Image Processing and Analysis

Henrik Drægni Stolpnes – University of Bergen

Abstract

The task is making a program that could recognize if a logo is present in an image. The program does this by first finding areas in the two images that are going to be compared, and then searches through these areas for similarities.

Introduction

Image processing and analysis are found in many of the applications used in everyday life; from analyzing stars to adding a filter on social media. Today one can easily manipulate images with software. The code behind the software is made of advanced algorithms, this also applies to filters of social media. An image is a function of two variables, *f* (x, y). X and Y is the position, making up a point. The value of *f* in every point is called intensity.

When speaking of image processing one talks about changing an image, based on the information it contains. This does not add any information, but can give a result that changes what and how the information is displayed. This can be useful in several cases. If one wants to improve the quality of an image, changing it to highlight certain areas or just remove unnecessary data.

Image analysis is about getting new information based on the information in the image. Examples of this could be calculating the temperature of a star or recognize a face in a crowd. Every time an operation is done on an image to get information it is image analysis. It is all about understanding the image that is processed. To do this there is often needed to run calculations on the data from the image.

The program in this report uses both image processing and analysis. It is far from the state of the art software, but should be able to process and analyze images to give results. It runs image processing to derive information that is useful for further analysis. The algorithm then looks at that information to run further calculations and analysis. Compared to modern software and programs this application is significantly slower, contains simple errors and can at times be hard to work with due to the limited experience of the programmer, combined with it being done over the course of a semester.

The program

The program is made in Visual Studio 2017, a program for writing code and developing software for windows. ReSharper 2017.2 is used on this project and is a plugin for Visual Studio made by JetBrains. It analyzes code and helps with error correcting. The code contains 9 classes, and can be divided into four bigger parts. The first being the interface. This was done by both writing *xaml* [1]code and using the Visual Studio Designer in the MainWindow.xaml file. The interface is where the user interacts with the program and gets all the feedback.

Next, there is the edge detection part of the program. This is written in Visual C# language as the rest of the program except the interface. The edge detection defines edges for later use in the program.

The third part of the program is the shape detection. This part takes the information from the edge detection to define shapes in an image.

The last part of the program is the comparison of *areas of interest* from an image, called *subject*, with the areas from another image containing a known logo. The image containing a logo, called *logo*, is the image one imports after the subject. This part of the program runs all the code from the previous parts. Interface of program (Figure 1), with area for displaying images marked with a red rectangle and buttons for each of the parts of the program. Figure 2 shows the flowchart of the program.

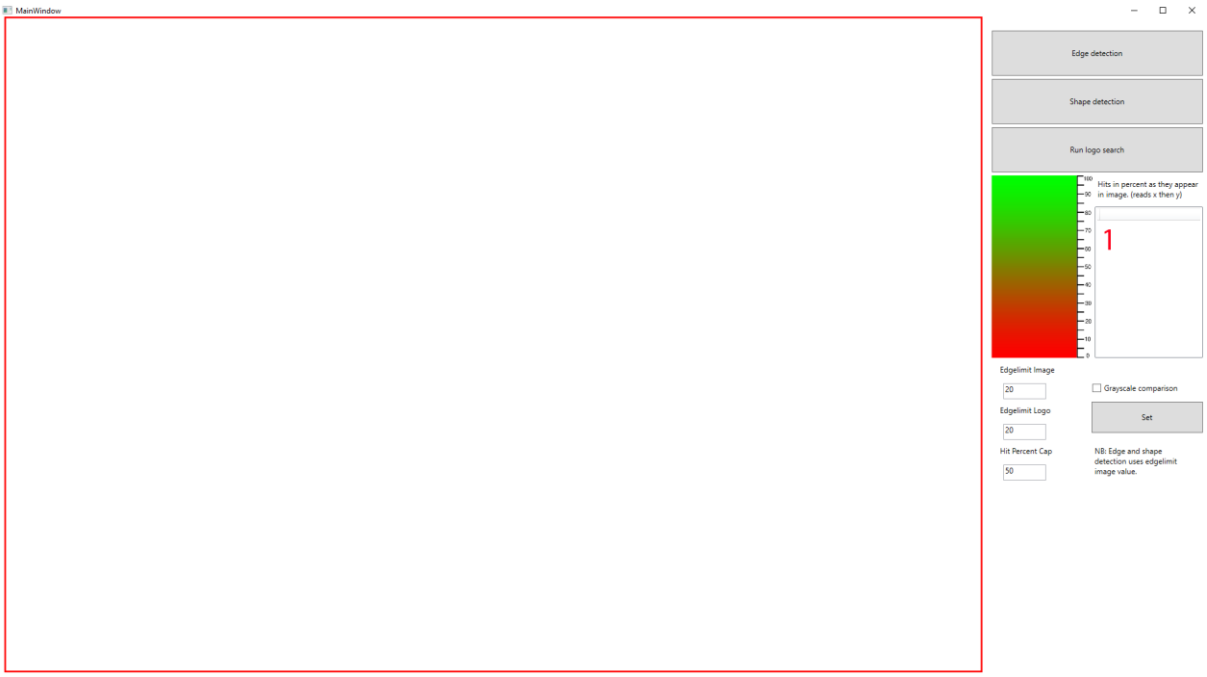

Figure 1: Interface of program, with area for displaying images marked with a red rectangle and buttons for each of the parts of the program. The white box marked with a red 1, displays the hits from the "run logo search".

[1] "Extensible Application Markup Language, or XAML (pronounced "zammel"), is an XML-based markup language developed by Microsoft. XAML is the language behind the visual presentation of an application that you develop in Microsoft Expression Blend, just as HTML is the language behind the visual presentation of a Web page. Creating an application in Expression Blend means writing XAML code, either by hand or visually by working in the Design view of Expression Blend." [1]

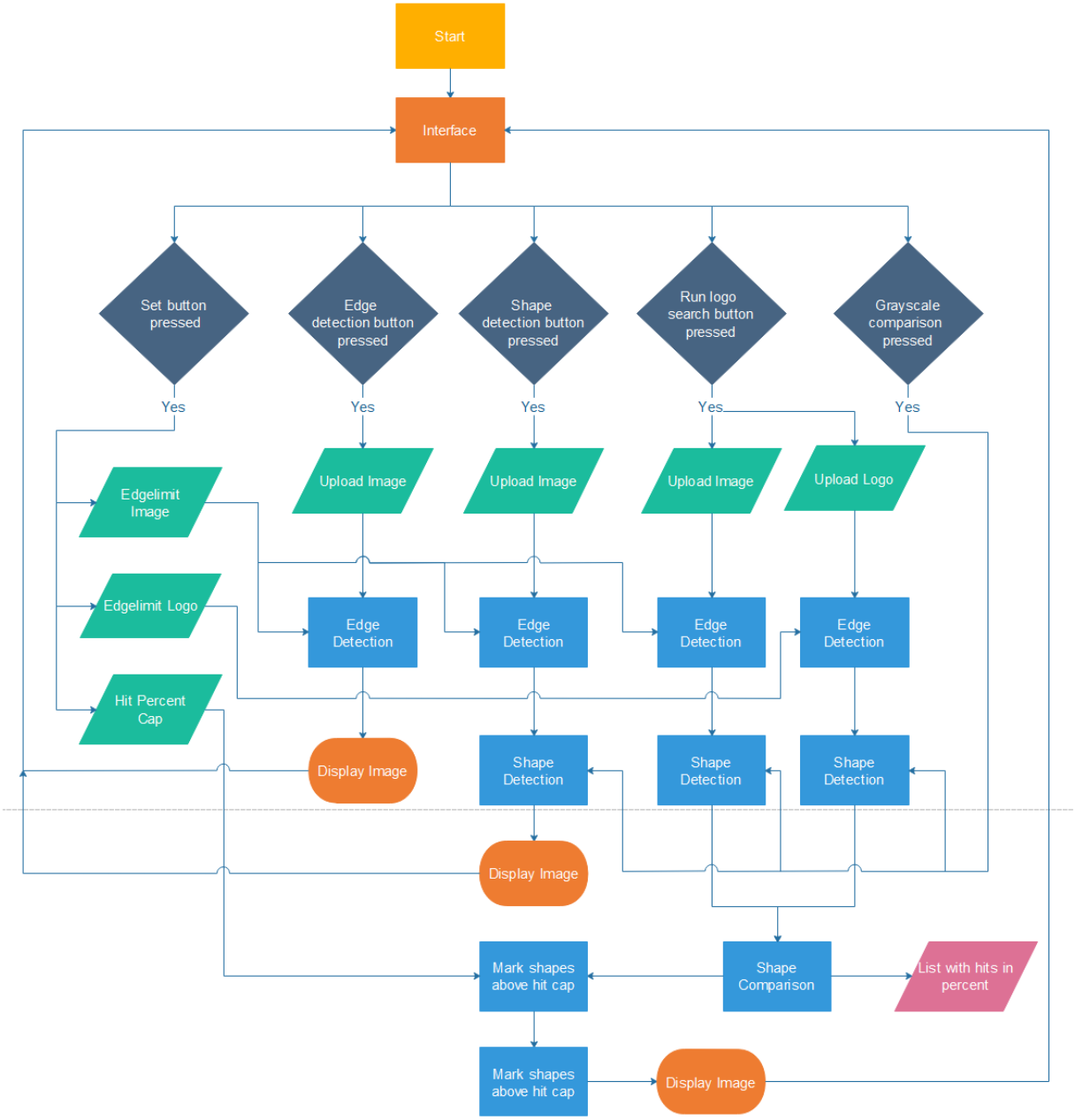

Figure 2: Program flowchart. Gray rhombuses are buttons, green parallelograms are inputs, blue rectangles processes, magenta parallelogram is output and orange ellipses are displaying of image result.

Images

Once the program is started, the user gets the opportunity to choose what process to run. Each of these processes requires the user to upload an image. If the user does not choose an image, a test image will be used instead. Both edge and shape - detection only need one image. These runs are used by the user to display the set edge limit and visualize what impact it has on the result. The comparison run needs two images, a subject and a logo.

After an image is imported to the program, it is converted to *Bitmap* image, with pixel format 32bppArgb. This format has 32 bits per pixel, 8 bits for each alpha[2], red, green and blue components. This means that each component can have 256 values since $2^8 = 256$, (0 – 255). In the code, this is again converted to byte arrays to faster read and write data from and to images. When the data are stored in these arrays it is read through BGRA, meaning every pixel has four bytes, with blue component first then green, red and alpha. When looping these arrays based on *points*, the program calculates the corresponding array index to the X and Y of each point. This value is called *byteoffset* and is given by equation (1).

$$byteoffset = (y * imagewidth + x) * 4 \quad (1)$$

Edge detection by convolution

To be able to define shapes in the images, a Sobel operator is convolved[3] with the given image. This makes it possible to detect edges in the images. The definition used for a *gradient magnitude* is; a change in RGB-color values between a pixel and the adjacent pixels. The higher the change, the higher the gradient magnitude equation (2). In this program, the red, green and blue color values are the same when the gradient magnitude is calculated. This is because the image is converted to *grayscale* [4]in the edge detection code, to not differentiate between colors. Not converting to grayscale would lead to three different values for the gradient magnitude per pixel. Therefore, the grayscale comparison checkbox, see Figure 2, is made to have no effect on edge detection code.

$$|G| = \sqrt{|Gx^2| + |Gy^2|} \quad (2)$$

[2] Alpha is the transparency of the color; an illusion of transparency by showing the color behind.
[3] "Convolution provides a way of `multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality.
In an image processing context, one of the input arrays is normally just a graylevel image. The second array is usually much smaller, and is also two-dimensional (although it may be just a single pixel thick), and is known as the kernel." [2]
[4] A gray scaled image consists only of shades of gray. Red, green and blue components have the same value.

The Sobel operator consists of the two 3x3 convolution kernels; Gx and Gy (Figure 3).

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 3: Sobel convolution kernels [2]

These kernels are applied to every pixel by matrix multiplication and summing every entry in the resulting matrix. Below is an example of calculation of gradient magnitude for pixel P_5 from Figure 4.

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

Figure 4: Image matrix [3]

$$\left|G_y\right|(P_5) = (P_1 + 2{\times}P_2 + P_3) + (-P_7 - 2{\times}P_8 - P_9)$$

$$\left|G_y\right|(P_5) = (P_1 + 2{\times}P_2 + P_3) + (-P_7 - 2{\times}P_8 - P_9)$$

$$|G|(P_5) = \sqrt{|G_x^2| + \left|G_y^2\right|}$$

Because the kernel size is 3x3, it is not possible to apply the operator on border pixels. To solve this the program does not check border pixels for edges. The same would apply for a kernel of 5x5 or 7x7, but the border would have had a width of 2 and 3 respectively. If the edge detection button is pressed to run this part of the code, the analyzed image is displayed in the interface.

Defining shapes

The code running the edge detection returns a list of points that are defined as edges. In this program, every point with a gradient magnitude equal to or above the set *edge limit* is considered an edge point. The edge limit is chosen by the user and should be chosen by trial and error using the edge detection button to find a value that gives a satisfying result. The lower the edge limit the longer the program takes to finish. It should be noted that setting a higher value, makes the code ignore more data from the image. It is therefore important to check the value with the edge and shape -detection part of the program to make sure that shapes are defined properly.

The program then checks each of the points in the list for adjacent edge points. If a point from the list is connected to another point, they are saved in a new list as a shape. The process of defining shapes is repeated until there are no more shapes in the image. Any shape that contain less than 10 pixels is not be counted as a shape. It then makes a point from the min X and Y from all the points in the shape, same is done for max X and Y. These two points make up the rectangle around the shape, marking the area of interest. These two points are saved in a list of points. This prosses is done for both the image and logo, one list for each. If the shape detection button is pressed to run this part of the code, the areas of interests are marked by a deep pink color frame (255, 20, 147), and the image is then displayed in the interface.

Comparing shapes

The comparing part of the program takes the two lists of points, making up the shapes from each the subject and logo, and compares each area of interest from the logo up against each area of interest from the subject by subtraction. Since these areas rarely have the same dimensions, they must be transformed. If the area from the logo is 30x40 pixels, the subject is transformed into the same dimension regardless of the original aspect ratio. This is not a problem since the logo is in its original aspect ratio, seen from directly front. For example, if the area of interest from the logo is a square and the area in the subject is deformed, the transformation will make it a square. Therefore, if it is the same square it will result in a match. In addition, if the area is both a square in the logo and the subject, the area from the subject will be compared as a square. This only applies the areas deformed in the width and height everywhere, stretched. If a shape in an area of interest is rotated or deformed in another way, it will not result in a match. This is called perspective correction and is much more complicated than scaling in X and Y-axis, but in this project, it is ignored simply due to the lack of time to implement it. One thing to note is that the transformation is done by *nearest neighbor* scaling. Meaning that when an image is made bigger the image does not have an increase in information, it just gets more pixels. (Figure 5)

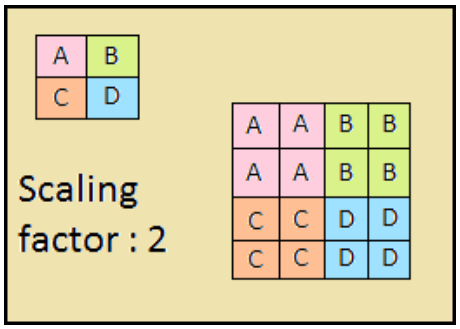

Figure 5: Nearest neighbor scaling (5)

When the areas are compared by subtraction, each red, green and blue - value of each pixel in the subject is subtracted the red, green and blue value from the logo. Equation (3) is for the red component, but it is the same for the green and blue component.

$$dr = \frac{\sum_{n=1}^{N} |r_{sub}(n) - r_{logo}(n)|}{N} \quad (3)$$

All these differences from each color, are summed and dived by number of pixels *N*. The result difference between the logo and subject, called *match factor*, is calculated by equation (4). Match factor is a value between 0 and 1.

$$match\ factor = 1 - \frac{(dr+dg+db)}{3} \quad (4)$$

Each time an area from the subject has gotten a match factor, the program checks if the same area has gotten a higher match factor in an earlier comparison. If not, the match factor is stored in a list of match factors, called *hits*, containing the highest match factor for that particular area. The hits appear in the order the program reads though the areas of interests. First X-axis then Y-Axis. For example, the scaled squares in Figure 5 would be read in the order; AABBAABBCCDDCCDD. The list is then displayed in the interface. The areas of interests are then marked by color if the hit is above the user set hit cap. The analyzed subject is then displayed in the interface.

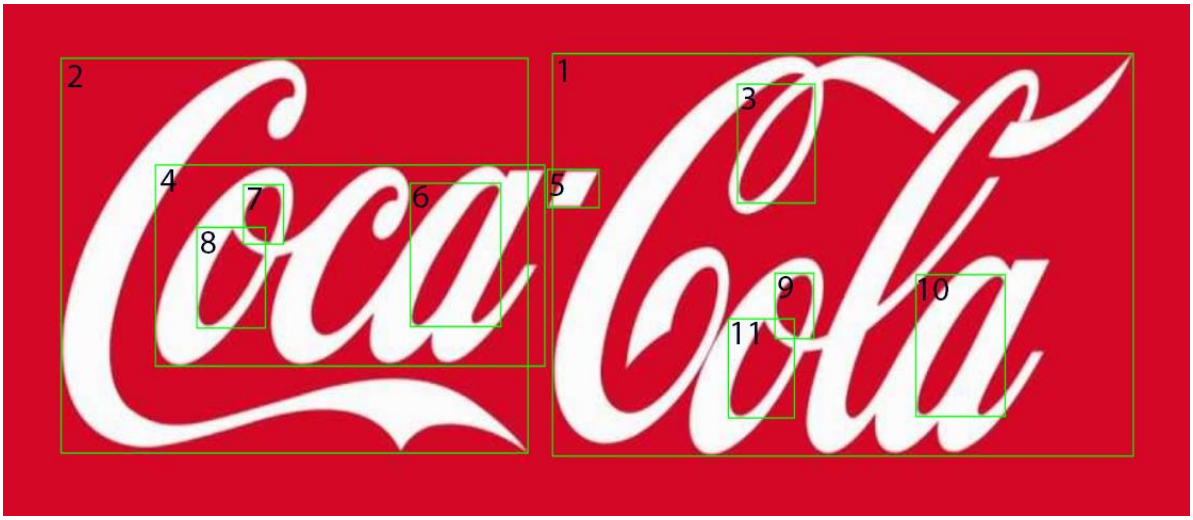

Figure 6: Test run for the comparison with edge limit at 100 for image (subject) and logo. Numbers are added to illustrate where the different areas are located.

Table 1: Table of hits for each area of interest in test run comparison (Figure 6)

Area of interest	Hit	Area of interest	Hit
1	100%	7	100%
2	100%	8	100%
3	100%	9	100%
4	100%	10	100%
5	100%	11	100%
6	100%		100%

Results

The results from the edge detection of a coca cola logo are shown in Figure 8.

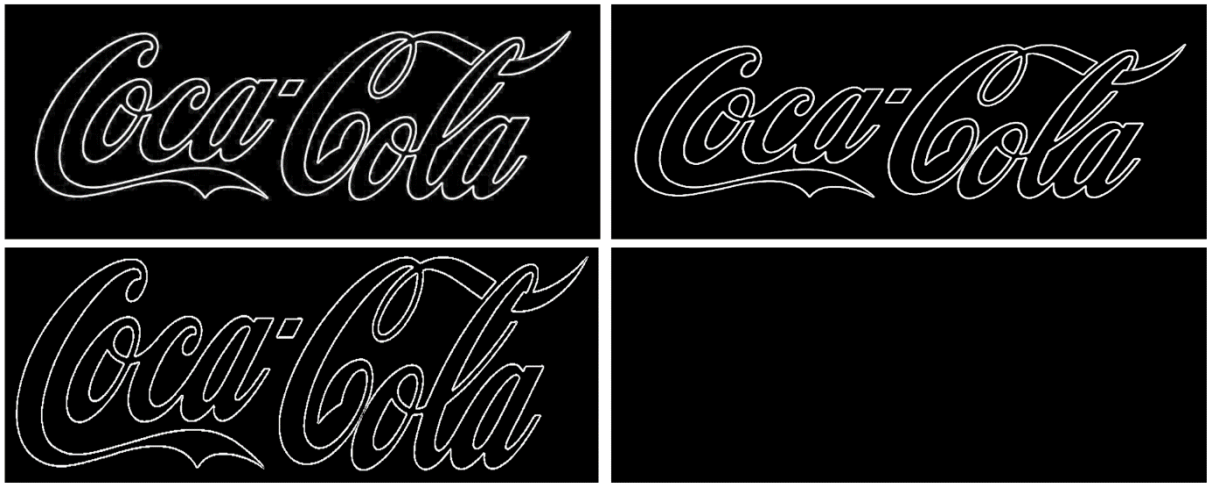

Figure 7: Edge detection with edge limit of 5 in upper left, 100 in upper right, 255 in lower left and 256 in lower right. Logo from Figure 6.

The results from the shape detection of a coca cola logo are shown in Figure 8.

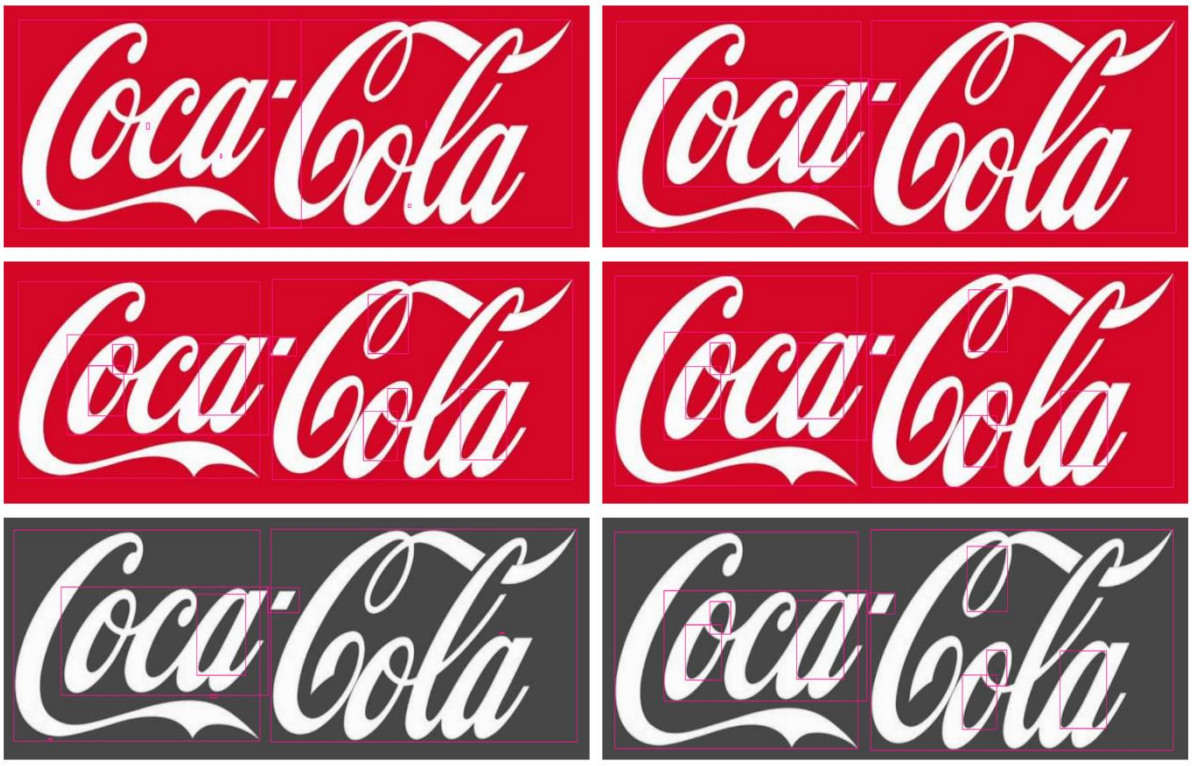

Figure 8: Shape detection with edge limit of 5 in upper left, 30 in upper right, 100 in middle left, 255 in middle right, 30 in lower left and 100 in lower right. Both lower left and right is done in grayscale by checking the grayscale checkbox (see Figure 2). Logo from Figure 6

Figure 9: Logo search on fridge with an edge limit of 120 on the image (subject) and 100 on the logo. Same logo as in Figure 6.

Table 2: Table of hits for each area of interest in comparison (figure 9)

Area of interest	Hit	Area of interest	Hit
1	74,49%	10	72,64%
2	77,54%	11	72,29%
3	71,11%	12	71,15%
4	70,16%	13	69,03%
5	76,80%	14	69,64%
6	70,03%	15	73,31%
7	69,91%	16	72,93%
8	70,17%	17	72,03%
9	70,86%	18	71,03%

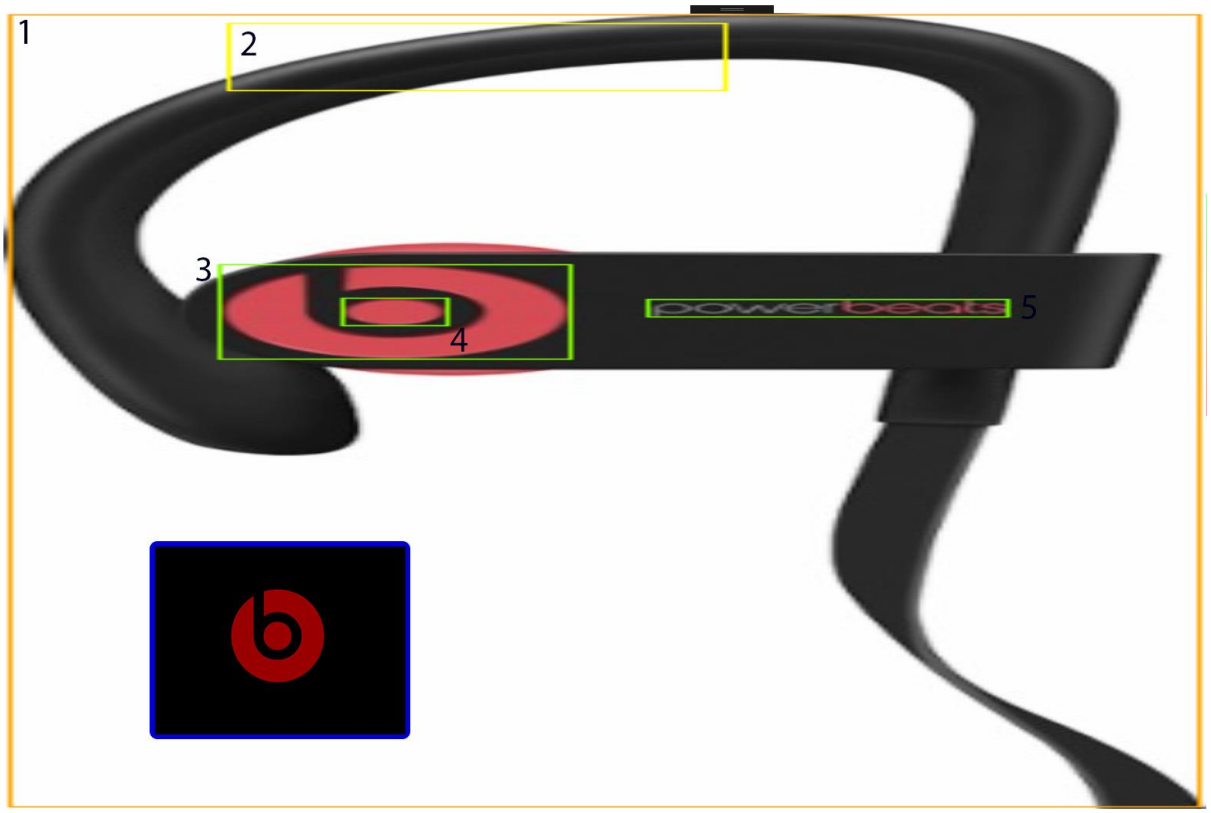

Figure 10: Logo search on earpiece with an edge limit of 100 on the image (subject) and 100 on the logo. Logo used for comparison is the inserted with a blue border, this means it was not analyzed with the rest of the image.

Table 3: Table of hits for each area of interest in comparison (figure 10)

Area of interest	Hit
1	32,68%
2	49,05%
3	73,61%
4	74,52%
5	77,38%

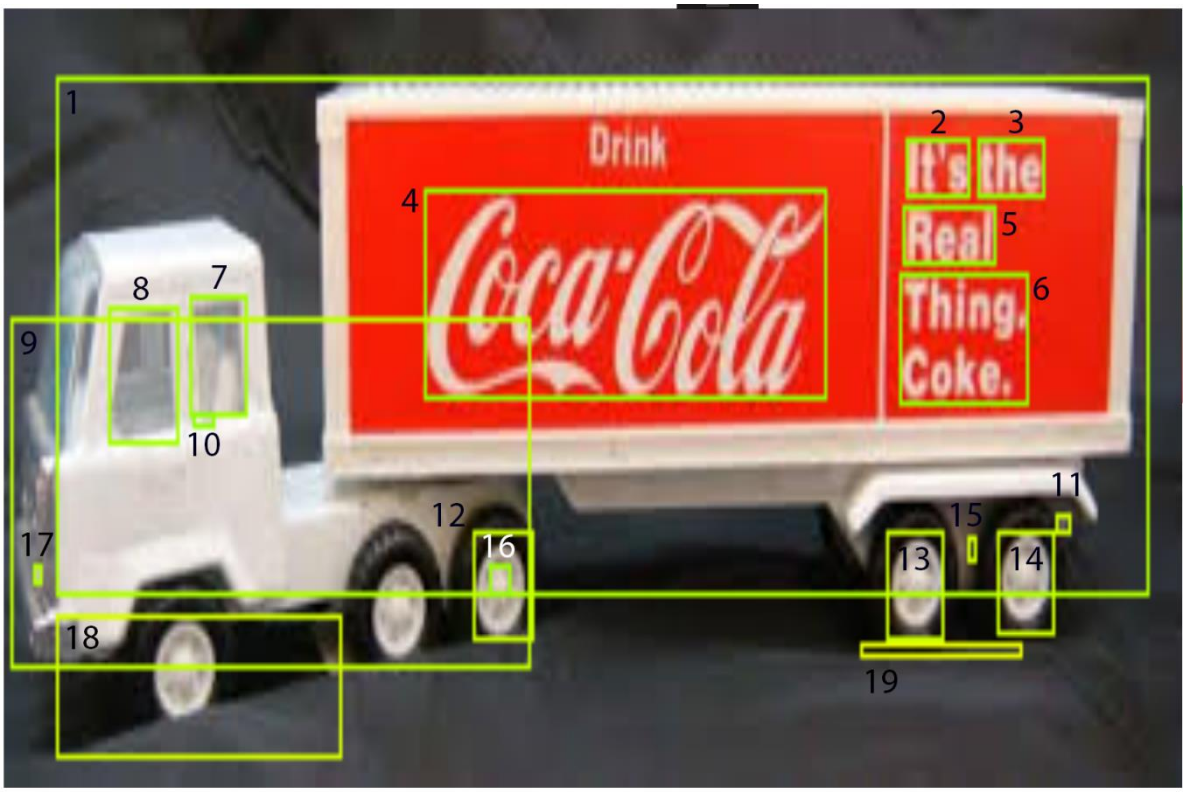

Figure 11: Logo search on a toy truck with an edge limit of 150 on the image (subject), and 100 on the logo. Logo from Figure 6.

Figure 12: Same logo search as in figure 11, but with hit cap of 70%

Table 4: Table of hits for each area of interest in comparison (figure 11)

Area of interest	Hit
1	65,46%
2	71,35%
3	72,20%
4	71,68%
5	72,15%
6	71,22%
7	68,26%
8	67,25%
9	62,13%
10	71,58%
11	59,43%
12	60,30%
13	60,40%
14	59,97%
15	58,96%
16	65,98%
17	69,94%
18	57,87%
19	53,79%

The results from running the edge detection algorithm with different edge limits on a Coca Cola logo are shown in Figure 7.

The results from running the shape detection algorithm with different edge limits on the same Coca Cola logo are shown in Figure 8.

First comparison or “logo search” run was to test if the program was calculating the match factor correctly (see Figure 6). Since all the areas of interests had a match factor of 100%, it can be said it is working correctly.

Figure 9 shows the results of a comparison, known to include a Coca-Cola logo. This was less accurate and gave an impression that the areas of interests in the Coca-Cola logo are hard to separate from the other areas in the subject.

Next, a new comparison has been run on an earpiece by beats (see Figure 10), with another logo (Figure 10). There is a higher hit at the word “powerbeats” than on the logo.

Figure 11 shows a logo search on a toy truck. To illustrate difference in hits, the hit cap was set to 70% and run search was ran again with all other inputs remaining the same (Figure 12).

Conclusion

The edge detection part of the program runs fast and gives satisfying results. It is clear from the results that increasing the edge limit removes data and therefore also information from the image. The image logo used here (Figures 6 to 10) is of too good quality to ever get to a point where increasing the edge limit removes too much data for the edges to no longer give a recognizable logo.

The shape detection is slower than the edge detection, which is not surprising considering the shape detection runs the edge detection code as a part of the process. However, since it only runs it once and the time it takes to complete the shape detection is more than doubled, it is fair to say that the shape detection takes longer time. Increasing the edge limit did in this case make for a more accurate shape detection. As can be seen in Figures 12 and 13, the increase in edge limit makes the program detect more non-adjacent shapes. In this case the image never reaches the point where too much data has been ignored by the edge limit for it to split an adjacent shape into two or more areas of interests. But in other cases that very well may happen.

From the results of the logo searches one can see that the program is not a 100% accurate. It looks like the colors in the shapes are more important for the match factor, then the form of the shape. This can perhaps be improved by comparing the angles of the edges instead of the color values, which can be calculated by the gradient magnitude. Alternatively, it can be done by just comparing the gray-scaled edge detection results instead of the original images. However, that would lead to a significant loss of information. The program is able to detect areas that are similar in color and gives an indicator to where to look for the logo. This is shown by the difference in Figures 21 and 22.

There should be a save feature that makes the user able to save both image and hit list locally. Also, the program does not say if the logo is in the image or not, it only displays where it can be. The user will have to do an effort to see what match factor belongs to what area of interest, which definitely can be improved.

Trademarks

References

[1] Microsoft. What is XAML? [internet]. [accessed 2017-11-30]. Available from: https://msdn.microsoft.com/en-us/library/cc295302.aspx

[2] Fisher R., Perkins S., Walker A., Wolfart E. Convolution [internet]. University of Edinburgh: 2003 [accessed 2017-11-27]. Available from: http://homepages.inf.ed.ac.uk/rbf/HIPR2/convolve.htm

[3] Fisher R., Perkins S., Walker A., Wolfart E. Sobel edge detector [internet]. University of Edinburgh: 2003 [accessed 2017-11-27]. Available from: http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

[4] Esterhuizen D. C# How to: Image Edge Detection [internet]. 2013 [accessed 2017-11-27]. Available from: https://softwarebydefault.com/2013/05/11/image-edge-detection/

[5] Rajakaruna A. Nearest-Neighbor [internet]. 2013 [downloaded 2017-11-27]. Available from: https://aditharajakaruna.files.wordpress.com/2013/07/nn1.png

[6] Item: Vintage Coca Cola Tractor Trailer Toy Truck [internet]. [downloaded 2017-11-30]. Available from: https://www.icollect247.com/itempics/312_1475027356A.jpg