ENSC 474

Assignment 3

Cyrus WaChong

cwachong@sfu.ca

301306459

Question 1:

The question simply asked the student to implement the bilinear interpolation method taught in class. By simply inputting the image, with the x and y coordinates of the point in question, the program will return the intensity at that point, approximated using bilinear approximation.

```
This is the output with a x+0.25 and y+0.35 to show how similar they are
ans =

    0.3813

This is the output without the change in x or y

 0.385542
```

Figure 1: Example output of bilinear interpolation function

```
Question_1(Grayscaled, 15, 40);
xyz= bi_lin_interpolate(Grayscaled, 15, 40);
```

Figure 2: two function calls that show demonstrate interpolation is working

Above, we can see me passing the “requested” x and y values into the functions “Question 1” and “bi_lin_interpolate”. Question 1 simply adds 0.25 to the x value, and 0.35 to the y value, and then calls “bi_lin_interpolate” with those new values.

The values are very similar, showing that the method shown in class works to approximate the value of a “pixel” that is not an integer value.

Question 2:

This question was quite a bit heftier, asking for shifting and rotation of our eye image, and do rotate a grid, and showing a difference at every point between this grid, and our “x” values. I used the interpolation function for both the rotate and shifting of the eye.

This leads to many images, like the ones that will be posted next.

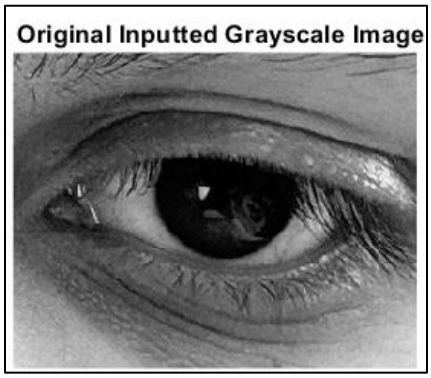

Figure 3: Original image inputted

Above we can see the original image that was then rotated and shifted in the next images.

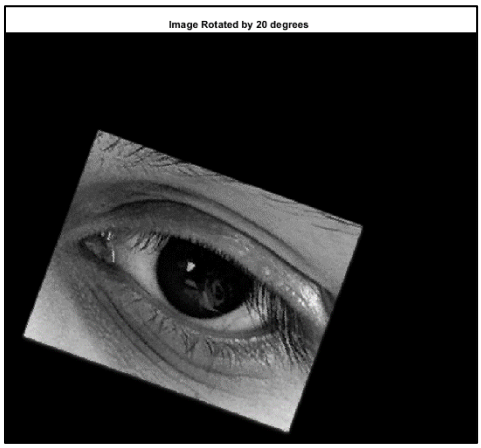

Figure 4: Rotated image by 20 degrees counter clockwise around the origin (top left)

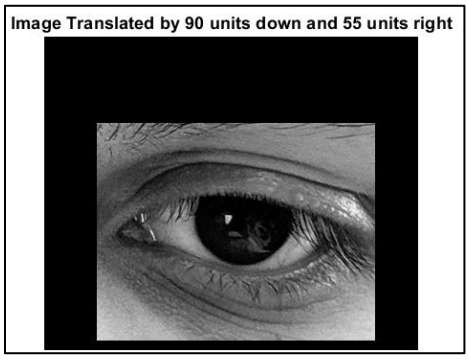

Figure 5: Image shifted 90 down and 55 to the right

In these two images, padding of back pixel had to be added so that the final product could include the entire images. If this was not done, we would get cut off images, and by doing this, we have quite a few redundant pixels, but complete rotated and shifted images. The rotated image also has an averaging function (from the 4 neighbours), getting rid of any black pixels.

The next task was to graph the phi, which is a grid going through the same rotational transformation as our image, which as a rotation of 20 degrees counter clockwise.

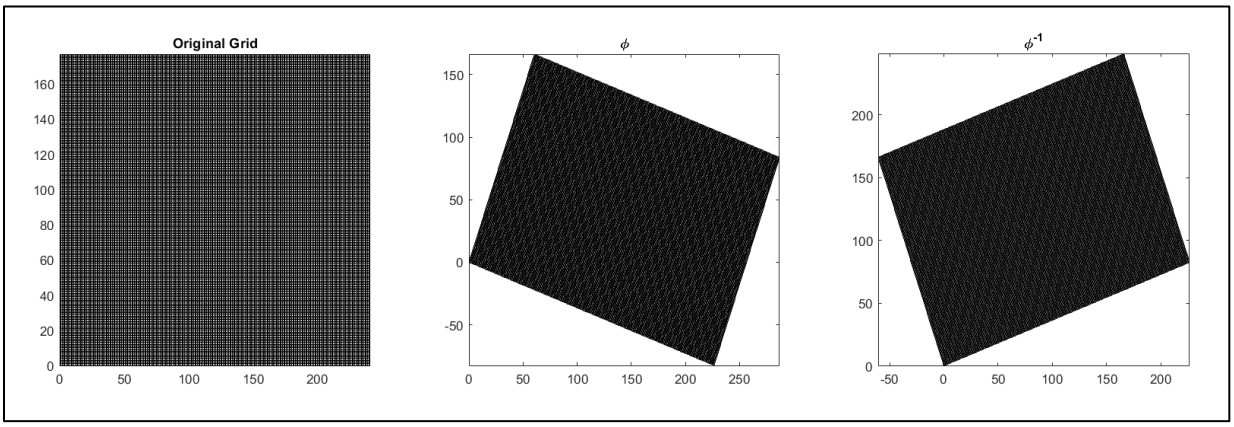

Figure 6: original grid, with phi and phi^-1 rotations

Above, we can see the original grid (which looks quite dark, but it is very zoomed out, if code is run, you can zoom in). To the right of the original, we can see the grid rotated 20 degrees counter clockwise, and the rightmost figure shows the image getting rotated 20 degrees clockwise.

The next task was to graph the d(x), which is the difference between phi(x) and x at every point (using quiver), and d^-1(x).

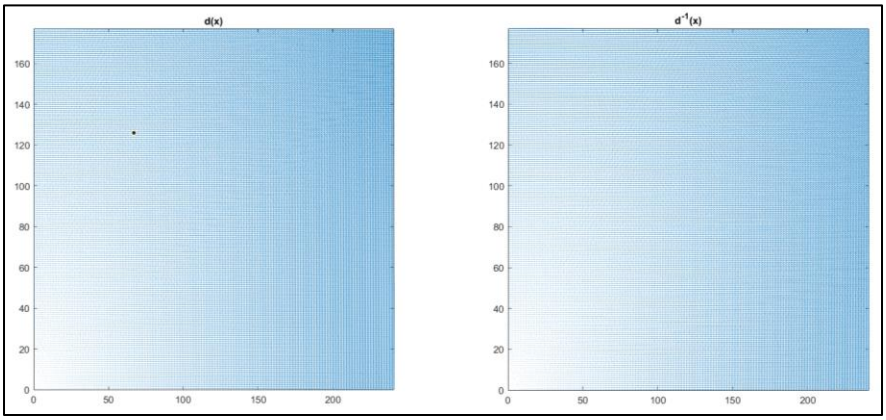

Figure 7: d(x) and d^-1(x) plots

Above, we can see the d(x) and d^-1(x) functions. These are the differences between phi(x) and the x value itself.