Greg Matthews
CS 583 – Section 900
May 9th, 2017

Computer Vision
Project 2: Panorama

Planar to Cylindrical Images:

Objective:

- Convert the captured images of the panorama from planar coordinates to cylindrical coordinates

Steps:

1. Convert the captured image points into their respective cylindrical coordinates: θ and height h
2. Convert the cylindrical coordinates onto the unit cylinder: $(\hat{x}, \hat{y}, \hat{z})$
3. Normalize the unit cylinder coordinates to (x, y) by inverse warping and interpolation
4. Apply radial distortion correction

Challenges:

There wasn't much trouble to be had with creating the cylindrical coordinates, with an intuitive conversion from image coordinate points, to cylindrical, and accounting for radial distortion, the function didn't pose many problems.

Lucas Kanade Optical Flow:

Objective:

- Compute the displacement between 2 images by estimating the images optical flow using Lukas Kanade.

Steps:

1. First an intensity mask was made to only pick image points that are bright, which will help in the optical flow estimation.
2. The partial image derivatives where then computed by convolving the images with a Sobel kernel for both the x and y indices.
3. The partial products are to then be ANDed with the intensity mask.
4. The AtA and Atb matrices are finally created and solved via total least squares for the velocity vector [u,v].

Challenges:

One of the main issues that I faced was finding a way to convolve the images with the Sobel kernel, when there are 3 color values that have to be taken into account. We've been shown so far how to convolve the images given that they are grey scale, however given that we now have RGB values, each color must be convolved separately with the Sobel kernel, and the combined together to get the complete partial derivatives. There was also issues with the matrix to being

invertible, however this is guaranteeing that there is an error in the implementation then.

Gaussian Pyramids:

Objective:

- Estimate the optical flow more efficiently by scaling down the image size until it is less than a pixel, using Gaussian blending each ½ size decrementation from the original size.

Steps:

1. Create the Gaussian kernel with the desired sigma, and kernel size values.
2. Convolve the images with the Gaussian kernel, and then subsample the image

Challenges:

This was also a relatively easy implementation, the only tricky part was again how to correctly convolve each RGB value with the Gaussian kernel.

Gaussian Pyramids with Lucas Kanade:

Objective:

- Estimate the refined optical flow by using both the Gaussian pyramid and Lucas Kanade implementations.

Steps:

1. Create the Gaussian pyramids for both the images
2. Starting at the coarsest scaled image, translate the second image by the initial displacements that were calculated beforehand, and then run iterative Lucas Kanade to obtain the refined displacement for the current pyramid image.
3. Scale the calculated displacement to be used in the next Gaussian pyramid image.

Challenges:

Figuring out how to properly scale the displacement with each Gaussian pyramid iteration was slightly difficult, because you have to consider the initial displacement, as well as incrementing this displacement with the refined displacement you receive from running the iterative Lucas Kanade algorithm

Building the Panorama:

Objective:

- From running the Pyramid Lucas Kanade implementation and receiving the final displacements, build the panorama by aligning images correctly onto a blank canvas.

Steps:

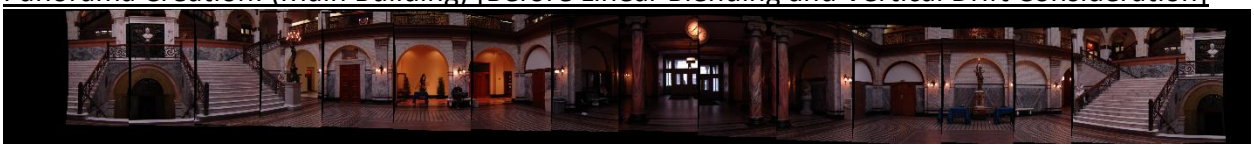
1. Place the last image that should be warped to align with the first image on the panorama.
2. Iterate over the rest of the images and increment the displacements with each steps to after each loop.
3. Perform linear blending to remove artifacts that are seen in between the images.
4. Interpolate the image and get rid of any vertical drift from the image

Challenges:

This was the most difficult section of the project, stemming for all the considerations necessary to account for linear blending, vertical drift, and stitching the panorama together, it was truly a daunting task. The linear blending technique from the previous Homography function helped in thinking of how to tackle that challenge. The vertical drift was also at first tricky to deduce how to do, mostly stemming from not knowing what numpy has to offer for this type of implementation. I ended up using an Mgrid and indexing over the columns, pushing up the index values incrementally by the total drift.

Note for images below, the final displacements that were emailed by paras were used for the images, as my Lucas Kanade implementation was not accurate. For my images, I used the initial displacements found in the test file.

Panorama Creation: (Main Building) [Before Linear Blending and Vertical Drift Consideration]

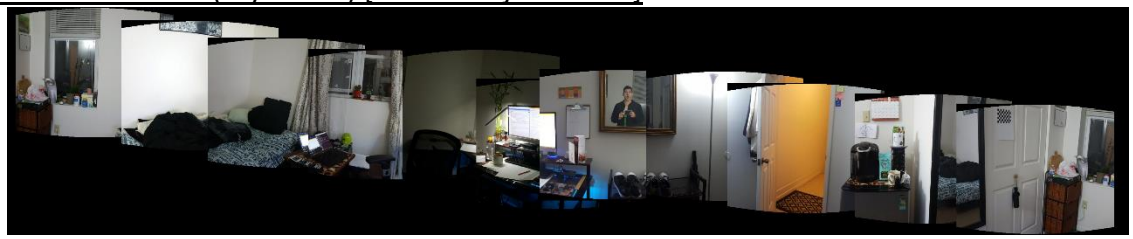


Panorama Creation: (Main Building) [Adding Linear Blending]

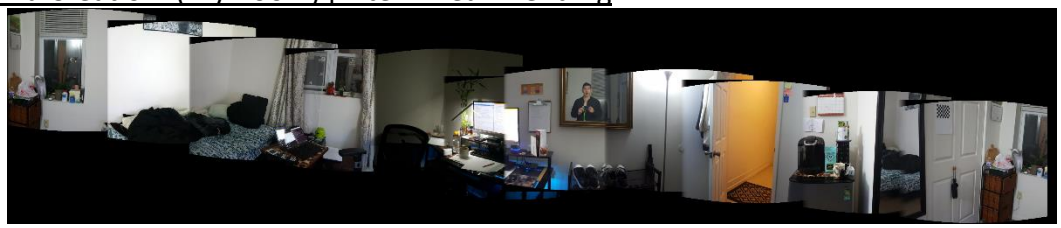
Panorama Creation: (Main Building) [Forming a Loop at beginning and end of Panorama]

Panorama Creation: (Main Building) [Fixing Vertical Drift]

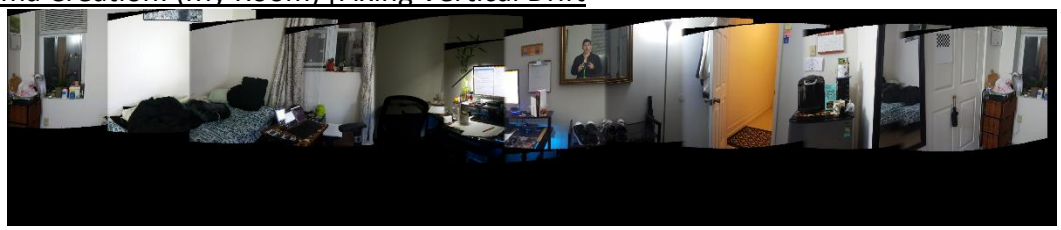
Panorama Creation: (My Room) [Before Adjustments]



Panorama Creation: (My Room) [After Linear Blending



Panorama Creation: (My Room) [Fixing Vertical Drift



Panorama Creation: (My Room) [Cropping and forming loop]