Jonathan Ksiezopolski

Digital Signal Processing Project

Rami Madbouly, Peter Cai

15 May 2015

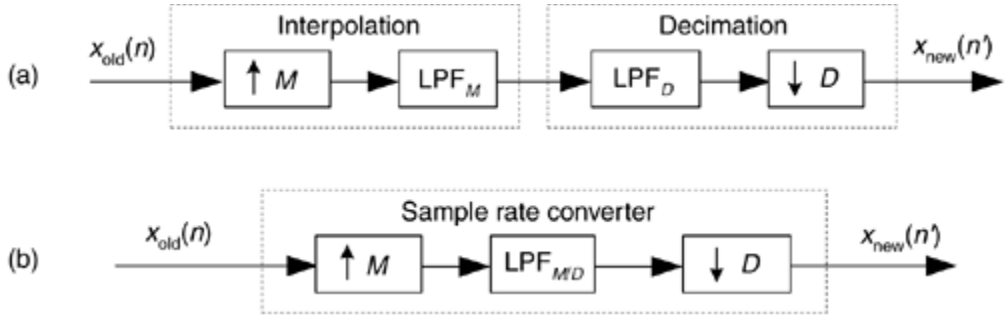
Color Image Processing

Abstract

Color image processing is a very broad field consisting of core principles and phenomena of digital signal processing and computer vision. The main goal of the project was to manipulate, filter, and track RGB colors using MATLAB. Doing so we demonstrated the various phenomena and problems we could of had in the realm of color image processing and some key elements in digital signal processing. The overall results are successful using a combination of blob analysis and color image filtering, but it was not fully automated.

Recording and Sampling

Recording and successfully tracking color all starts with understanding the potential problems that could arise from simply recording. The main concern was the image and video data that enters MATLAB is bundled into a matrix with each pixel accounted for. At a high resolution this would mean we would have to compress any sequence of images or video stream that comes in since it would eat up processing time and result in a hideous lag. In our project this would mean a compression of a sequence of one hundred images and each image would lose quality. This quality loss due to compression is called decimation which in order to get back the best quality we would use interpolation using a Gaussian filter to achieve a smaller cleaner image. [1] [2] This process can be similar to the illustrated block diagram below only in our case it would run in reverse. [2]



Where x(n) is our 100 image input, our low pass filters are the Gaussian filter, and M is our sampling factor that equals to the inverse of the period times its length

$$M = \frac{1}{NT}$$

Fortunately, the default for the webcam chosen has a format of RGB at 640x800 (512000 pixels) rather than most cameras that now operate at 1080x1920 (2073600 pixels) high definition. The amount of pixels in this RGB format along with the fact that we are evaluating 100 images instead of a live stream video allows us to easily extract red, green, and blue matrices of color data without any true limitation or loss of information.

Color Filtering

To imagine this RGB matrix we can mathematically represent it as a weighted sum. For example think of the weighted sum as [1]

$$x(n) = 0.5r+.44g+29b$$

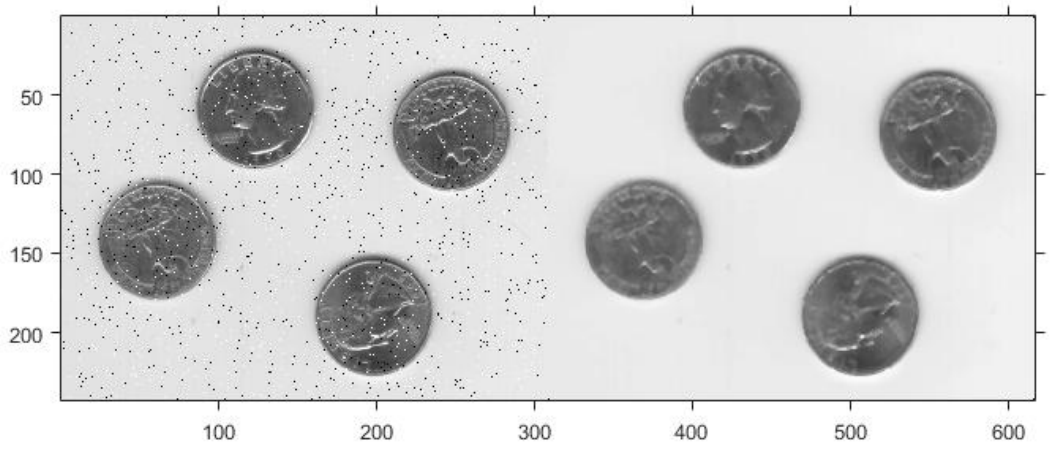
With r, g, and b as the (x,y) color value matrices of the image. Making this sum can be easily done using the rgb2gray algorithm in MATLAB and allows us to extract the color values needed. In the program we give the user a choice of choosing red, blue, or green to extract out. Once picked we can filter out the full sum of the grayscale from let's say 0.5r leaving only (x,y) values of one color resulting in an image like this

All that is left in that picture is a spectrum of red put in grayscale as seen where the whitest spots are more concentrated in its representation of red. This is done so by using a simple function called 'Imsubtract(X,Y) which subtracts each element in the array Y from the corresponding element in the array X. [1] Keep in mind that this works on any array that has real non sparse numeric arrays in which we do have when dealing with the grayscale image in MATLAB. Non sparse, in the case of arrays, simply means that there are very little amounts of zeroes.

Image Filtering

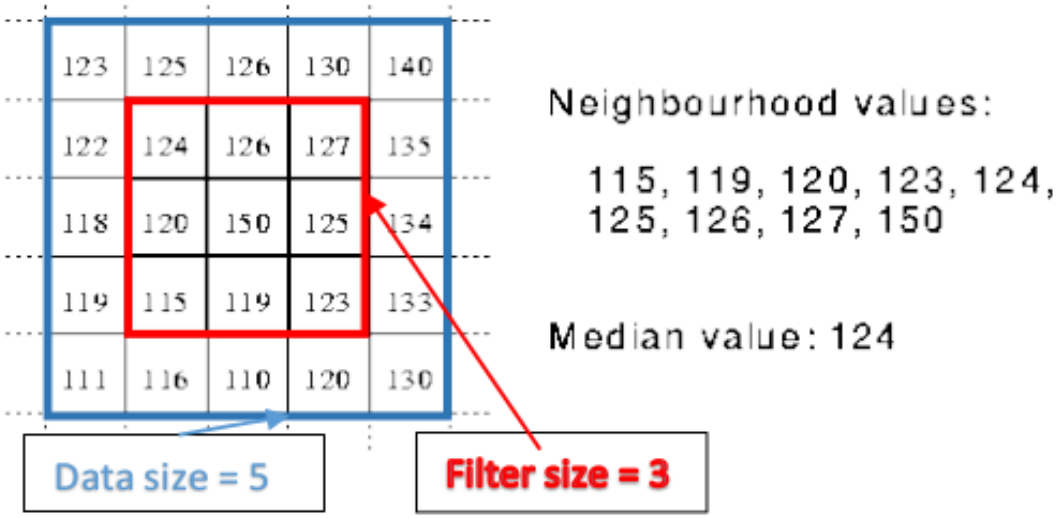
Much like Imsubtract(X,Y) our median filter also requires a real, non-sparse, and numeric matrix to work with in order to filter out any spots that would otherwise hinder our progress of detecting color. [1] In this case, since we are working with red in grayscale, the biggest annoyance is the interference of small blotches or lettering (black pixels) that isn't red (white pixels) in the middle of a red object. This annoyance is called salt and pepper noise. [3] This noise may cause the program to detect red all the way around the lettering and would otherwise give a messy detection of multiple objects when we get to blob analysis.

For a more dramatic example of this noise let's take this image [3]



Although the image is not in grayscale it's easy to see how even these small dots have the potential of ruining an image analysis. The way this median filter works is simply a discrete convolution of our new input (the red grayscale) with a median filter as our transfer function. Where our median filter takes the average values of a [3,3] neighborhood in the values of the matrix and zero pads the perimeter of that neighborhood size.

Here's an example [5]

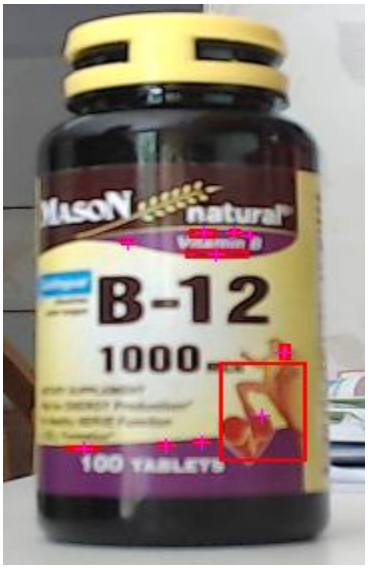


This process ends up smoothing out the salt and pepper noise to better represent the surrounding area values making it easier to detect the full area.

Once the noise is reduced we can transform the image into a binary mapping of '1' for white and '0' for black. The function im2bw(color_grayscale, level) replaces all pixels in the input image as '1' or '0' using 'level' as a luminance indicator of that color in the image. The method of finding this luminance or the threshold of one color is a little complicated. The formulation to find the threshold is derived from a gray level histogram in Discriminant function analysis. [4] Discriminant function analysis is a statistical analysis that predicts a categorical dependent variable by one or more continuous or binary independent variables. In this case luminance is the eigenvalue of the matrix that indicates how well a function differentiates one group from another. [4] Through experimentation we found that values in the ball park of 0.2 tend to differentiate red and blue objects fairly well with green needing a much smaller threshold of 0.05. In the image below we can now see the true red in the picture farthest to the right. Which although it picked up a little purple here and there just means our values can use some tweaking.


Basic Blob Analysis

At this point we have acquired a binary mapping of what is considered truly red in the image and all this is left is to form a box that would surround that red to the best possible fits. The method of doing so is known in computer vision as blob detection which is not only limited to color, but can be used for brightness or any imaginable comparison in an image. [1] Using bwlabel(image,x), we can group a sufficient amount, x, of logic ones or "red" in the matrix of the map. Once we can do this setting the box to surround each cluster of logic ones along with setting a centroid of that box just takes a simple command of regionprops(x,y). Once we can set this box we can return the regular image and set the extracted bound boxes on it giving us this result.



Conclusion and Further Applications

After all is said and done we were able to successfully track red, blue, and green areas on an image through RGB color manipulation, image filtering, and basic blob analysis. We did run into problems of finding a luminance value for green, but fiddling with the values we found that green happens to have a low threshold. The hardware limitations and potential sampling problems were avoidable keeping all settings at a default RGB format 640x800 on a webcam. The applications of such color tracking can be utilized in autonomous control where a car could scan the road for a distance between colored lines that would represent a lane. Other applications can be involved in triggering sensors that need differentiate light or even for tracking the speed of an object by calculating the time and pixel/distance ratio of the bounding box.

References

[1] Documentation. (n.d.). Retrieved April 20, 2015, from http://www.mathworks.com/help/

[2] Section 10.3. COMBINING DECIMATION AND INTERPOLATION. (n.d.). Retrieved April 27, 2015, from http://flylib.com/books/en/2.729.1.102/1/

[3] Documentation. (n.d.). Retrieved April 19, 2015, from http://www.mathworks.com/help/images/ref/medfilt2.html

[4] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 9, No. 1, 1979, pp. 62-66.

[5] Two dimensional array median filtering. (n.d.). Retrieved April 24, 2015, from http://stackoverflow.com/questions/26870349/two-dimensional-array-median-filtering