that did end up working well, had well-defined structures in the scene, buildings, and railings which helped with feature detection, and matching. I did have to tune the num of matches though. If that parameter was fixed it would not have been so straightforward to get the algorithm to work on a variety of scenes. The provided helper functions also helped in directing my thinking towards a solution, special the region masks.

2) **What did not work well?**
   My final output image showed quite a bit of empty space on the left in both the sample images provided and my original shots. On my original shots, the warping is pretty extreme on the left end, causing the objects to appear slanted. I was not able to figure out what aspect of my pipeline was causing the warp to behave that way. My other sets of original images were so warped that they were not usable even with tuning. I specially on images that had water in them. I took the images at noon, so the water was pretty reflective. That would most likely have cause a lot of variation between the images and probably had false-postivie when detecting features. The pipeline wasn't sophisticated enough for these types of scenarios.

3) **Were there any problems you could not solve?**
   I wasn't able to figure out how to align the leftmost image so that the edge was vertical after the transformation. Though my code passed Gradescope, it wasn't clear even after watching the lectures multiple times what I was doing wrong in my implementation that caused the left-most image to not line up with the left edge of the output canvas.

4) **What could you add to your pipeline to improve upon your existing blending implementation?**
   The current pipeline takes in images left to right and creates a panorama. What I noticed is that with each iteration, the leftmost image gets warped more and more. An alternative pipeline would be to create panoramas of every pair of adjacent images and then blend adjacent panoramas together. This could be repeated recursively until two images are blended together for a final image. I speculate this might help with some of the more extreme warping seen on the left edge.

5) **If you started the project over again, what could you do differently and how would you go about doing it? Do NOT say there is nothing you could do differently. There's always a way to improve.**
   Based on our pipeline's capabilities, I would have chosen a different set of pictures. My location had large areas covered by either a body of water or clouds. My final results show ghosting effect in the portion of the image covered by clouds. I would also try and experiment with using cuts instead of blends - as the blending seams are already pretty good, this would be more so for experimental purposes rather than improving visual quality. I would also start with smaller original images from the beginning to cut down on processing time, as I spent a good amount of time waiting for the pipeline to run on my full size images.

References

[1] Ed Lessons, 5-1:3 , Accessed: Sept 18, 2022
[2] OpenCV, Tutorial, Feature Homography, Accessed: Sept 18, 2022
[3] StackOverflow, Drawing Contours, cv2.drawContours, Accessed: Sept 18, 2022
[4] OpenCV docs, drawContours, drawContours, Accessed: Sept 18, 2022
[5] OpenCV docs, perspectiveTransform, perspectiveTransform, Accessed: Sept 18, 2022
[6] OpenCV docs, warpPerspective, warpPerspective, Accessed: Sept 18, 2022