Exploring metrics for Aesthetic Evaluation of Web-pages layout

Filip Sotirov

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

December 6th, 2019

Abstract

The purpose of this project is to evaluate the aesthetic appeal of web-pages by exploring metrics like equilibrium, density, and simplicity. They will be used to match the perception for better aesthetics. The program will be called MC, coming from Metric Calculator for future reference.

With further integration of all other metrics into this program, the system could be used by web-developers/designers to better assess the website design prior writing the code for a web-page, or to do evaluation of already built websites. This type of evaluation could be useful also to build better templates for platforms like Wix, and Squarespace.

Table of Contents

Chapter 1 Introduction

There have been many theories and methods developed for the design of interfaces, focusing on the usability of the products in the past. With the development of the digital devices and the Internet more and more people started using them. This created crucial necessity of learning more about the usability of digital devices. Over time it is noticed that there have been an increasing recognition of the visual aesthetics in interface design. Studies started acknowledging the importance of system beauty. [11]

There are a few approaches for evaluating the aesthetic appeal of a web-page. The currently designed program, MC, uses a screenshot of a web page as an input data. Then the image goes through different processes in order elements(sections) from the image to be extracted. These elements are then passed to an algorithm that calculates metrics, which output values are the aesthetic measure.

In ideal conditions the program would calculate the metrics correctly. The main challenge comes from detecting the correct elements from images. The results coming from processing an image are not the outcome needed to calculate the 'real' metric values.

Chapter 2 Background

2.1 Digital Image Processing

The earliest use of computers was to do numerical operations only. Numbers were fed in, then the computer performed calculations, having an output - another number. The use of computers has evolved since then and some new roles embed the computer in systems where human is rarely involved such as in control systems in autopilots and other controllers, speech recognition, medical diagnosis, image processing and many more. Advances in image processing have been rapid since the first use, and extremely useful. The processes involved provide extension to the human's senses and the aspiration for aesthetic design.[13]

2.2 Aesthetics and preferences of web-pages

Screens should display information clearly as in most cases this information is the way through which users interact with the system. Achieving this is a complex matter where graphical design as well as psychological understanding is involved. The key of designing effective screen displays is knowing the user tasks and competences. Example studies for alphanumeric screens analyses are Tullis' Display Analysis Program in 1988, Streveler and Wasserman's in 1984, these are some of the early examples. Tullis' explored links between a few metrics - screen density(local and overall one), objects grouping etc. The time to extract information from screen was also explored. After experiments, he developed equations predicting preference ratings and search times. Streveler and Wasserman developed a system for 'quantitatively assessing screen formats'. 'They also developed techniques for screen format analysis: "boxing", "hot-spot", and "alignment" analysis.' Many of the studies conducted after Tullis' were concentrated mostly on alphanumeric screens, and they were focused predominantly on the usability of the display. [12]

There is a study led by Ngo, Teo and Byrne(2000) that focused on the perception of structure on screen. The system measure similar qualities to the once done by Tullis and others(although using different techniques), but the difference is that the values are used to measure the aesthetics of the screen, not the usability. Until then there were no formally codified guide of how to create attractive screens, and the theory about designs was not well developed. Ngo and Byrne(2001) developed fourteen crucial characteristics of aesthetically pleasing objects – metrics.[12]

There are many studies based on the work of Ngo and Byrne, one of which is conducted by Purchase, H. and Hamer, J.[11] which investigates whether the use of this metrics can help in producing aesthetically pleasing web pages. This paper extends the use of Ngo measures by using web pages, a much richer type of interface than those used in Ngo's studies. Perceived usability is also considered, as well as the effect of color. The input for the calculations have been collected from the html code of the of web-pages. Moreover it has been concluded that quantifying the aesthetic appeal is possible via these metrics, and that it would be very useful for web designers and developers throughout the designing stage of building a website.[11] I will use images from their data set as an input data in MC, which images are screenshots of the web-pages used in their study.

Metrics developed by Ngo and Byrne(2001) are described below. More about the metrics and all formulae can be found here[12]

Metrics – brief description[12]

- Balance – that is the distribution of optical weight in the picture. Large objects are considered to be heavier than the smaller objects on screen.
- Equilibrium – a picture is having good equilibrium measure when the center of the layout correspond somewhat to the center of the frame. This measure is calculated as the difference between the mass center of all elements on screen and the center of the screen.
- Symmetry – that is the extent to which the elements on screen are symmetrical in horizontal, vertical, and diagonal manner.
- Sequence – this refers to how the elements are arranged on screen, and to achieve better sequence the screen should start with bigger objects while moving to smaller.
- Cohesion – a good cohesion is achieved in images when the aspect ratios of objects stay the same when scanning the screen.
- Unity – this metric is calculated to give the extent to which the elements on the screen belong together. This is achieved by giving less space between the elements than the margin space.
- Proportion – this metric deals with the size relationship of one shape compared to another.
- Simplicity – better simplicity can be achieved by reducing the number of alignment points on screen. It counts all rows and columns that are used as starting point for objects on screen.
- Density – it calculates how much of screen space is occupied by objects. The optimum percentage is 50%.
- Regularity – this is achieved by organizing standards for the space between the horizontal and vertical alignment points of the objects, and by reducing the number of the alignment points.
- Economy – good economy can be achieved by using less sizes for the objects and by grouping the similar size objects together.
- Homogeneity – this is the measure of how evenly the objects are distributed in all quadrants of the screen.
- Rhythm – this is achieved by systematic ordering of objects on screen. This metric depends on the dimensions, number, type of arrangement of objects etc.
- Order and Complexity – This measure is the aggregate of all the previous described measures. It gives an overall view of the complexity of the layout of an image.

It is suggested that some improvements to the metrics can be made to enhance its usability. In Balance for example the scope can be broaden by adding the tone, shape, and color of the objects(e.g. black color is visually heavier than white color, big size objects are visually heavier than smaller once).[12]

2.3 External Libraries

There are a many external libraries that can help to 'speed up' the process of developing any program by providing large set of tested algorithms. Some of the libraries involving image processing are – OpenCV, BoofCV, Deeplearning4j, JavaCV, AlgART.[1]

- OpenCV is widespread computer vision and machine learning library applied in great variety of contexts. What makes this library powerful is the provided huge amount of both classic and up to date computer vision algorithms(over 2500), and not surprisingly because it launched way back in 1999. They are supplied for: image processing, feature detection, object detection, machine-learning, and video analysis. [2] OpenCV is written in C++, and has primary interfaces for C++, Python and Java, and is also expanding to cover many more languages like C#, Perl, Ch, Haskell and Ruby. That is making it very diverse.

- BoofCV is an open-source library licensed under Apache 2.0 license for real-time computer vision and robotics applications. It can be used for low-level image processing routines such as convultion and interpolation to high-level functionality such as image stabilization.
- Deeplearning4j is a complete set of packages and libraries for deep learning in Java. The library also has APIs for other languages like Scala, Python, Clojure and Kotlin.
- JavaCV is not a traditional library but it is a wrapper for popular CV packages. It wraps up under its hood many libraries like OpenCV, FFmpeg, Leptonica and many more. Such libraries offer features of the base libraries adding to it improved usability and some additional features.
- AlgART is an open-source library for array-based computations and image processing. The algorithms are designed for any number of matrix dimensions, allowing users to easily deal with 2D, 3D or other multidimensional image processing.

From the above mentioned libraries I choose to work with openCV(version 4.1.1) as it's been around for a long time, and it provides plenty of proven algorithms for processing and manipulating image information.

Chapter 3 System Implementation

To disperse the potential confusion between the use of the terms ‘method’ and ‘function’: as for the purpose of this project I am going to work with Java programming language, thus I am going to use only the term ‘method’ in this paper, although in some of the used sources, ‘function’ is the referred term.

This chapter provides a description of the implementation and the operations that take place during execution of the program, MC. Moreover the decision making process on why one solution is chosen over another, while building the program, will be also portrayed.

The system structure of MC can be seen on **Fig. 1** below. The structure is split into different sections, and each section is labeled with a letter from ‘A’ to ‘F’. What processes go in each section will be explained separately.

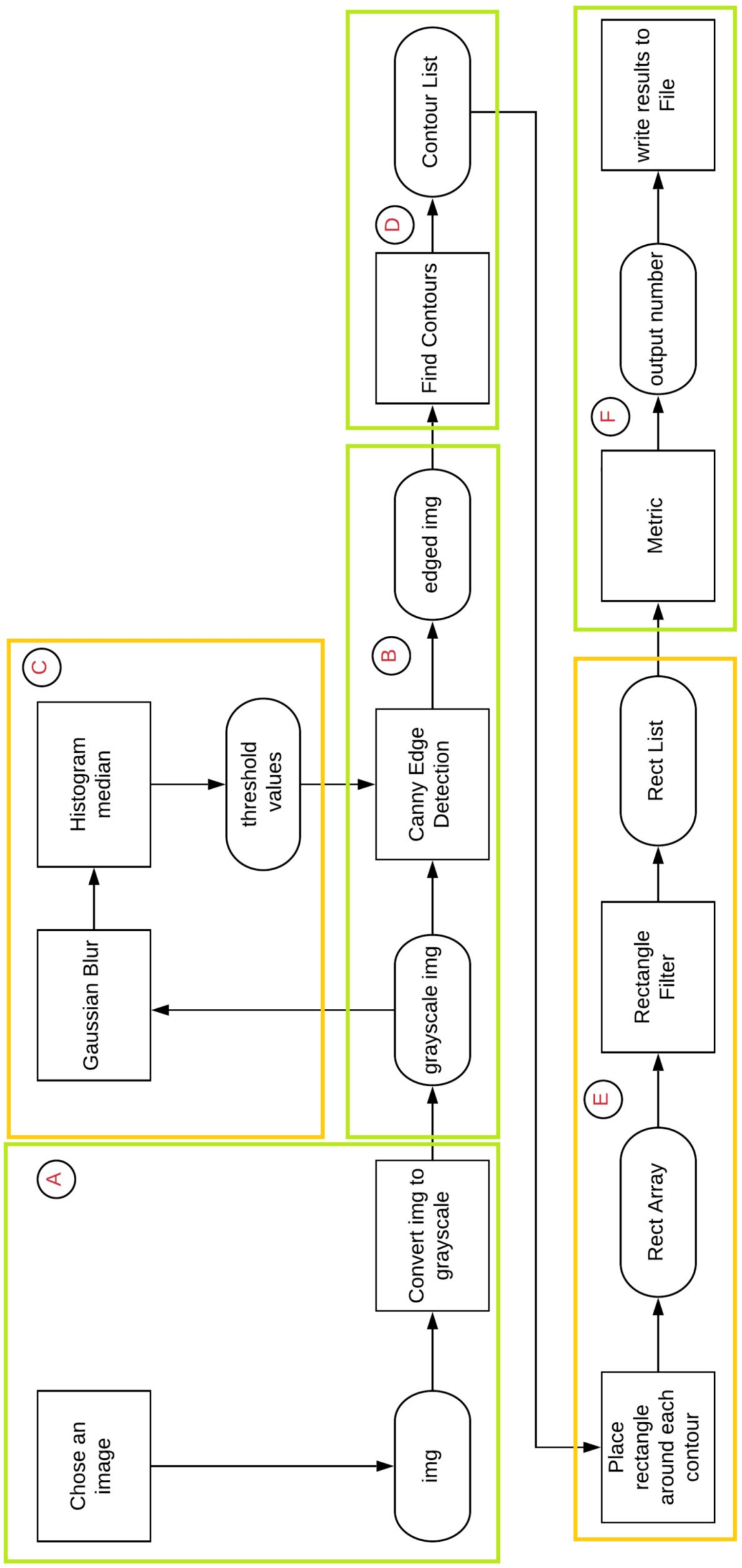


Fig. 1 System structure

<u>Section 'A':</u> Input

This section allows the user to choose the input to the system. The input is an image file or a directory, containing one or more image files. All extensions of the image files are accepted. This image(s) is a screenshot of a web-page, which is a colored image(s). In the selected data set there are seventeen images, three of them will be tested.

<u>Images – digital representation, reading, storing, manipulate image characteristics</u>

There are many ways how to acquire digital images from the world around us – cameras, computed tomography, scanners etc. In all cases the human eye sees images. However, the digital devices are designed to detect and to record images as numerical values for each point of the image. Images are represented as a matrix, which consists of all the intensity values of each pixel point. According to our requirements, how we take and how we keep these pixel values may differ, but in general in the digital world this is reduced to numerical matrices and other information that describes these matrices. [4]

In OpenCV images are stored as Mat objects. Mat is simply a class with two data parts – header and pointer. The header contains information such as the matrix size, at what address is the matrix stored, method used to store the matrix and so on. The pointer leads to the matrix that contains the pixel values. Although this matrix can be multidimensional, depending on the method chosen for storing, it will be viewed as 2D matrix in the current case. A few words about the storing methods – the color space and the used data type can be chosen when defining a Mat object. The color space is the combination of color components in order to code a given color. The simplest way for example is to store the image as gray scale, which is the combination between the black and white colors(that gives many shades of gray). For storing colorful images there are a lot more methods to choose from. Each one breaks it down to three or four basic components, the combination of which create the others. The most popular one is RGB, because that is the way how the human eye builds up color. The base colors are red, green, and blue. Sometimes a fourth element is given – transparency. In OpenCV the display system is BGR instead of RGB. [4]

In section 'A' once the colored image(s) is selected, it is stored as Mat object. A conversion to gray scale image is done simply by using a method from OpenCV. The input as well as the output of the method are Mat objects, the only difference is that the output is a gray scale image.

<u>Section 'B':</u> Canny edge detection

Once the colored image is converted, the program can proceed to detecting all edges from the gray scale image. For that purpose Canny edge detection algorithm is used.

The algorithm is popular edge detection methodology and it was developed by John Canny in 1986. It is multi-stage algorithm, and each of the stages will be described below.
The Canny edge detection algorithm is generally used in computer vision to locate sharp intensity changes and to find object boundaries in an image. The Canny edge detector classifies a pixel as an edge if the gradient magnitude of the pixel is larger than those of the pixels at both its sides in the direction of the maximum intensity change.[5]
This algorithm consist of a few steps(stages)[6]:

- Step 1 – Gaussian Filter. In order to implement this algorithm series of steps must be followed. The first step is to filter out any noise in the original image before trying to locate any edges. Gaussian filter is used here. This filter can be computed using a simple mask. The convolution mask which is much smaller than the image can glide over the image manipulating a square of pixels at a time.

- Step 2 – After the noise has been eliminated, the next step is to find the edge strength by taking the gradient of the image. A 2-D spatial gradient measurement is performed by the Sobel operator(it used a pair of 3x3 convolution mask), then the edge strength at each point can be found.
- Step 3 – Finding the edge direction. This becomes trivial once the gradient in the 'x' and 'y' directions are known.
- Step 4 – Relate the edge direction to a direction that can be traced in an image
- Step 5 – Apply non maximum supervision. After the edge directions are known, non-maximum suppression now has to be applied – it is used to trace along the edge in the edge direction and suppress the pixel value that is not considered to be an edge. This will give thin line in the output image.
- Step 6 – Hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating below and above the threshold. Hysteresis uses two thresholds, a high and a low one(see on **Fig. 2** below). Any pixel in the image that has a value above the high threshold is presumed to be an edge pixel(edge A), every pixel with value below the low threshold is disregarded, and the pixels in between the two thresholds are disregarded too if there is no continuity with pixels from above the high threshold(edge B). Otherwise they are considered part of the edge(edge C).

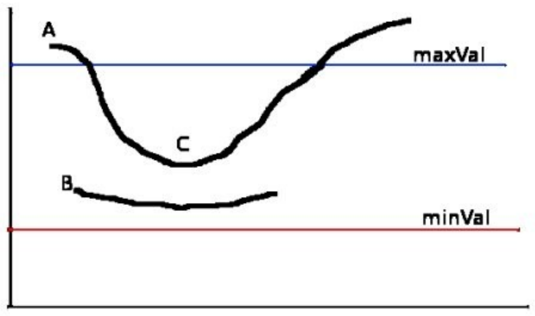


Fig. 2 Histeresis[7]

After familiarizing with the Canny algorithm it is time to be applied in MC. Among many other useful methods in OpenCV there is a method called 'Canny' that is used to find edges from a gray scale image. The input for this method is a gray scale image and the output is an edged image(white pixels on a black background). The only arguments that needs to be provided are the values for the low and the high thresholds, and for the rest the default provided values are set. How the thresholds values were determined is described in the next section. On **Fig. 3** below an image after Canny edge detection is performed can be seen. This image is chosen from the data set and it is called 'borders.bmp'.

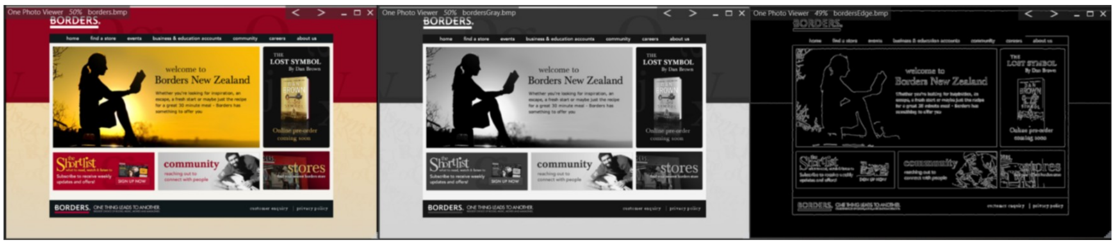


Fig. 3 Source image(left), gray scale(middle), edged image(right)

<u>Section 'C':</u> Thresholds

Determining the thresholds - these values are not defined in the Canny method, so they need to be provided. There are many methodologies about how to determine these values, one of which is through finding the Histogram median. An example done in python is followed while adjusting it to java. This example will be called 'PE', and it is described in a few steps below:[8]

- Gaussian Blur. There is a method from OpenCV that does Gaussian blur in order to reduce the noise, where the default values for the arguments are set, and the kernel size is set to 3x3.
- Calculate the histogram. Histograms are collected counts of data organized into set of predefined bins. This data could be any kind of data that suit your needs, in the current case these values will be the pixel intensity values, the range of which is [0, 255] inclusive, these are the values in a gray scale image that every pixel can take. Then the bin size should be specified, this is if there is a need for further segmentation of the data to smaller ranges. The number of bins is chosen in MC is the maximum possible – 256. This way the calculated median is the actual median value of the histogram. Again predefined method from the library is used to calculate the histogram, while specifying arguments like type of the tested image, number of channels(one channel in our case as our input is gray scale image; for colored images there are three channels, for each of the colors), number of bins, range, and mask.
- Get the median from histogram. Once the histogram is known, an algorithm is written to get the median value. A point should be made here that this value is different from the AVG value of the histogram. Then the low and high threshold are set as shown on **Fig. 4** below.

```
    double sigma = 0.33;
    int lowThresh = (int) (Math.max(0, (1 - sigma) * histMedian));
    int   highThresh   =   (int)   (Math.min(255,   (1   +   sigma)    *
histMedian));
```

Fig. 4 Low and High thresholds[8]

Sigma is a coefficient that can be set in the beginning if there is a need to. In 'PE', and in general sigma being 33% shows good results – that is why it is set the same way in MC. Setting it higher will mean higher thresholds values and tighter gap between the two thresholds and vise versa.

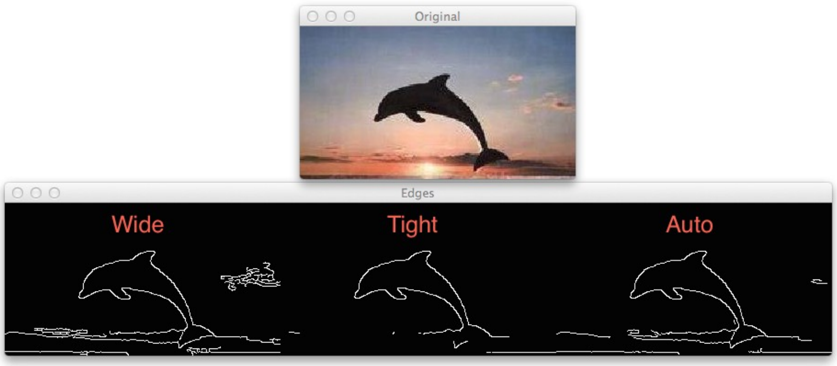


Fig. 5 ‘PE’ results[8]

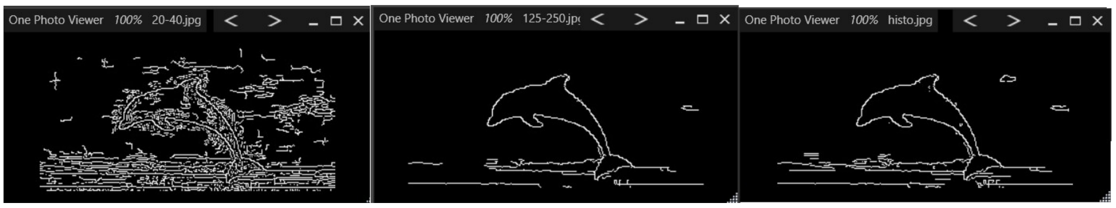

One Photo Viewer 100% 20-40.jpg
One Photo Viewer 100% 125-250.jpg
One Photo Viewer 100% histo.jpg

Fig. 6 Difference between fixed(left and middle images) and auto threshold(on the right)

After carrying out testing, with images taken from ‘PE’, very similar results are achieved. See on **Fig. 5**. and **Fig. 6** above. It is noticeable that when fixed values are given the results are not as they should be. On **Fig. 6**(left) the thresholds are set low, where the low thresh is set to 20, and the high thresh is set to 40. Clearly it is noticeable that there is too much noise and too many edges detected. The image in the middle(**Fig. 6**) is with low thresh = 125, high thresh = 250. Here most of the background is missing(not detecting the clouds). This is where the automatic detection comes in place. The image on the right(**Fig. 6**) is with automatic threshold calculation(histogram median).

<u>Section ‘D’:</u> Contours

The next step after the edges are detected is to find the contours. Contours can simply be described as a curve joining all the continuous points(each with ‘x’ and ‘y’ coordinate), having some color or intensity. They are a useful tool for shape analysis and object detection. For better accuracy, it is suggested that binary images should be used. That is why Canny edge detection was performed prior. In OpenCV finding contours is like finding white objects on black background. Each individual contour is an array of coordinates of the boundary points of the object.[9]

In MC a method from the library is used to find the contours from an image called ‘findContours’. The input of this method is the output of the Canny edge detection method. The contours are stored as vector of points. When using the ‘findContours’ method it is required that the <u>retrieval mode</u> must be specified, as well as the <u>contour approximation method</u>.

Retrieval mode options:

- retrieves only the outer contours
- retrieves all the contours without establishing any hierarchy
- retrieves all of the contours and organizes them into a two-level hierarchy
- retrieves all the contours with full hierarchy of nested contours

For MC is chosen the last option as all contours are retrieved and full tree hierarchy established, which could help for quicker search or for contour filtering.

Contour approximation methods:

- it stores absolutely all the contour points
- compresses all the horizontal, vertical and diagonal segments and leaves only their end points.

For MC is chosen the second option to save memory. For example when storing a line, it stores only the two end points that define that line, or if the object is a rectangle it stores only the four vertices, which represent that rectangle.

Section 'E' Bounding Rectangles

The retrieved contours from the previous section have a few features that can be explored.
Contour Approximation:
It approximates one contour to another with less number of vertices. An example of this would be if we are looking for a square in an image we wouldn't find it if it had 'bad' shape. This method would straighten up the sides of the square. There is a method from the library that performs contour approximation. The second argument is called epsilon, which is maximum distance from the contour to the approximated one. [10]
In the program epsilon is set to 10% of arc length. As lower this percentage is as closer the approximated contour will be to the original one. If set to 2% in some cases will mean that there is almost no difference between the original contour and the approximated contour.
Bounding Rectangle:[10]
Bounding rectangles are places around each contour after contour approximation. There are two types of bounding rectangles that could be used. Straight(upright) and rotated one – in the current case rotated rectangle is the better choice as this is the rectangle with the minimum area that goes around the contour(See **Fig. 7**). More likely the rotated rectangle would have greater angle of rotation relative to the axis when surrounds an object that does not represents upright rectangle object, not like the upright rectangle which is always upright no matter what object surrounds.

Rectangle Filtering:
Each rotated rectangle is specified by its center point(mass center), width, height, and angle of rotation. To exclude the tiny rectangles from the image, which might represent letters and other small objects the filter is set to extract rectangles larger than 20 by 20 pixels. After this there might be still rectangles that represent objects like people or similar(See **Fig. 7**) that are with large size, but more likely they have great angle of rotation. That is why the filter is set to extract rectangles with angle of rotation less than |2| degrees.

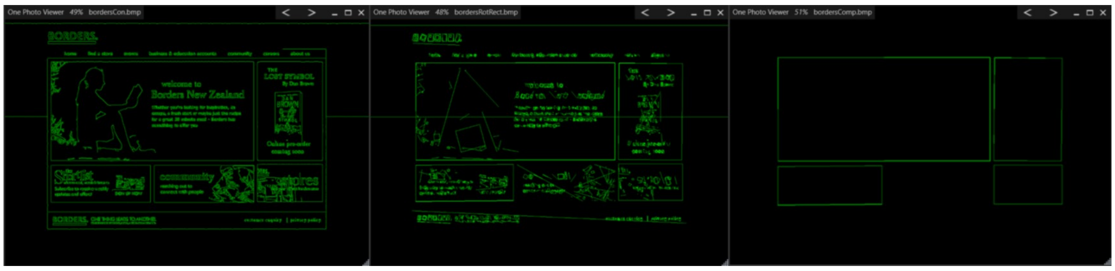


Fig. 7 Rotated rectangles(middle), and rectangles after filter applied(right) around contours(left)

<u>Section 'F':</u> Metrics

For every input image MC creates an object, which takes from the image its name, the frame width, and the frame height. After the image processing is done this object preserves the information about the filtered rectangles, while each rectangle carry information about itself – mass center coordinates, rectangle width, rectangle height, and angle of rotation. These rectangles are the representation of elements from the web-page like text fields, buttons, menu items, search boxes etc. Once the object holds that information the metric algorithms can explore it. This object will be called Pimage(from Processed image), for future reference.

Chapter 4 Metrics

The following three metrics are part of fourteen metrics developed by Ngo and Byrne(2001). More complete information about each one of them can be found here[12]. The output of all metrics is a number between zero and one. The following metrics are chosen for the calculations as they do not require information such as color, which Pimage does not currently hold. In the following metrics descriptions the word ‘object/s’ is mentioned several times. In MC these are the rotated rectangles from Pimage.

4.1 Measure of Equilibrium

Equilibrium is calculated as the difference between the center of mass of the displayed elements and the physical center of the screen. For example image four on **Fig. 8** has low equilibrium, wheres all other images are with high equilibrium value.

The formula for equilibrium(EM) is[7]:

$$\mathrm{EM} = 1 - \frac{|EM_x| + |EM_y|}{2}$$

The equilibrium components along the ‘x’ and ‘y’ axes accordingly are[7]:

$$EM_x = \frac{2\sum_i^n a_i(x_i - x_c)}{n b_{frame} \sum_i^n a_i} \qquad EM_y = \frac{2\sum_i^n a_i(y_i - y_c)}{n h_{frame} \sum_i^n a_i}$$

,where (x_i, y_i) are the coordinates of the center of the object ‘i’; n is the number of objects on frame; a_i is the area of the object; (x_c, y_c) are the coordinates of the center of the frame; b_{frame} is the frame width; and h_{frame} is the frame height.

4.2 Measure of Density

Density is the measure that shows to what extent the screen is covered with objects, and an example with low density value are images one and two on **Fig. 8**, and image three - ‘google.bmp’ is with high density value.

The formula through which the Density is calculated is[7]:

$$DM = 1 - \frac{\sum_i^n a_i}{a_{frame}}$$

,where a_i is the area of object ‘i’; a_{frame} is the area of the frame; and n is the number of objects on frame.

4.2 Measure of Simplicity

Better simplicity can be achieved by reducing the number of alignment points on screen(image three on **Fig. 8**). Simplicity measure counts all rows and columns that are used as starting point for objects on screen[12].
The Simplicity formula is[11]:

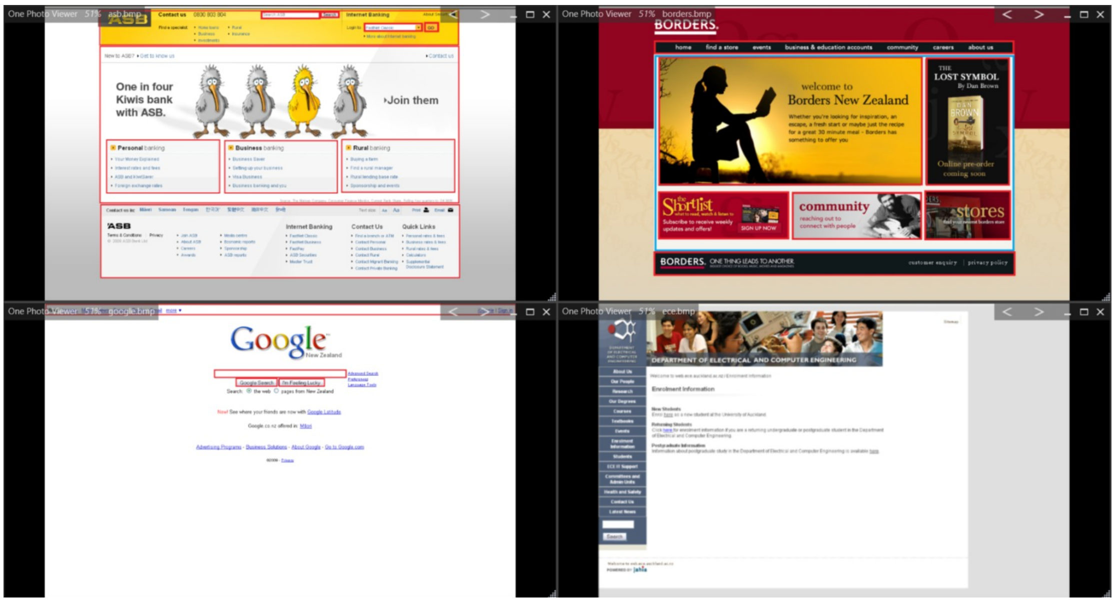
$$SMM = \frac{3}{n_{vap} + n_{hap} + n}$$

, where n_{vap} is the number of vertical alignment points; n_{hap} is the number of horizontal alignment points; and n is the number of objects on frame.

4.4 A comparison between manually calculated metrics and the program

The aim of this comparison is to better evaluate the correctness of MC. There are three images taken from the data set[12] which are used for the testing - ‘asb.bmp’, ‘borders.bmp’, and ‘google.bmp’. The calculated metrics are equilibrium, density, and simplicity.

Furthermore the manually attained results have the measuring tool limitations and are subjective to what sections are chosen for the calculations. More on what sections from the images are being used can be seen below.



One Photo Viewer 51% asb.bmp
One Photo Viewer 51% borders.bmp
One Photo Viewer 51% google.bmp
One Photo Viewer 51% ece.bmp

Fig. 8 Images 1, 2, and 3 used for manual calculations(sections marked in red and blue)[12]

The following table displays the manually calculated values, and the initial values calculated by MC. Although the metric values suppose to be a number between zero and one there are output results from MC like - ‘NaN’ and ‘infinity’. This will happen always when there are no objects(rectangles) passed to Pimage after filter applied(**See Fig. 1**). Following the formula for equilibrium from above Java is trying to do calculations like:

$EM_x = \frac{zero}{zero * zero}$, which it declares as ‘NaN’ - not a number.

Moreover for the simplicity calculations Java calculates the number three over a sum of zeros, which gives infinity as an outcome.

This issues will be addressed in the next chapter.

<u>Stage ‘0’</u>		‘asb.bmp’	‘borders.bmp’	‘google.bmp’
Equilibrium	Manual	0.9854	0.9942	0.9980
	MC	NaN	0.9907	NaN
Density	Manual	0.5800	0.4786	0.9493
	MC	1.0000	0.2454	1.0000
Simplicity	Manual	0.2307	0.2500	0.3750
	MC	Infinity	0.2308	Infinity

Fig. 9 Initial results from MC

Chapter 5 Improvements

During the development of any program, decisions, why to choose something over another, are accompanying the process along the way. In this chapter the results are reviewed and the changes that are being made in MC are explained. These decisive choices can be used after as a reference from which MC could be further improved.

Stage 1 - Using OTSU to find thresholds

On **Fig. 9** is shown that MC calculates the metric for ‘borders.bmp’, but not for the other two images. The reason behind is that there are no objects on screen after the filtering process is complete. This objects would of been passed to the metric calculation algorithms in MC. The filter requires objects larger than 20 by 20 pixels and angle of rotation less than 2 degrees. When using the histogram median methodology to determine the thresholds, edges and contours are still detected in images - ‘asb.bmp’ and ‘google.bmp’, but some of the contours are not complete compared to the original image(See **Fig. 10).** In order more edges to be detected, therefore more complete contours, the threshold values must be set lower. As explored in Chapter 3 ‘System Implementation’, Section ‘C’, setting manually those values does not give good results for all images. There is a method from the library that uses the OTSU algorithm in order to automatically determine the optimum threshold value. In MC the output from this method is set to be the high threshold with a ratio of 3:1 high/low threshold. The results are visible on the figures below, so metrics has been calculated for ‘asb.bmp’ after this improvement.

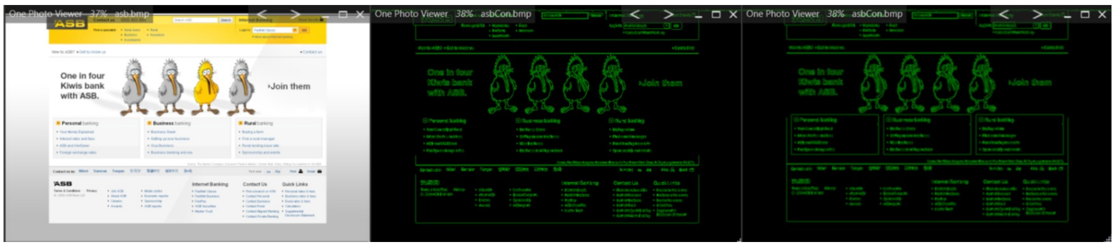
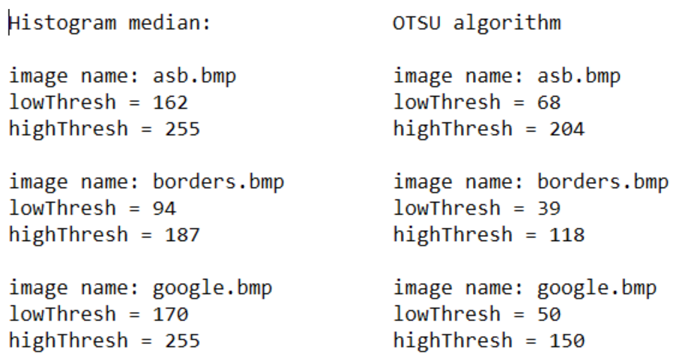


Fig 10 image ‘asb.bmp’ – original(left), contours histogram thresh(middle), contours OTSUthresh(right)



```
Histogram median:              OTSU algorithm

image name: asb.bmp            image name: asb.bmp
lowThresh = 162                lowThresh = 68
highThresh = 255               highThresh = 204

image name: borders.bmp        image name: borders.bmp
lowThresh = 94                 lowThresh = 39
highThresh = 187               highThresh = 118

image name: google.bmp         image name: google.bmp
lowThresh = 170                lowThresh = 50
highThresh = 255               highThresh = 150
```

Fig. 11 threshold values calculated using different methods

<u>Stage ‘1’</u> (OTSU)		‘asb.bmp’	‘borders.bmp’	‘google.bmp’
Equilibrium	manual	0.9854	0.9942	0.9980
	MC	0.9918	0.9879	NaN
Density	manual	0.5800	0.4786	0.9493
	MC	0.7577	0.3216	1.0000
Simplicity	manual	0.2307	0.2500	0.3750
	MC	0.3000	0.2500	Infinity

Fig. 12 Results when OTSU algorithm determines threshold values

Stage 2 - Better Filtering

The Density metric is highly dependent on the sum of the areas of all the objects on screen. The filter at stage ‘1’ doesn’t prevent rectangles from overlapping or be very close to each other, representing the same section of a web-page. This cannot be seen with ‘naked’ eye. Additional code is added to the filter that removes rectangles for the calculation which mass center point is next to another mass center point with a distance of less than 10 pixels away, and differences between the rectangles widths and heights with also less than 10 pixels. That is why the values for the metrics changes in this stage, seen in the table that follows.

<u>Stage ‘2’</u> (Stage ‘1’, and improved filter)		‘asb.bmp’	‘borders.bmp’	‘google.bmp’
Equilibrium	manual	0.9854	0.9942	0.9980
	MC	0.9882	0.9845	NaN
Density	manual	0.5800	0.4786	0.9493
	MC	0.7979	0.3660	1.0000
Simplicity	manual	0.2307	0.2500	0.3750
	MC	0.3333	0.3000	Infinity

Fig. 13 Results after improved filter

Stage 3 - Extracting external contours only

For the purpose of comparing the output metric values and choosing the optimum stage of MC at the end, the results in this stage are attained with the stage ‘2’ settings but, extracting the external contours only. This means that object(s) located within the borders of another object(s) on screen will not be extracted and used for the calculations.

Stage ‘3’ (Stage ‘2’, with external contours only extracted)		‘asb.bmp’	‘borders.bmp’	‘google.bmp’
Equilibrium	manual	0.9854	0.9942	0.9980
	MC	0.9672	0.9686	NaN
Density	manual	0.5800	0.4786	0.9493
	MC	0.8783	0.6073	1.000
Simplicity	manual	0.2307	0.2500	0.3750
	MC	0.4286	0.4286	Infinity

Fig. 14 Results with only external contours extracted

Stage 4 Calculating with external contours only without filtering

This stage explore results achieved with MC settings from stage ‘3’, but with no filtering at all. This means that all rectangles will be taken into account for the calculations, no matter of their size or their angle of rotation.

Stage ‘4’ (Stage ‘1’, with no filtering)		‘asb.bmp’	‘borders.bmp’	‘google.bmp’
Equilibrium	manual	0.9854	0.9942	0.9980
	MC	1.0000	1.0000	1.0000
Density	manual	0.5800	0.4786	0.9493
	MC	0.8649	0.6002	0.9935
Simplicity	manual	0.2307	0.2500	0.3750

	MC	0.0033	0.0117	0.0037

Fig. 15 Results with only external contours extracted and no filter applied

5.1 Evaluation

On the graphs below are shown all metric values for the three images going through all the stages. The manual calculations are considered the closest to the real values of the images. Metric values from previous project, done by colleague students, are used for comparison purposes. They used the web-page html source code as an input to their program, and not screenshot images. It is visible that these values are not close to the manual calculations.

Equilibrium: the equilibrium values for all the stages, apart from ‘Previous’, are close to the manual calculated once. Values for ‘google.bmp’ are missing in places for reasons discussed above.

Density and Simplicity: Values at stages ‘4’ and ‘5’ theoretically should be the farthest from the manual once as they do not take into account objects that are inside other objects for the calculations. This is confirmed in the results below. Stage ‘0’ is disregarded as there are important edges not detected at this stage. It is noticeable that the values at stage ‘1’ are the closest to the manual, ‘real’ once, but actually MC is designed better and is more effective at stage ‘2’. The reason behind is that MC does not detect all objects(discussed more in limitations). If all objects were detected, the values at stage ‘2’ would be the optimum values, and the closest to the real once.

MC will be set to stage ‘2’ as this stage is the optimum stage.

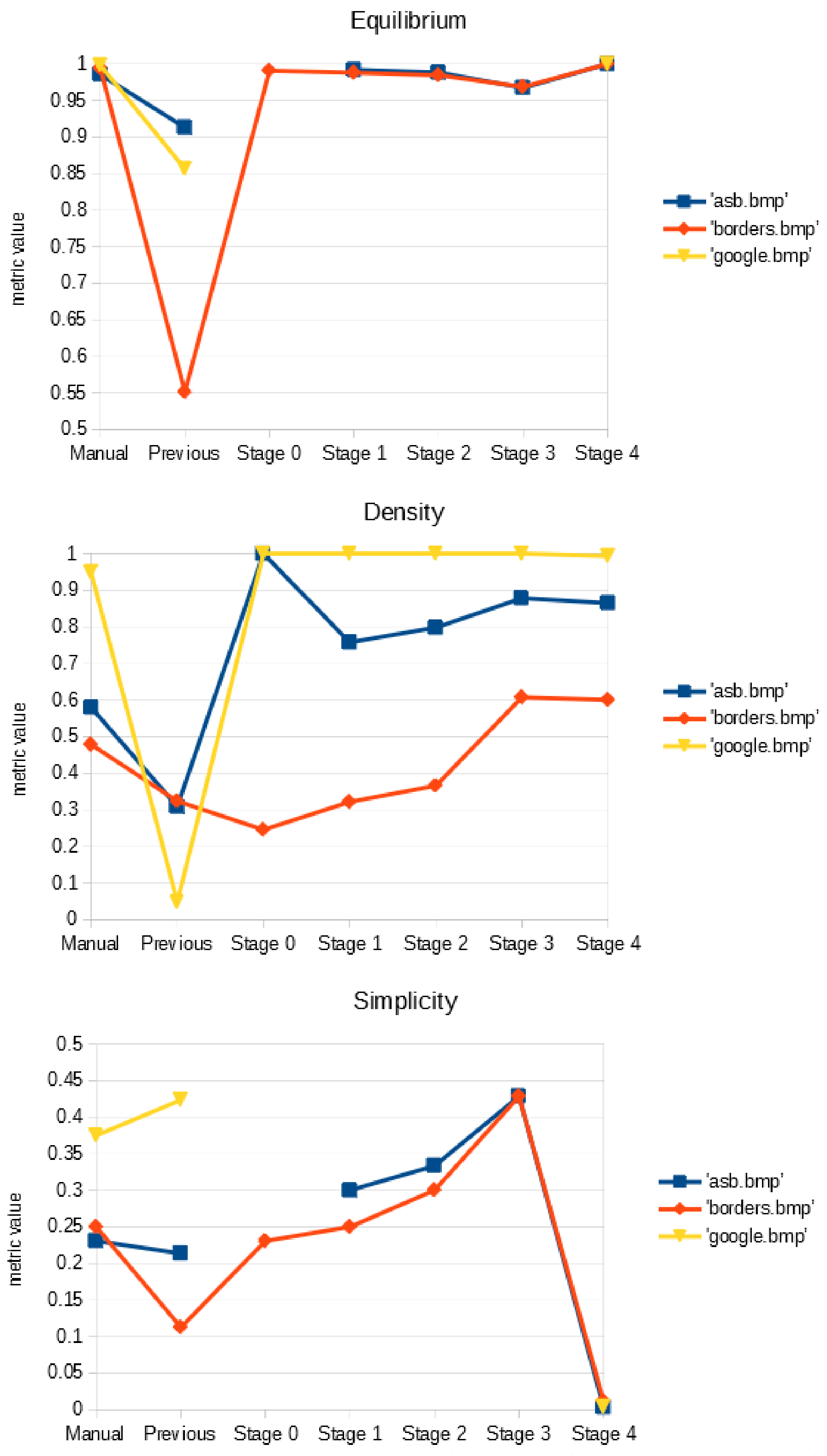


Fig. 16 Metric values in graphs

5.2 Limitations

In order calculated metrics by MC to be closer to the manual once more accurate data for the objects on screen must be fed into the metric algorithms. ‘google.bmp’ does not provide results for the metrics at all stages, apart from the last one, as no output data(objects) is coming out of the filter. As shown on the figure below, edges and contours are detected for this image, but the rectangles around the contours don’t represent the real objects on screen. This is because there are missing pixels from different parts of the objects, and MC ‘sees’ one object as multiple smaller objects. To improve the object detection MC should add the missing pixels on places, where is more likely to have some.

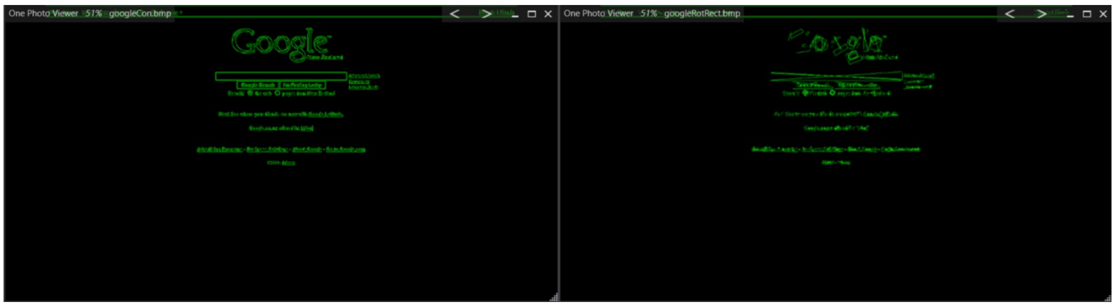

One Photo Viewer 51% googleCon.bmp
One Photo Viewer 51% googleRotRect.bmp
Google

Fig. 17 Image: (img3), contours(left), rotated rectangles around contours(right)

Chapter 6 Conclusion and Further work

MC currently detects successfully the edges of images. If another automatic method for detecting thresholds is set in place, while determining lower thresholds will mean more noise on images and therefore more objects on screen than there really are. If it determines values, less than the current once it was discussed above that it would skip detecting some/all objects on screen.

MC finds successfully the contours of images.

MC filter needs improvement as it does not provide the metric algorithms with the correct data for calculations. This could be achieved by adding another section between Section 'D' and Section 'E' (**See Fig.1**), which section would connect the missing pixels between different parts of objects.

MC metric algorithms could be truly tested once the additional section is implemented and attained results compared with manual calculated data.

Future work

The new section will process additionally the contours before rectangles are put around the contours. The processes known as 'dilation', followed by 'erosion' would take place for more accurate detection of objects on screen. These processes are morphological operations and there are methods in OpenCV library that explore these object transformations. They are normally performed on binary images, and there are two inputs – the original image and structuring element called 'kernel'.[14]

Erosion:

The kernel slides through the image while on places changing pixels to '0', or to '1'. The pixel will be considered '1' only if all pixels under the kernel are '1', otherwise is it made '0'. The idea is similar to soil erosion, but it erodes only the boundaries of the object. This method is useful to remove white noises.[14]

Dilation:

It is the opposite process of erosion. While the kernel size slides through the image a pixel will be changes to '1' only if there is at least one pixel with value '1' in the kernel. This method is very useful, because it can join broken parts of an object. In MC for example it would restore rectangle object.[14]

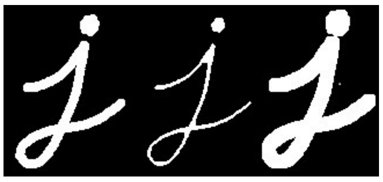


Fig. 18 original(left), erode operation(middle), dilate operation(right)[14]

Chapter 7 References

[1]A. Nair and A. Nair, "Top 5 Libraries For Image Processing In Java", *Analyticsindiamag.com*, 2019. [Online]. Available: https://analyticsindiamag.com/top-5-libraries-for-image-processing-in-java/. [Accessed: 31- Oct- 2019].
[2]"OpenCV", *ImageJ*, 2019. [Online]. Available: https://imagej.net/OpenCV. [Accessed: 31- Oct- 2019].

[3]"Mat - The Basic Image Container — OpenCV 2.4.13.7 documentation", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html. [Accessed: 31- Oct- 2019].

[4]"Mat - The Basic Image Container — OpenCV 2.4.13.7 documentation", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html. [Accessed: 17- Nov- 2019].

[5]L. Ding, "On the Canny edge detector", 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320300000236. [Accessed: 06- Dec- 2019].

[6]2019. [Online]. Available: https://scholar.googleusercontent.com/scholar?q=cache:2lzNgQBAgC4J:scholar.google.com/+canny+edge+detecting+algorithm&hl=en&as_sdt=0,5. [Accessed: 19- Nov- 2019].

[7]"OpenCV: Canny Edge Detection", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/4.1.1/da/d22/tutorial_py_canny.html. [Accessed: 20- Nov- 2019].

[8]A. Rosebrock, "Zero-parameter, automatic Canny edge detection with Python and OpenCV - PyImageSearch", *PyImageSearch*, 2019. [Online]. Available: https://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/. [Accessed: 12- Nov- 2019].

[9]"OpenCV: Contours : Getting Started", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/4.1.1/d4/d73/tutorial_py_contours_begin.html. [Accessed: 20- Nov- 2019].

[10]"OpenCV: Contour Features", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/4.1.1/dd/d49/tutorial_py_contour_features.html. [Accessed: 20- Nov- 2019].

[11]H. Purchase, J. Hamer, A. Jamieson and O. Ryan, "Investigating objective measures of web page aesthetics and usability", *Dl.acm.org*, 2011. [Online]. Available: https://dl.acm.org/citation.cfm?id=2460619. [Accessed: 26- Nov- 2019].

[12]D. Ngo and J. Byrne, "Application of an aesthetic evaluation model to data entry screens", *https://www.sciencedirect.com/science/article/pii/S074756320000042X#aep-section-id54*, 2001. . [Accessed 1 December 2019].

[13]L. Harmon and K. Knowlton, "Picture Processing by Computer", *Science*, vol. 164, no. 3875, pp. 19-29, 1969. [Accessed 1 December 2019].
[14]"OpenCV: Morphological Transformations", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/4.1.1/d9/d61/tutorial_py_morphological_ops.html. [Accessed: 04- Dec- 2019].