EECE 253 IMAGE PROCESSING

LABORATORY ASSIGNMENT 5

Jack Minardi
06 Nov, 2011

Abstract

This paper reports the results of experiments done to explore different methods of manipulating image sizes and perspectives. Two different scaling algorithms are demonstrated, the simplest of which is nearest neighbor. This method simply duplicates pixels up to the desired image size. The other method used is bilinear interpolation, which while a little more involved, is still a basic technique. It interpolates linearly between known values rather than simply duplicating. A method of rotating images about their center was also demonstrated. And finally a method of skewing images to adjust for perspective was explored. This method can be used to fix angles and unskew distorted images.

Table of Contents

EECE 253 IMAGE PROCESSING
LABORATORY ASSIGNMENT 5

Abstract
Table of Contents
Introduction
Description of Experiments
Results of Experiments
    Experiment 1
    Experiment 2
    Experiment 3
    Experiment 4
    Experiment 5
Conclusion

Introduction

Image resizing is useful in many applications. Often times the medium though which an image is displayed can limit the total or useful file size. For example if an image will only ever be displayed on a website, it may not have to me many megapixels big. Or if you are transmitting video over a limited bandwidth line, you might have to downsize the image spatially, or reduce the number of bits per pixel while keeping the dimensions the same. The other methods explored such as rotating or skewing can be useful in other image manipulation tasks.

Description of Experiments

There were five experiments performed in this lab. Matlab was used for all the computation. All the scripts that were written can be found in the appendix and in the zip file this report came in. The results were somewhat subjective and discussions can be found in the report below.

1. Write a function to quantize a color image. That is, it should input a nominally 8-bit per pixel per band image (a truecolor image of class uint8) and return an image that is quantized to n bits where n $\in$ {1, 2, . . . , 7}.

The function should input a color image of any size and return a new color image of the same size. The function also should accept an integer argument that specifies the number of bits per pixel per band in the output. The function call should look something like J = QuantImg(I,n); where I is the input image and n is the number of bits per pixel per band in the resulting image. The output image, J, should remain class uint8 even though each pixel will have fewer than 8 bits of intensity resolution.

(a) Test your function on a single original image and produce 7 new images from it, one for each

quantization level (7 bits through 1 bit per pixel per band). Make a table of quantization levels and number of colors in the the result for a one band image and a 3-band image. Before you display or save the n = 1 image you should first scale the results so that the individual pixels have values 0 and 255. Otherwise the image will appear dark. I suggest using a small image (say around 256 × 256 in linear dimension) for this part so that they can all be displayed on 1 or 2 pages in your report.

*(b) Experiment with adding noise to the image, both before and after quantization. For the experiments use a quantization level of 3 bits. To add the noise before quantizing do, for example, I = uint8(double(I)+sigma*randn(size(I))); before applying your quantization routine. When adding the noise after quantization of the image, be sure to quantize the noise to 3 bits before adding it to the 3-bit quantized image. Determine which which value of sigma produces the best results in terms of the reduction of false contours and determine whether it is better to add the noise before or after quantization of the image. I suggest you first try powers of 2 for sigma then refine your search once you've bracketed the results.*

2. Write a program to implement a nearest neighbor resampling of an image. The function's arguments should be the input image and the number of rows and columns in the output image.

(a) Find a truecolor image, I, with dimensions in the 512 to 768 range. Use your function to reduce the image to 1/4 size and 11/16 size. Then use the function to increase the size of the 1/4 image to its original size. That is, if I is R × C × 3, nearest-neighbor downsample I to get a R/4 × C/4 image, J. Then use the function to upsample J to a 4R/4 × 4C/4 image, K. Do the same for the 11/16 size. Display the results next to the original image and comment on their appearances.

(b) Use your NN resampling function to increase the size of I by factors of 17/9 and and 3/2. Display the results and comment on their appearances.

3. Write a program to resize a color image using bilinear interpolation. Repeat the two sections of problem 2 using this new function. Hint reuse your nearest neighbor algorithm to select the actual (fractional) pixel locations that determine, for each output pixel, which four input pixels to interpolate.

4. Write a program to rotate an image an arbitrary angle in degrees. Positive rotation should be clockwise since the y-axis points downward in an image. The output image should be large enough to accommodate the complete result, but not larger. Use center of the image as the center of rotation. Hints: (1) Use backward mapping. First determine the size of the output image. Then for each pixel location in the output image determine the location of the pixel in

the input image to copy. (2) To find the corresponding input location, rotate the the output pixel location in the opposite direction and round or truncate the results. If the computed location of the input pixel is out of range, zero the output pixel. Display both positively and negatively rotated versions of the same image.

5. Find a true-color image on which to perform linear warping. Display the image and select in it four or more points as warp-point origins using [x y] = ginput(). Create a set of valid target points. Generate a planar homography that maps the input points to the output points. Use the homography to remap the entire image.

Results of Experiments

Experiment 1

a)

Number of bits	Number of colors (1-band)	Number of colors (3-band
n	2^n	2^{n+2}

Figure 1-7 - Quantized images (1-7 bits)

b)

Figure 8. Image quantized to 3 bits with noise added before quantization

This section of the lab dealt with adding noise to the image before quantization. I found random noise with a sigma of 4 produced the best results. It needs to be added before the quantization for the effect to work.

Experiment 2

Figure 9. Image used in subsequent experiments.

a)
This experiment dealt with image resizing using the nearest neighbor technique. The following two images were downsampled to 1/4th and 11/16th the size and then upsampled to see the results. As you can see there are blocky artifacts because this algorithm simply duplicated the pixel values to fill in the new area, or else simple deleted pixels to downsize. This is the most basic form of image resizing.

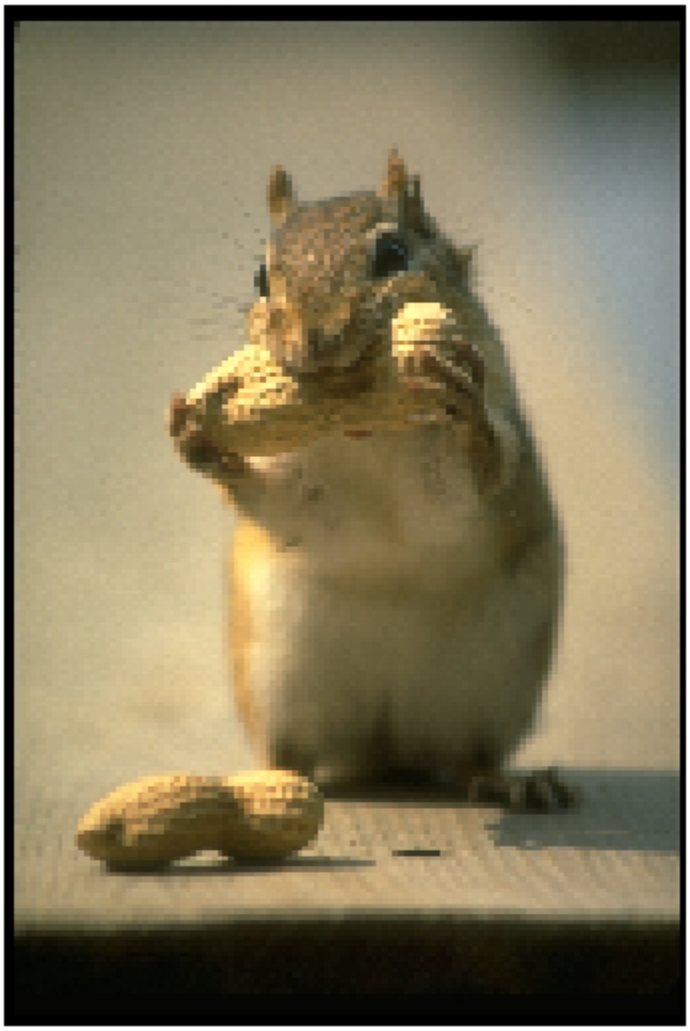

Figure 10. Downsampled by 1/4th and then blown back up again

Figure 10. Downsampled by 11/16th and then blown back up again

b)
The following images were blown up by different scale values to point out the effects of the nearest neighbor technique. To see actual detail I have selected portions of the image and shown them here at 100% You can see the pixelization in these images.

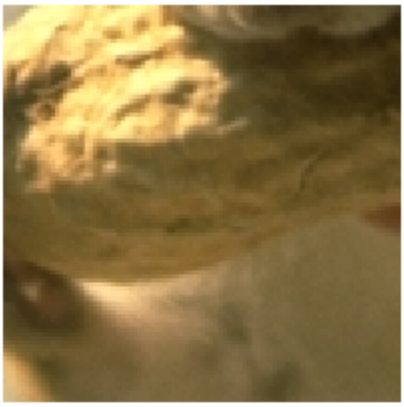

Figure 11. Subsection of the image blown up by 17/9ths

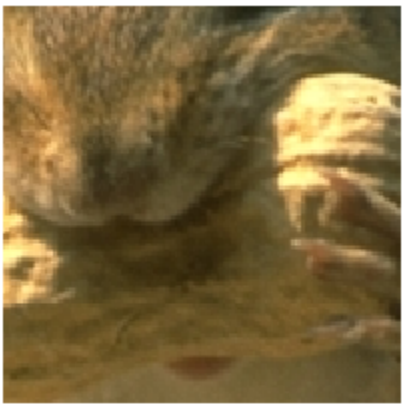

Figure 12. Subsection of the image blown up by 3/2nds

Experiment 3

This experiment repeated the process of experiment 2 but with a new resampling algorithm known as bilinear interpolation. Rather than simply duplicating pixels, this algorithm interpolates the values of the pixels created between known values. When shrinking an image, the new pixel value takes into account the value of the 4 neighboring pixels. Note how this technique produces much smoother results.

a)

Figure 13. Image downsampled by 1/4th and then scaled up to original size.

Figure 14. Image downsampled by 11/16ths and then resized to original dimensions

b)

Figure 15. Subsection of image blown up by 17/9ths

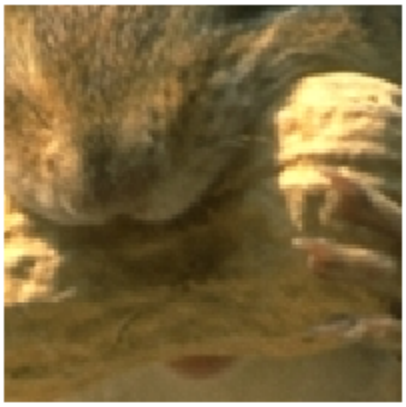

Figure 16. Subsection of image blown up by 3/2nds

Experiment 4

This experiment demonstrates how to rotate an image an arbitrary number of degrees about the center of the image. The following two images were rotated by +45 degrees and -45 degrees. The extra pixels were padded with 0s (black here)

Figure 17. Image rotated by +45 degress.

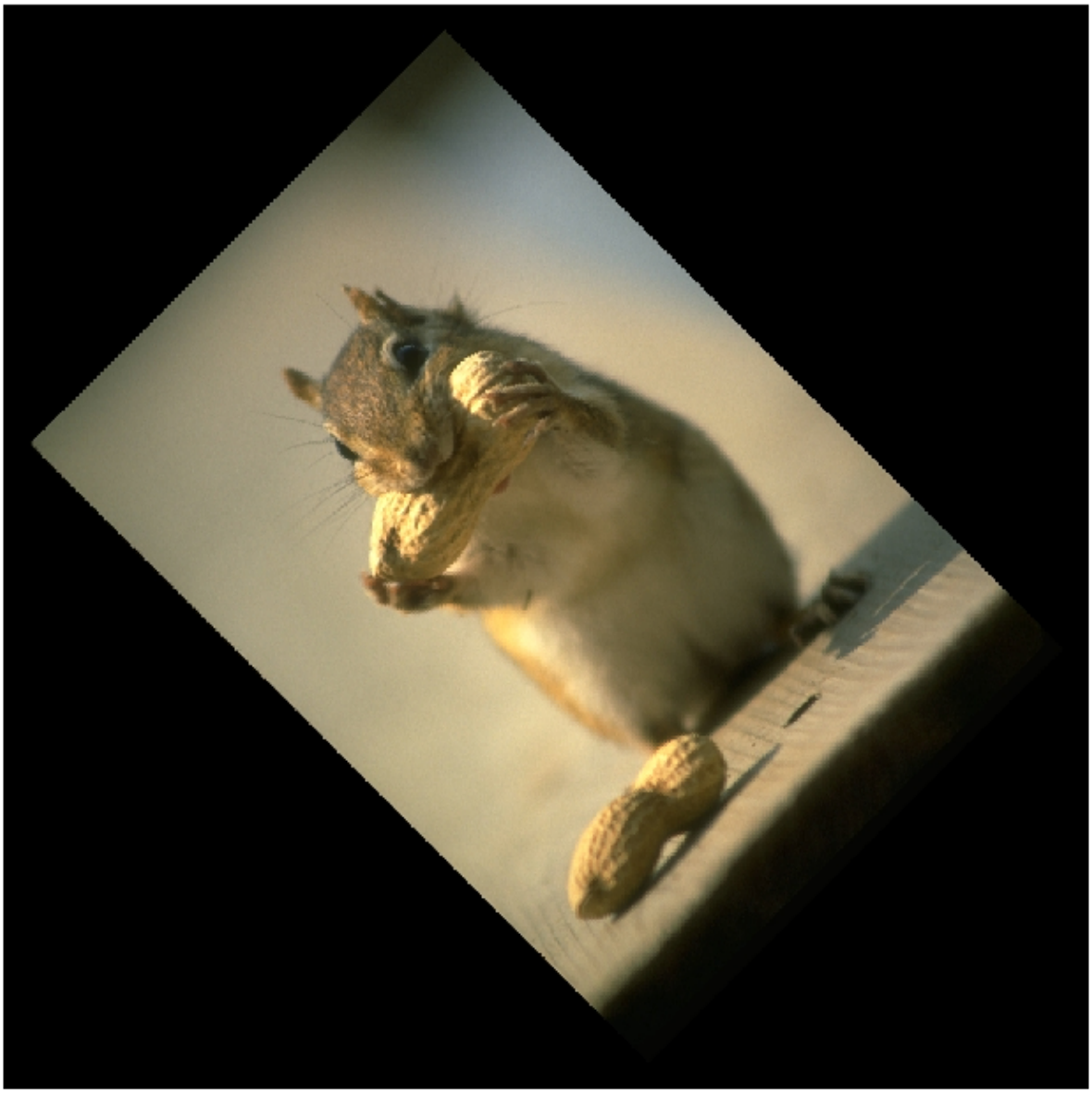

Figure 18. Image rotated by -45 degrees.

Experiment 5

This experiment dealt with linear remapping of images. I was able to correctly compute

the two different A matrices, and compute its singular value decomposition. Using this I found the minimum value and used that to select a vk vector. Using this vector I computer H and then H inverse. This was used to remap the image points. I made a slight error somewhere because the resutls are not exactly as they should be. Below if the original image and the ‘fixed’ image. I was attempting to straighten out all the sides.

Figure 19. Original image to be warped

Figure 20. Attempt to fix the distortion

Conclusion

All of the preceding experiments worked with resizing images in some way. Either by reducing the number of bits per pixel, by actually changing the number of rows and columns in the image, by rotating the image, or by skewing the image to adjust for perspective distortion. All of these are useful image manipulation techniques for either storing or displaying an image in the way that is desired.