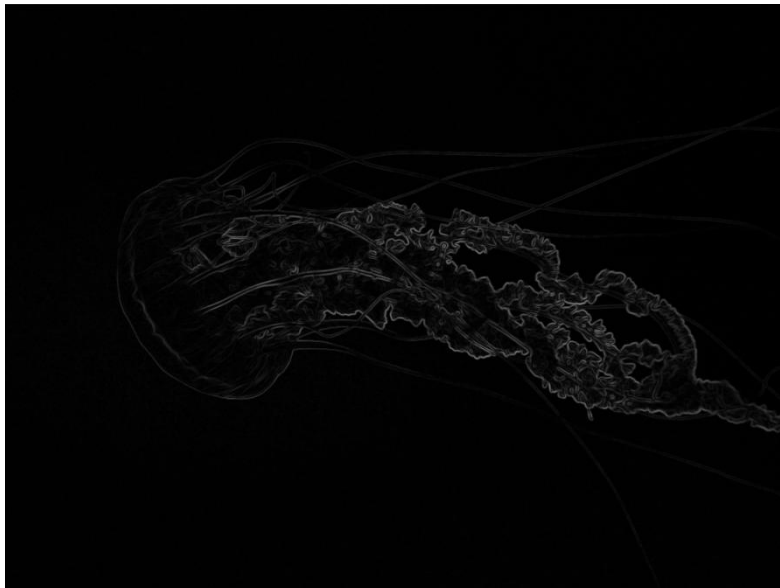


## The Computing Practical Project



*From Microsoft sample images*

## Digital Image Manipulation

---

<b>Candidate Name</b>	Jonathan Keable
<b>Candidate Number</b>	5238
<b>Centre Name</b>	Woodbridge School
<b>Centre Number</b>	19269
<b>Date</b>	15.03.12

# Table of Contents

---

<b>ANALYSIS .....</b>	<b>5</b>
INITIAL PROJECT BACKGROUND INFORMATION AND IDENTIFICATION .....	5
<i>Edge detection</i> .....	5
Simple Laplacian Edge Detection.....	5
Sobel Edge detection.....	6
Grey-scaling .....	8
<i>Noise Removal</i> .....	9
<i>Smoothing</i> .....	9
<i>Red Eye reduction</i> .....	10
DESCRIPTION OF THE CURRENT SYSTEM .....	10
IDENTIFICATION OF THE PROSPECTIVE USER(S).....	10
IDENTIFICATION OF USER NEEDS AND ACCEPTABLE LIMITATIONS.....	10
DATA SOURCE AND DESTINATION.....	12
<i>Inputs</i> .....	12
<i>Outputs</i> .....	12
DATA VOLUMES.....	13
ANALYSIS DATA DICTIONARY .....	14
DATA FLOW DIAGRAMS .....	17
<i>L0</i> .....	17
<i>L1</i> .....	18
NUMBERED SMART OBJECTIVES .....	19
POTENTIAL AND CHOSEN SOLUTIONS.....	20
<i>Edge Detection</i> .....	20
<i>Algorithm Choices</i> .....	20
Canny Edge Detection .....	21
Difference of Gaussians.....	23
<b>DESIGN .....</b>	<b>24</b>
INTRODUCTION.....	24
DESCRIPTION OF MODULAR STRUCTURE SYSTEM .....	24
<i>System Design Flowcharts</i> .....	25
<i>Educational Tools</i> .....	26
<i>Algorithm Editing</i> .....	26
HIERARCHY CHARTS.....	27
STRUCTURE CHART .....	27
DESIGN DATA DICTIONARY .....	28
<i>OriginalImage</i> .....	28
<i>TemporaryImage</i> .....	28
<i>EditedImage</i> .....	28
<i>RGBvalue</i> .....	28
<i>Redvalue</i> .....	28
<i>Bluevalue</i> .....	28
<i>Greenvalue</i> .....	29
<i>Record: RedTolerance</i> .....	29
RedTolerance.min .....	29
RedTolerance.max.....	29

<i>EdgeTolerance</i> .....	29
<i>FunctionRequired</i> .....	29
<i>NeighbourMean</i> .....	30
<i>NeighbourMedian</i> .....	30
DESCRIPTION OF RECORD STRUCTURE (NOT APPLICABLE) .....	30
VALIDATION REQUIRED .....	30
FILE ORGANISATION AND PROCESSING .....	31
<i>Name and Purpose</i> .....	31
<i>Structure</i> .....	31
SAMPLE OF PLANNED SQL QUERIES (NOT APPLICABLE) .....	31
IDENTIFICATION OF STORAGE MEDIA .....	31
PSEUDOCODE .....	31
<i>Sobel Edge Detection</i> .....	31
<i>Laplacian Edge Detection</i> .....	32
<i>Smoothing</i> .....	32
<i>Noise Reduction</i> .....	32
<i>Grey-Scaling</i> .....	33
<i>Red Eye Reduction</i> .....	33
<i>General Algorithm</i> .....	33
GRAPHIC USER INTERFACE DESIGN .....	34
<i>Prototyping</i> .....	34
Main Program .....	34
Red Eye Reduction Tolerance Editor .....	35
Educational Information Display .....	36
<i>Usability</i> .....	36
<i>Colour Scheme</i> .....	37
<i>Input/output devices</i> .....	37
<i>Error messages, feedback and dialogue boxes</i> .....	37
<i>Exits and actions</i> .....	37
DESCRIPTION OF MEASURES FOR PLANNED SYSTEM SECURITY .....	37
TEST PLAN .....	38
<b>TECHNICAL SOLUTION</b> .....	<b>39</b>
<b>TESTING</b> .....	<b>40</b>
<b>SYSTEM MAINTENANCE</b> .....	<b>47</b>
SYSTEM OVERVIEW .....	47
<i>Implementation Hierarchy Chart</i> .....	48
<i>GUI Implementation</i> .....	49
Original Design .....	49
Implemented .....	50
DETAILED ALGORITHM DESIGN .....	52
<i>Pseudocode Algorithm</i> .....	52
<i>Pascal code</i> .....	52
ANNOTATED LISTINGS .....	60
<i>Procedures/Functions</i> .....	60
Unit2 .....	60
Unit3 .....	61
Unit4 .....	62
<i>Constants/Variables</i> .....	62
Unit2 .....	62

<b>USER MANUAL .....</b>	<b>65</b>
<b>APPRAISAL .....</b>	<b>66</b>
INTRODUCTION .....	66
COMPARISON OF IMPLEMENTATION WITH “SMART” OBJECTIVES .....	67
END USER FEEDBACK .....	70
ANALYSIS OF USER FEEDBACK .....	70
<i>Imaging</i> .....	71
Laplacian Edge Detection .....	71
Smoothing .....	71
Noise Removal .....	71
Red Eye Reduction .....	71
<i>User Interface and Usability</i> .....	71
POSSIBLE EXTENSIONS .....	72
<i>Imaging</i> .....	72
Improved red eye tolerance editor .....	72
Improved file handling interface .....	72
Improved installation .....	72

## Appendices

---

*Appendix A: Technical Solution*

*Appendix B: Evidence of Testing*

*Appendix C: User Manual*

*Appendix D: End User Feedback*

# Analysis

---

## Initial project background information and identification

Not soon after the first digital image was captured, it was desired to be able to edit digital images in ways never possible before such technology. In this project I will be creating a simple bitmap editing system able to carry out a number of editing techniques including edge detection, noise removal, smoothing, and red eye reduction; all of these techniques will be part of a learning tool designed around the image editing chapter of the AS physics syllabus. Other kernel imaging algorithms may be added later as expansion material on the syllabus.

The project was suggested to me by my A-Level physics teacher, Mr Cottrell, who has been a most cooperative 'customer' and I thank him sincerely for his aid. He suggested the project as many of the algorithms and ideas to be used are part of the AS physics syllabus. I have also however used other sources for more information on the subject, so that my knowledge on the wider subject of image editing is good enough for the system to be cohesive. The problem is that the current electronic learning system used by the physics department has been designed to cover the entire syllabus and thus is inadequate in certain areas, notably the image manipulation chapter of the AS course. Thus Mr Cottrell would like a simpler system that focuses entirely on image editing, specifically the algorithms I have already described. Some edge detection methods outside of the syllabus should be included to show how there are other techniques for this complex image editing device other than the simple Laplacian method covered in the AS syllabus. The system should be compact and self contained unlike the current system which requires multiple programs and has large amounts of material not related to the specific task in hand.

For clarity, I have explained the algorithms I will use below (n.b. all kernels are applied to the original image, not the image after some of the pixels have already had the kernel applied to them).

## Edge detection

### Simple Laplacian Edge Detection

In the AS syllabus, the simple edge detection kernel is used. Though not the most effective algorithm, for the purposes of this project I will implement it. The kernel is a simple+ shape, as shown by this diagram.

	-1	
-1	4	-1
	-1	

By multiplying the pixel you are looking at by 4, and subtracting each one of its 4 immediate neighbours, and taking the modulus of an answer, we get a result that we can use to decide where edges are. The larger the values are the more likely they are edges. By setting

the tolerance, we can decide how sensitive the kernel is. A slightly less noise sensitive, but more complex kernel is this:

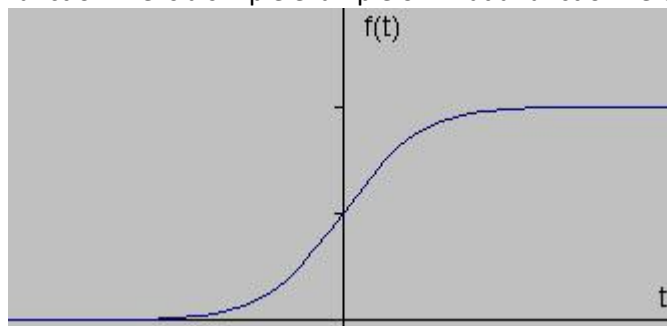
-1	-1	-1
-1	8	-1
-1	-1	-1

This kernel works in a similar way, by multiplying the pixel by 8, and subtracting its direct and diagonal neighbours from it. Because it covers a larger area, it is less affected by a single pixel of noise. This second kernel is also discussed in the AS physics course and will likely be the one I will implement

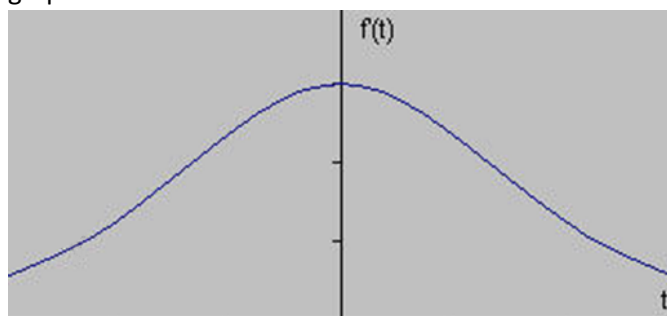
### Sobel Edge detection

I may also implement sobel edge detection if time allows, though this would be an extension of the AS physics course. I have explained the two methods below. [Some of The information in this section is paraphrased from [www.pages.drexel.edu/~weg22/edge.html](http://www.pages.drexel.edu/~weg22/edge.html). unless stated otherwise images are also from this website, and are under the copyright ownership of the author of the website].

The sobel method is of a group of edge detection method known as gradient methods. These methods use the gradient of a 1 dimensional signal, using the magnitude of that gradient to decide where the edge is. By taking the gradient in the x direction and the y direction edges can be detected on a 2D image. The gradient of a function is known as a derivative, and the sobel method treats every one dimensional line of pixels as a separate function. He is a simple example of what a function version of a line of pixels would look like.



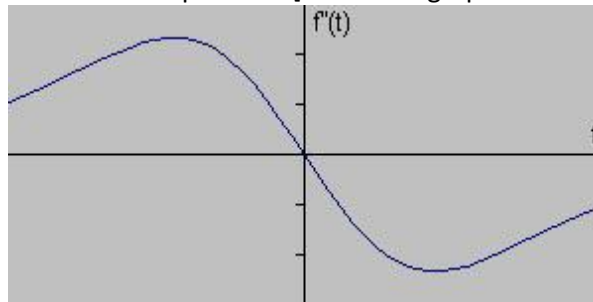
If we take the derivative for each point on the line of the function named  $t$  ( $f'(t)$ ) we get this graph



The graph clearly shows at what point the magnitude is greatest, and therefore where the edge is strongest on the image. If we set a tolerance, if the derivative at any point that which has a magnitude greater than the tolerance, then that point will be counted as an edge. An edge will either be assigned a black or a white value and a non edge will be assigned the opposite value. It would also be possible to set multiple thresholds and assign a different

grey value to each value to different strengths of edge, for instance non-edges are black, and edges move closer and closer to white as they become stronger.

The Laplacian method explained in the physics textbook actually uses the 2<sup>nd</sup> derivative, i.e. the derivative of the first derivative. In this case it is the derivative graph of the graph above. By doing this, we see that at the point where the magnitude of the 1<sup>st</sup> derivative is greatest, the 2<sup>nd</sup> derivative of this point is 0. Thus the Laplacian method detects an edge where the 2<sup>nd</sup> derivative is equal to 0. [the below graph is the 2<sup>nd</sup> derivative of the function  $t$ ,  $f''(t)$ ]



For sobel, in order to convert the 1D edge detection algorithm into a 2D one we use a pair of 3x3 convolution masks in the Sobel algorithm. Each one is an estimate of the gradient in either the x or y direction, and it is calculated by using the values of the 8 neighbouring pixels to get an approximation of the gradient. The algorithm for this is shown in the images: for the x direction subtract 2\* the value to direct left, subtract the values to the left and up and left and down, and then add 2\* the value to the direct right, and add the values to the right and up and right and down; the y direction works in a similar way. In order to combine the two gradient masks we must add the two gradient magnitudes together, but this only works if they are both positive. There are two ways of doing this: either use the squares of both gradients, add them together, and take the square root of the root, for a fairly accurate approximation, or take the modulus of both gradients and add those together to give us a less accurate approximation. The two masks and the two different approximation equations are given below.

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

Gx

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

Gy

$$|G| = \sqrt{G_x^2 + G_y^2}$$

$$|G| = |G_x| + |G_y|$$

A good explanation for how this works can be found on Wikipedia.

The result of the Sobel operator is a 2-dimensional map of the gradient at each point. It can be processed and viewed as though it is itself an image, with the areas of high gradient (the likely edges) visible as white lines. The following images illustrate this, by showing the computation of the Sobel operator on a simple image.



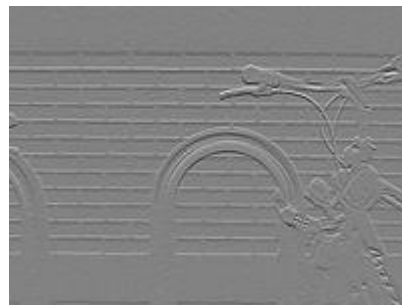
Grayscale image of a brick wall & a bike rack



Normalized sobel gradient image of bricks & bike rack



Normalized sobel x-gradient image of bricks & bike rack



Normalized sobel y-gradient image of bricks & bike rack

It is important to note that the image above has been grey scaled before the edge detection algorithm has been carried out, and this is because grey scaling often improves the effectiveness of sobel edge detection. It would thus be useful to implement this algorithm also in the system, and it is explained below.

### Grey-scaling

In order to convert a RGB image to greyscale you must look at the RGB values of each pixel. Then add together 30% of the Red Value, 59% of the green value, and 11% of the blue value for the best results for a typical image, and then store these as the linear luminescence values. These can be stored in 8 bit greyscale pixels, and the same edge detection algorithm applied to them.

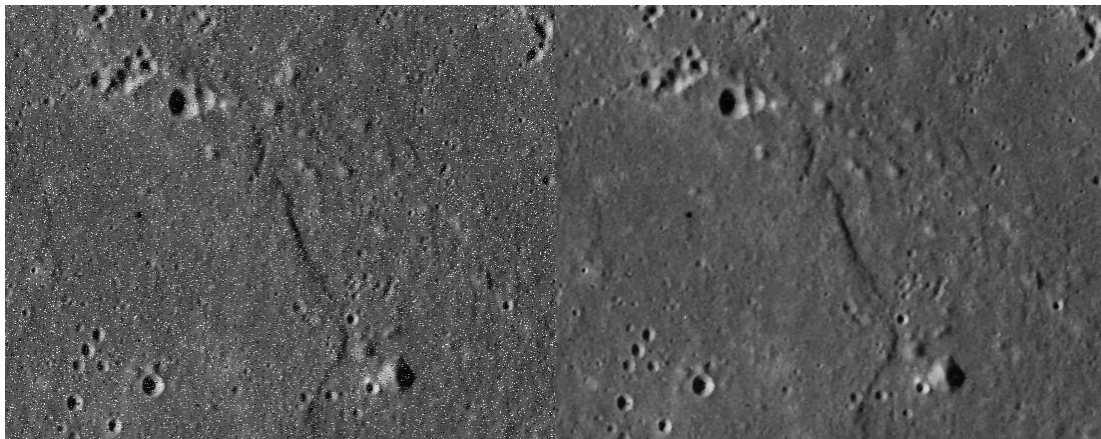
In the Laplacian method we use just a single 3x3 convolution mask, as opposed to 2 3x3 grids in the sobel method. All of the values in the grid are subtracted from the central pixel multiplied by the number of pixels in the kernel excluding itself, and if the result is 0 an edge is said to be detected. This method is simpler, but more sensitive to noise, as only one mask is used.



For both methods the mask must be slid across the image either row by row, or column by column (usually the former). Because the masks are both 3x3, the algorithm cannot be applied to the pixels on edges and comers, without changing the kernel. I have a choice of either adding more kernels, or simply deleting the outer pixels of the image after the kernels are applied.

## Noise Removal

The algorithm for noise removal is relatively simple. Take the median value for the pixel itself and the 8 surrounding pixels in a 3x3 kernel, and set its value to that. This keeps most of the data, whilst still doing an effective job of removing noise. A noisy and cleaned up image are shown below.



*[Images from 'advancing physics AS edition' learning tool, and 'scion image editor']*

## Smoothing

The algorithm for smoothing, like noise removal, is relatively simple. Take the mean value for the pixel itself and the 8 surrounding pixels in a 3x3 kernel and set its value to that. This loses more data than median based noise removal, and makes the image appear blurred. By using the smoothing algorithm multiple times, noise can be removed, but this is not as effective as standard noise removal. An original image and the same image after multiple smoothing algorithms are shown below.



*[Images from Microsoft (sample images)]*

## Red Eye reduction

In order to reduce red eye the red colour in an image must be de de-saturated. This is achieved by searching for pixels within a certain tolerance of red (e.g. FF0000). In this example pixels from FF0000 to 000000 could be counted as red. Once these pixels are located, they can be changed to a grey value such as 808080. This will obviously cause all red pixels in the image to be changed to grey, so to avoid the loss of useful data it would be preferable for the user to select the area for red eye reduction to be applied before it is carried out.

## Description of the current system

Currently the Physics department uses a system called “advancing physics, AS edition”. It uses an “electronic textbook” with large amounts of reading material as well as question and answer sections. The system works well generally, but in order to demonstrate digital media editing it has a multiple program installation including programs such as ‘scion image editor’ and ‘audacity’. The scion program is a simple program that requires installation and can be difficult to use. For this problem Mr Cottrell would like a more ergonomic program that is stand alone and requires no installation. The current program (scion) has no real educational tools, and the new learning program should including some limited information on the kernels used, though its primary purpose will still be the implementation of the image editing techniques in the AS syllabus.

## Identification of the prospective user(s)

At the moment the only user who will definitely want to use the system is Mr Cottrell, as a learning tool on the projector, and the program could also be distributed to the physics laptops, though is less likely. Having said this, I will also be able to use the system of course, and other students may wish to view the source code and program to be created for the purposes of learning, and thus I will likely make it available through the use of the shared documents system, specifically the sandbox area allocated for programs created by students, at some point. The sharing of code for the purposes of the improvement of programming is common in the school computing community and the sharing of code through the sandbox is very likely to occur at some point.

## Identification of user needs and acceptable limitations

Here I will explain what the system needs to be able to do for the user and what it would be useful for it to do if possible in the specified time with the resources available. Firstly the system *must* include the four algorithms which are outlined in the AS advancing physics textbook which are:

- Laplacian Edge Detection
- Red-Eye Reduction
- Smoothing
- Noise Reduction

Mr Cottrell must have these algorithms both implemented into the system as well as described in detail to the user of the system, namely students, so that they understand how they work. It is important that the students can be set tasks to do using the system by the teacher which they can do both in and out of class. To this end a simple menu based system has been suggested, although the most important point is that all 4 of the main algorithms are in one place.

However there are other elements to imaging which though not explicitly explained in the AS syllabus are useful to understand for more adept students. These are:

- Grey scaling: Students do not need to know how grey scaling works, but should be aware that edge detection on colour images is much less reliable due to the way the algorithm works. As such Mr Cottrell would like a grey scaling algorithm implemented into the system so that students can compare results of edge detection with grey and colour images. Alternatively Images could be provided with the system which are in grey and colour if this is more practical
- Negative: Edge Detection leaves edges white with not edges black when carried out on an image. Some people find it difficult to distinguish white on a black background, so it would be useful for students to be able to invert the colour so that they can clearly see the edge detection algorithm has worked.
- Red Tolerance Editing: The AS course tells students that red eye reduction works by finding all “red” pixels on an image and simply replacing them with an arbitrary colour, usually grey. However more advanced student will understand that there are many shades of “red” and it would be useful, though not essential, to have a part of the program which allows the user to tell the program what “red” is, so that they can see the effect this has on the application of the algorithm.

Because the system will be a learning tool it must be provided with a number of predefined images which exhibit well the imaging algorithms in the AS course. These images should be chosen to show the algorithms working on both a large scale (on full size images) and on a small scale (small grids of pixels) So that they can be used in a structured way as part of teaching the particular part of the course. These images should have a combined size in

terms of memory used so that the whole system can fit on portable media, most likely CD's ,although personal USB drives or even a downloadable form of the system on the school's Virtual Learning Environment may also be used. Ultimately however the result of any of these distribution methods is that the system will be limited in size.

The system should also be able to manipulate images other than those provided however so that users can look at how the algorithm affects other images, to help encourage learning outside class and interest in this art of the course. For instance users may wish to see how effective the red eye reduction algorithm is on an image of their face with redeye, and by seeing how these algorithms work on a variety of images Mr Cottrell hopes users of the system will be more interested in the algorithms used.

The system must also work using the resources available to the Physics department. This means it must be able to be installed on the laptops which are situated there. These machines are fairly low spec so the system should work efficiently on slow processors. These machines are also in the process of being replaced with new machines running "Windows 7" as opposed to "Windows XP", so the system should work on both systems. The laptops also support multiple users, and the system should be available to all AS Physics teachers and students.

## Data Source and Destination

### Inputs

Item	Source	Input by	Frequency
Input Bitmap Image	User Documents	User	N/A
Editing Required	User	User	Every time the user wishes to edit the image

### Outputs

Item	Destination	Frequency
Output Bitmap Image	Cached to Disk	Whenever the user saves the new image
Temporary Bitmap Image	Temporarily Cached to Disk, Displayed to user	Every time Image editing algorithms are carried out



Input Bitmap Image	Cached to Disk	Whenever the user Inputs an image
Algorithm Data	Cached to Disk	Accessed by the program when editing is carried out
Kernel Data	Cached to Disk	Accessed by the program when editing is carried out, part of Algorithm Data


## Data Volumes

All images will have a size which varies depending on size and detail. For instance a 32 bit per pixel image of 1000 by 2000 pixels would take up  $4 \times 2000 \times 1000 = 8\text{MB}$ . Thus significant memory may need to be allocated in order to store these files.

Algorithm data and kernel data will be negligible in size.

## Analysis Data Dictionary

Variable (e.g. myvariable)	Type (e.g. integer)	Field size	example	comment
OriginalImage	File (Bitmap)	An acceptably large sized and below bitmap with an acceptably high and below resolution	Chrysanthemum. bmp 	The original bitmap image will be read from directly from the hard drive, and the system must be able to convert the image to a 2D array of binary for editing, and back to a bitmap for viewing by the user
TemporaryImage	File (Bitmap)	An acceptably large sized and below bitmap with an acceptably high and below resolution	Chrysanthemum. bmp 	The temporary image will be stored in a temporary location for editing. It will be stored as a file (Bitmap) for viewing by the user, but will also be stored as a 2D array of binary for editing

Variable (e.g. myvariable)	Type (e.g. integer)	Field size	example	comment
TemplImageBin	Array of binary (2D) [x..y][z..w]	An acceptably large sized and below bitmap with an acceptably high and below resolution, converted to binary	Chrysanthemum. bin	The binary version of the temporary image for editing
EditedImage	File (Bitmap)	An acceptably large sized and below bitmap with an acceptably high and below resolution	Chrysanthemum. bmp 	When the user saves the image it will be saved to a permanent location as a bitmap file only
pixeldata	record	A record of Binarydata, RGBvalue, and neighbouringpixels	n/a	A record of both the data stored in the pixel, and of where the pixel is in the image and what pixels neighbour it
Binarydata	Binary integer	From 1 bit to 64 bits	1101 0001	The raw binary data in each pixel
RGBvalue	HEX integer	From 1 bit to 64 bits stored as Hex	FA156B	The Hexadecimal value for each pixel so that it can easily be manipulated
neighbouringpixels	array	An array of 2d array coordinates with 4 different storagespaces	[(2,2),(2,4),(3,1),(3,3)]	An array of the locations of pixels in the bitmap file which neighbour a certain pixel

Variable (e.g. myvariable)	Type (e.g. integer)	Field size	example	comment
redtolerance	record	A record containing redevemin and redevemax	n/a	Record containing the red eye tolerance values
redevemin	integer	A Hex integer for the minimum value which will be accepted as red	660000	The minimum HEX value for which the red eye part of the system will take as red
redevemax	integer	A Hex integer for the maximum value which will be accepted as red	FF6666	The maximum HEX value for which the red eye part of the system will take as red
edgetolerance	integer	An integer value from – 2147483648 to 2147483647	5	When the sobel edge detection function outputs a value, if it is above or equal to this number then an edge has been detected.
Functionrequired	string	A string of up to 255 characters	RedEye	Each function will have a string value assigned to it so it can be identified as the required process.
neighbourmean	integer	An integer value from – 2147483648 to 2147483647	12324128	The mean of the 8 neighbours of a pixel

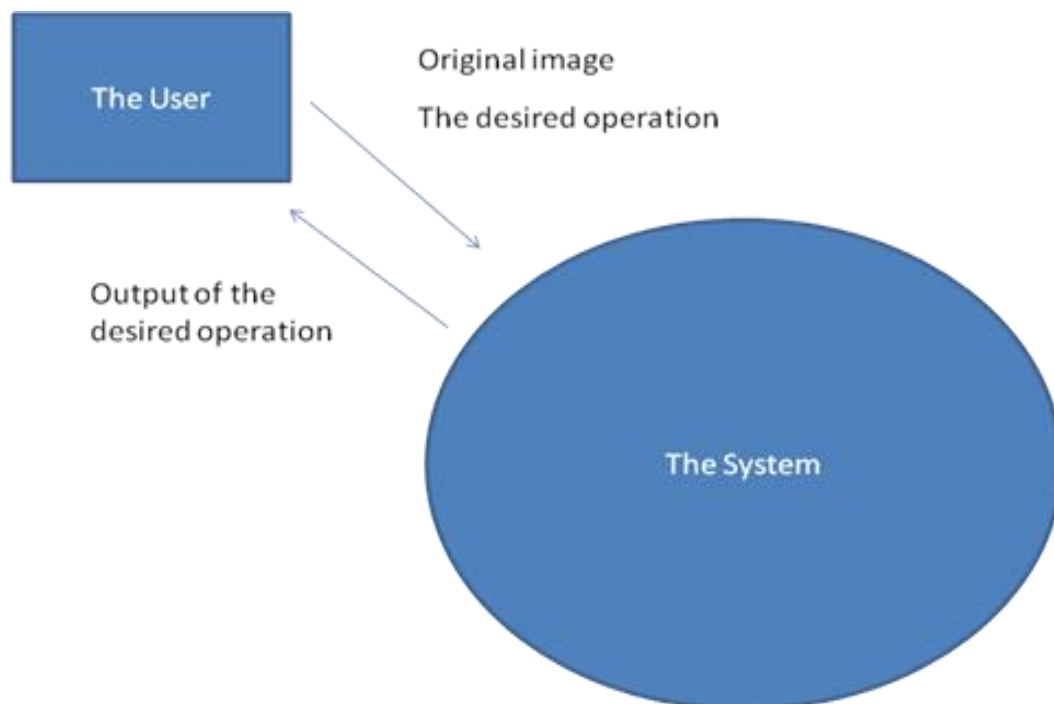


Variable (e.g. myvariable)	Type (e.g. integer)	Field size	example	comment
neighbourmedian	Integer	An integer value from – 2147483648 to 2147483647	12324128	The mean of the 8 neighbours of a pixel
...	...	...	...	...

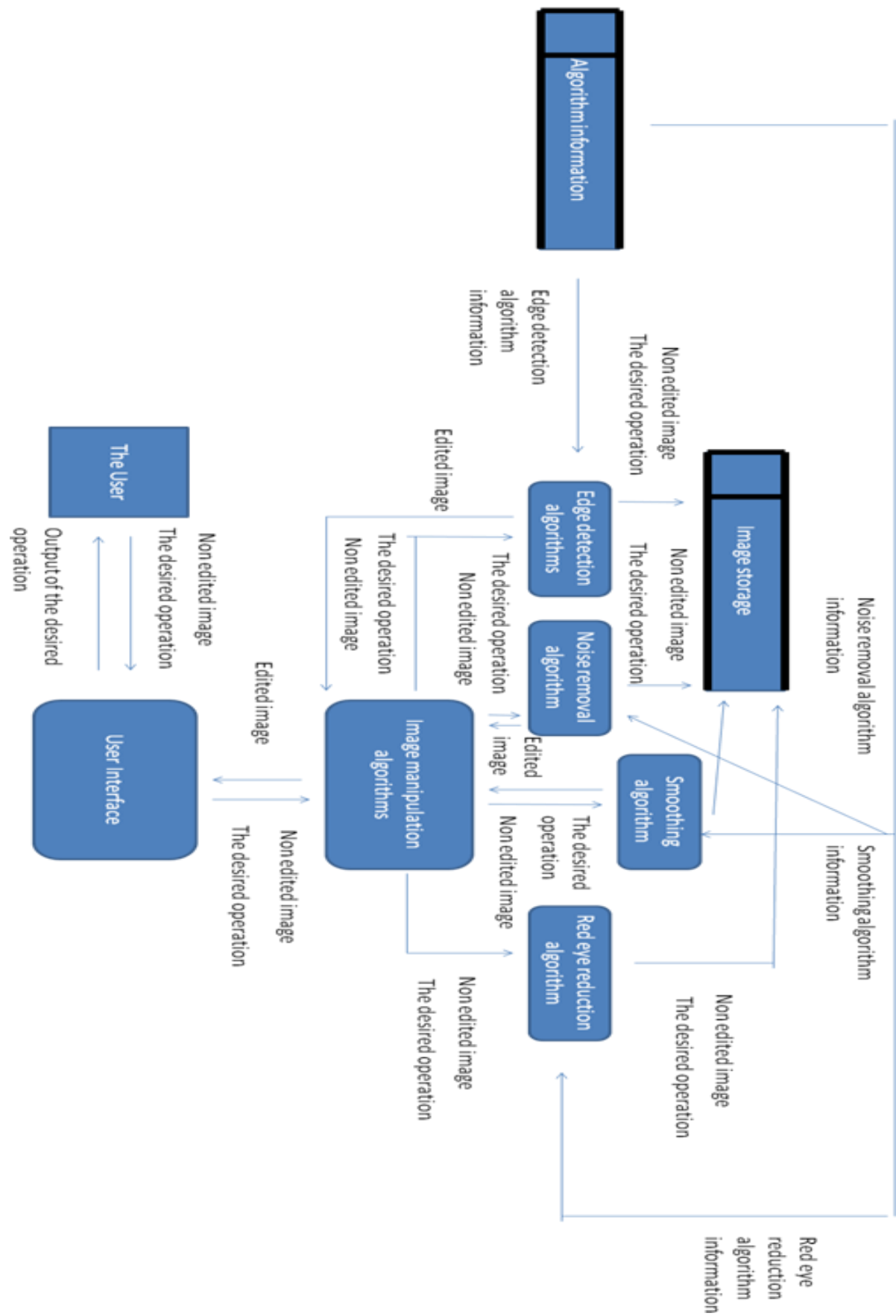
## Data Flow Diagrams

Due to the nature of the problem, no original system DFD is possible, however the DFD for the new system is:

L0



L1



## Numbered SMART objectives

1. The system must accurately and effectively implement the following algorithms
  - 1.1. Edge detection
    - 1.1.1. the Laplacian method must be implemented
    - 1.1.2. A sobel method could be implemented
    - 1.1.3. For the sobel method the user should be able to change for what value it is said an edge has been detected.
    - 1.1.4. The user must be able to choose whether the edges are white on black, or black on white.
    - 1.1.5. The system must be able to convert an image to greyscale before any edge detection is carried out
    - 1.1.6. The conversion of images to edge detection images should take no longer than 30 seconds
  - 1.2. Smoothing
    - 1.2.1. The system should be able to smooth images using the mean technique
    - 1.2.2. It could be possible to enter how many times the smoothing process should be repeated, if any.
    - 1.2.3. It should take no longer than 30 seconds for images to be smoothed
  - 1.3. Noise removal
    - 1.3.1. The system should be able to remove noise using the median method
    - 1.3.2. There should be as little loss of data as possible during noise removal
    - 1.3.3. It could be possible to enter how many times the noise removal process should be repeated, if any.
    - 1.3.4. It should take no longer than 30 seconds for noise to be removed from an image
  - 1.4. Red eye reduction
    - 1.4.1. It should be possible for the system to convert red pixels in an image to grey pixels
    - 1.4.2. It should be possible for the user to change the red tolerance of the process
    - 1.4.3. It could be possible for the user to select an area for which red eye is to be reduced
    - 1.4.4. It should take no longer than 30 seconds for red eye to be reduced.
2. There should be certain User Interface functions
  - 2.1. The user should be able to save an edited image without overwriting the original

- 2.2. The user could be able to undo changes to an image for at least 10 image edits
- 2.3. The user should be able to revert to the original image
- 2.4. There could be certain selection tools
  - 2.4.1. The user could be able to select a rectangular part of the image
  - 2.4.2. The user could be able to deselect a part of the image
  - 2.4.3. The user could be able to select more than one part of the image at a time
  - 2.4.4. The user could be able to apply the red eye algorithm only to a selection
  - 2.4.5. These tools may be based either on a curser system or on a coordinates system if this is impractical.
- 2.5. The user could be able to preview changes before they are made
3. The system should be easy to use
  - 3.1. It should not crash or have problems frequently
  - 3.2. It should be clear where different tools are
  - 3.3. It should not be possible to easily lose an edited image or delete an existing one
  - 3.4. The system must contain brief summaries of which each function does
4. The system should not damage other systems
5. The System should be compact enough to be sent by e-mail or to be put on a CD-ROM, DVD, or Flash memory Stick (it does not have to fit onto a Floppy Disk).

## Potential and chosen solutions

### Edge Detection

The reason I have chosen to approach the problem in a software based environment because it is very difficult to show how image processing algorithms work without a computer. Although students can work on a small scale, looking at for instance a 8x8 grid of pixels with values between 1 and 16, it is almost impossible to look at actual images with 1000s of pixels and colours. The reason I used Turbo Delphi to write the program is that is the language I am most comfortable with, and also contains a GUI building interface, so that I do not have to create classes for buttons, forms, images etc. Besides form these things it also possesses all the functionality required to write image editing software.

### Algorithm Choices

There are 4 main algorithms that are used for edge detection

1. Laplacian (zero edge detection)
2. Sobel (gradient edge detection)
3. Canny
4. Difference of two Gaussians

I must implement Laplacian edge detection as it is in the AS Physics course. None of the other algorithms are in the AS physics course, but it would be nice if maybe one of them could be implemented if time allows, and this would likely be the sobel algorithm. I have already explained Sobel and Laplacian edge detection, so I will explain how Canny and Difference of two Gaussians work and why I have chosen not to implement them

The Smoothing and Noise Reduction algorithms are taken directly from the AS physics book, using a mean and median 3x3 kernel respectively. There is no easy way to expand on these algorithms so I will not attempt to as I only need to cover the kernels in the syllabus anyway.

The Red eye reduction kernel is a simple “find colour red and replace with grey” process so I need not explain it in detail, and for the AS physics course it is the only procedure students need to know about.

In order to allow edges to be black on white or white or black a negative process will be implemented. It simply inverts colour by doing  $255 - \text{colour}$ , and is not in the AS course so will not be explained in educational information. The Greyscaling algorithm I use converts colour to grey using set values for how much of each colour (RGB) is included in grey. It too is purely for the purpose of making the edge detection algorithm more effective so also needs no explanation in educational information.

## Canny Edge Detection

The **Canny edge detection** operator was developed by [John F. Canny](#) in 1986 and uses a multi-stage [algorithm](#) to detect a wide range of edges in images. Most importantly, Canny also produced a *computational theory of edge detection* explaining why the technique works.

*From wikipedia.org*

The Canny edge detection is a multistage algorithm that has 6 main stages:

- [2.1 Noise reduction](#)
- [2.2 Finding the intensity gradient of the image](#)
- [2.3 Non-maximum suppression](#)
- [2.4 Tracing edges through the image and hysteresis thresholding](#)
- [2.5 Differential geometric formulation of the Canny edge detector](#)
- [2.6 Variational-geometric formulation of the Haralick-Canny edge detector](#)

*From wikipedia.org*

This method is one of the most effective edge detection methods known. Unfortunately it is also one of the most complex methods of edge detection, so is probably too complex for the A2 course. Basically it uses a Gaussian noise removal kernel, a gradient edge detection kernel, and the direction of the edges, to trace the image as if traced manually. This makes its outputs very accurate, but the sheer complexity of the algorithm means it is not often used. The image below shows just how effective it is.



*From Wikipedia.org, licensed under the Creative Commons Attribution-ShareAlike 3.0 License; author: JonMcLoone*

The images clearly show the effectiveness of the algorithm: the blurred edges that are not useful in the background are ignored, but even the individual hairs are identified as edges.

Advantages	Disadvantages
The most effective Edge Detection Algorithm, as it uses multiple algorithms to work out all the useful properties of an edge	The most complex algorithm to implement
The algorithm least effected by noise due to the Gaussian blur part of the algorithm	Due to the complexity of the algorithm the processing of large and/or high resolution images is slow
Is able to detect even very thin edges	Would require an amount of time to program which would be unreasonably considering the timescale involved
	Far beyond what is studied in the AS syllabus

## Difference of Gaussians

In [computer vision](#), **Difference of Gaussians** is a [grayscale](#) image enhancement algorithm that involves the subtraction of one blurred version of an original grayscale image from another, less blurred version of the original. The blurred images are obtained by [convolving](#) the original grayscale image with Gaussian kernels having differing standard deviations. Blurring an image using a [Gaussian kernel](#) suppresses only [high-frequency spatial](#) information. Subtracting one image from the other preserves spatial information that lies between the range of frequencies that are preserved in the two blurred images. Thus, the difference of Gaussians is similar to a [band-pass filter](#) that discards all but a handful of spatial frequencies that are present in the original grayscale image. [\[1\]](#)

*From Wikipedia.org*

By subtracting a blurred version of an image (blurred using a Gaussian kernel), you can achieve a grayscale edge detection image. The Gaussian kernel is quite a complex kernel, and is often large (5x5, 6x6, or even larger), making it a slightly more complex and slower algorithm, that is very good at eliminating noise, though not as effective as Canny Edge detection. It is therefore often used for high noise images, however is not used as much as other edge detection algorithms on most images; Because of this, and the complexity of Gaussian kernels, I have chosen not to implement this algorithm.



The images show that though good at not detecting noise, difference of Gaussians also loses some detail

*Images from Wikipedia.org, licensed under the Creative Commons Attribution-ShareAlike 3.0 License; author: SadaraX*

Advantages	Disadvantages
<p>Not Very Sensitive to noise (though slightly more sensitive than Canny edge detection)</p>	<p>Poor Detail</p> <p>Can take more time than most methods to process images due to kernel complexity, but faster than Canny edge detection</p> <p>Greyscale only</p> <p>Not widely used and not cohesive with the AS course</p>

# Design

---

## Introduction

For the purposes of this project I decided that it would be a good idea to make the system in three separate parts: Graphic User Interface (GUI), Logic, and Data Access. The GUI will deal with the basic processing of the user input (making sure it is the correct format, making it easy for the user to do what they want), and pass it on to the logic part of the system. The logic section will pass data back to the GUI, which will output this data to the user. The logic section will be the actual workings of the program; in this case handling bitmaps and pixel data, applying algorithms, etc. data will be passed between the Logic section and the GUI, and between the Logic section and Data Access section. The Data access section manages communications with the Data Storage, separating out the Data Handling from the Program Logic. There is no direct connection between Data Access and GUI sections of the system.

This way of breaking down the program into its constituent parts is known as the 3 box model as there are 3 separate parts of the system.

## Description of Modular Structure System

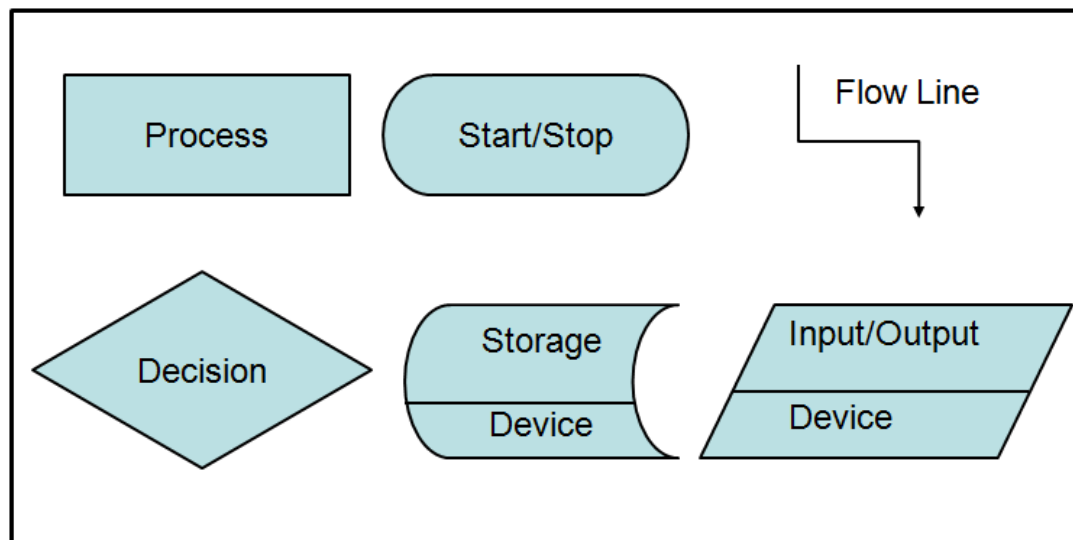
The System Can be broken down as follows:

- The Implementation of the Required Image Manipulation algorithms
- The Educational tools (Explaining how the algorithms work)
- The ability for the user to configure the algorithms (to an extent)
- The Graphical User Interface which must present images and information to the user
- The handling of data inputs and outputs, including the opening and saving of image files

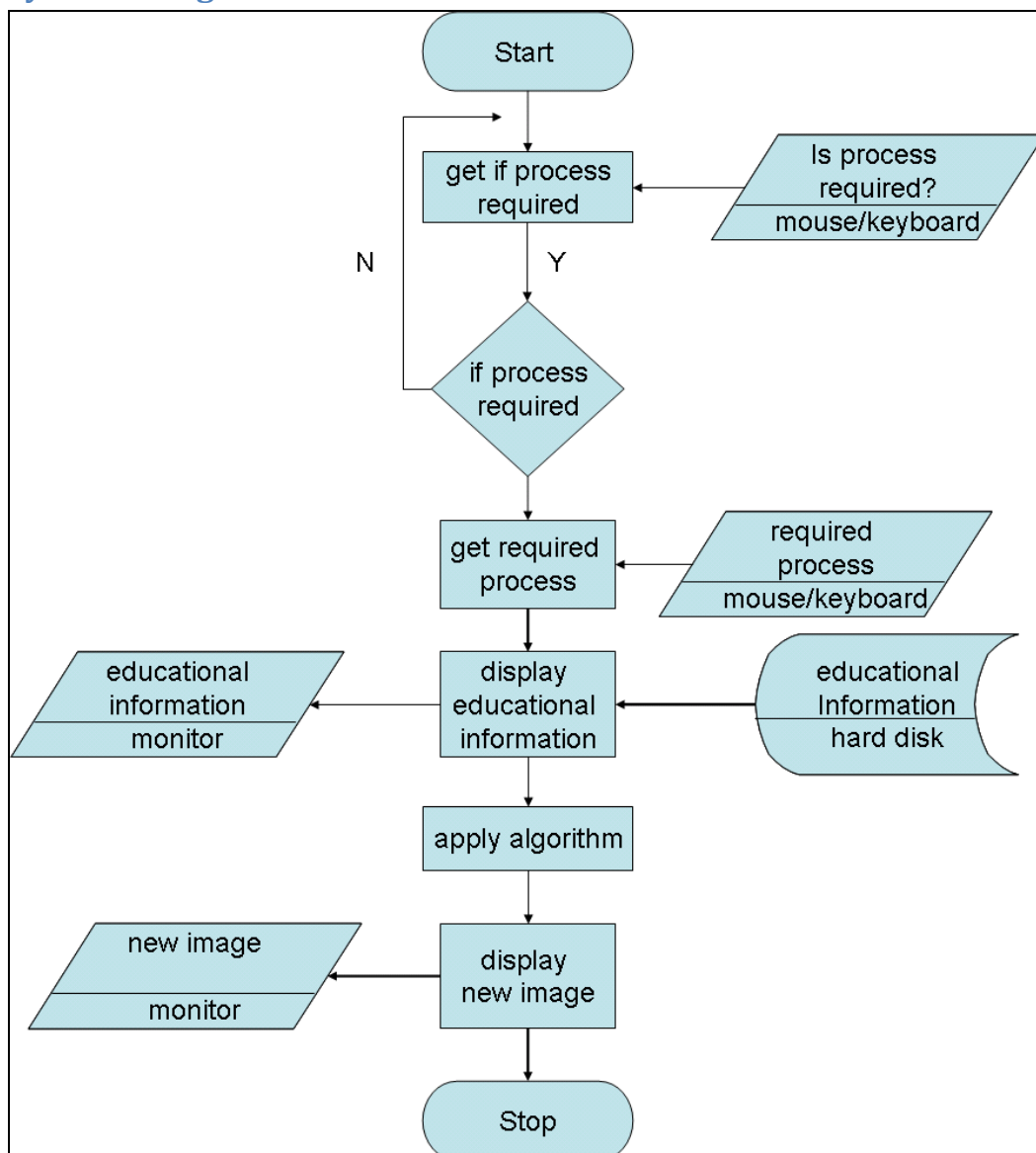
To a large extent the educational tools will be built into the other systems (i.e. information on the algorithms will be displayed when the algorithms are applied), however some information will be accessible without actually having to do any image editing for reference.

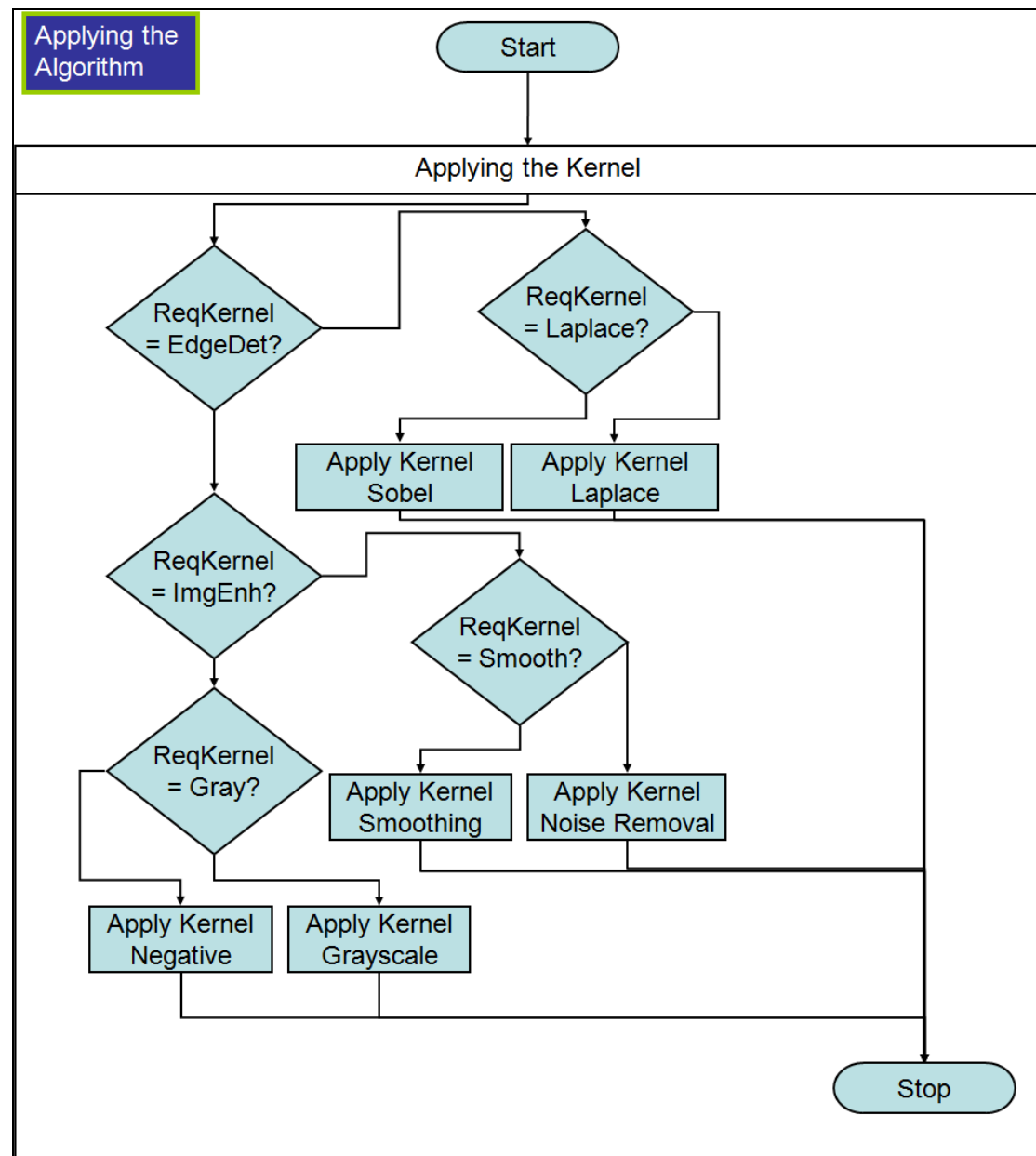
These general processes can be described using a series of system flowcharts. The key for these flowcharts is shown below:





### System Design Flowcharts





The apply algorithm function is triggered when the user wants to edit the image, and gets the required operation, and carries it out on the image which has already been inputted prior by the user, having displayed information on how the algorithm works to the user. The nested loop statement

### Educational Tools

The tools will be fairly simplistic, as in an electronic-textbook style resource, as well as being integrated into the other parts of the system. As such I believe it would be unnecessary to provide flowcharts for this part of the system.

### Algorithm Editing

The System must be able to allow the user to edit the algorithm so that students can see how changing the algorithm changes the outputs. In terms of the algorithms used this mainly involves editing the tolerances of the red eye algorithm, but also the sobel algorithm



## Design Data Dictionary

### OriginalImage

Description	The Original Image inputted by the user (a copy will be made for editing by the program)
Validation	Must be a bitmap image (no hard limit on image size will be imposed, instead the user will be asked not to use images above some size, approximately 15MB are not used)

### TemporaryImage

Description	Because Kernels are applied to the original image a Temporary Image must be created by the program before the finished edited image is saved
Validation	Must be a bitmap image (no hard limit on image size will be imposed, instead the user will be asked not to use images above some size, approximately 15MB are not used)

### EditedImage

Description	The Copy of the Image Created by the program for editing
Validation	Must be a bitmap image (no hard limit on image size will be imposed, instead the user will be asked not to use images above some size, approximately 15MB are not used)

### RGBvalue

Description	The data contained in a pixel (the RGB value)
Validation	Must be an Integer of a certain length depending on the n <sup>o</sup> of bits per pixel (8 bit, 16 bit, 32bit, 64 bit)

### Redvalue

Description	The Part of RGBvalue which represents red extracted from it.
Validation	Must be an Integer of a certain length depending on the n <sup>o</sup> of bits per pixel (8 bit, 16 bit, 32bit, 64 bit) where a certain number of those bits represent red.

### Bluevalue

Description	The Part of RGBvalue which represents blue extracted from it.
Validation	Must be an Integer of a certain length depending on the n <sup>o</sup> of bits per pixel (8 bit, 16 bit, 32bit, 64 bit) where a certain number of those bits

	represent blue.
--	-----------------

### Greenvalue

Description	The Part of RGBvalue which represents green extracted from it.
Validation	Must be an Integer of a certain length depending on the n <sup>o</sup> of bits per pixel (8 bit, 16 bit, 32bit, 64 bit) where a certain number of those bits represent green.

### Record: RedTolerance

Description	A record containing both the minimum and maximum value for which a pixel is said to be red
Validation	n/a
<b>RedTolerance.min</b>	
Description	An Integer containing the minimum and minimum value for which a pixel is said to be red
Validation	Must be a Hex Integer of size corresponding to that of the Hex data contained in each pixel of the image being edited
<b>RedTolerance.max</b>	
Description	A Hex Integer containing the minimum and maximum value for which a pixel is said to be red
Validation	Must be a Hex Integer of size corresponding to that of the Hex data contained in each pixel of the image being edited

### EdgeTolerance

Description	An integer value for which an Edge in one direction is said to be found if the value of a pixel after the sobel algorithm has been carried out
Validation	Must be an Integer between certain as yet undefined values (x<EdgeTolerance<y)

### FunctionRequired

Description	Each function will have an integer value assigned to it so it can be identified as the required process
Validation	A m Integer within a certain range (considering this will be a constant, this is not really appropriate)

### NeighbourMean

Description	The Mean of all the neighbours of a pixel as well as the pixel itself (used in the smoothing function )
Validation	An Integer Value within a certain range depending on the number of bits per pixel

### NeighbourMedian

Description	The Median of all the neighbours of a pixel as well as the pixel itself (used in the Noise Removal function )
Validation	An Integer Value within a certain range depending on the number of bits per pixel

### Description of record structure (not applicable)

The proposed system will have a simple save structure not linked to a database so this part of the design would not be appropriate.

### Validation Required

Detailed information about validation of variables is included in the Design data Dictionary so this section is not really useful.

## File organisation and processing

Due to the nature of the system most of the information will be stored on the portable media it is being run on, but it should also be possible to copy the entire program file structure to the user's computer for more rapid program execution.

Name and Purpose	Structure
ImageManipulationEducationalTool_1.0 Contains executable	----ImageManipulationEducationalTool_1.0       ImgTool_1.0.exe
Docs Help Information and a plain pdf version of the Image Manipulation virtual textbook	----Docs       IMET_Readme.txt       IMWT_Help.pdf       IMET_textbook.pdf
SampleImages Sample Images which are useful for showing students the effects of each image editing technique without them requiring their own images	----SampleImages       Sample1.bmp       Sample2.bmp       Sample3.bmp

## Sample of planned SQL queries (not applicable)

There are no SQL queries in this solution as it is not based on a relational Database.

## Identification of storage media

The program folder containing the executable and other data will be stored on a USB drive but will also be distributed via optical disks. The program must either be run direct from the USB or the program folder must be copied to the computer, and then executed from there. Temporary images will be stored inside the program folders, but the user will have to save permanent images to their own computer

## Pseudocode

For all of the algorithms below, unless stated otherwise, the pixels must be broken down into their separate colours before any kernel is applied

### Sobel Edge Detection

The Sobel algorithm uses 2 3x3 kernels, one for the y-direction one for the x-direction. They are described by <http://www.pages.drexel.edu/~weg22/edge.html>, which cites each one in turn.

```

/* 3x3 GX Sobel mask.  Ref:
www.cee.hw.ac.uk/hipr/html/sobel.html */
GX[0][0] = -1; GX[0][1] = 0; GX[0][2] = 1;
GX[1][0] = -2; GX[1][1] = 0; GX[1][2] = 2;
GX[2][0] = -1; GX[2][1] = 0; GX[2][2] = 1;

/* 3x3 GY Sobel mask.  Ref:
www.cee.hw.ac.uk/hipr/html/sobel.html */
GY[0][0] = 1; GY[0][1] = 2; GY[0][2] = 1;
GY[1][0] = 0; GY[1][1] = 0; GY[1][2] = 0;
GY[2][0] = -1; GY[2][1] = -2; GY[2][2] = -1;

```

As shown in here the pixels are ‘scanned’ left to right, then bottom to top. The best way to implement a kernel based system is to store the grid positions of each pixel so that it is easy to carry out the operations using the surrounding pixels.

## Laplacian Edge Detection

The single 3x3 Laplacian kernel is described below

```

G[0,0] = -1 G[0,1] = -1 G[0,2] = -1
G[1,0] = -1 G[1,1] = 8 G[1,2] = -1
G[2,0] = -1 G[2,1] = -1 G[2,2] = -1

```

This kernel will be applied to the image in the same way as the Sobel algorithm except with only one kernel being applied once, not one kernel for each direction with both kernels being applied separately and “collated”.

## Smoothing

The basic algorithm is this:

```

G[1,1] ← (G[0,0] + G[0,1] + G[0,2] + G[1,0] + G[1,1] + G[1,2] + G[2,0] + G[2,1] + G[2,2]) div 9

```

Though not technically a kernel, this algorithm works like one as it is ‘sild’ over each pixel in turn like one.

## Noise Reduction

The basic algorithm is this:

```

G[1,1] ← median(G[0,0] + G[0,1] + G[0,2] + G[1,0] + G[1,1] + G[1,2] + G[2,0] +
G[2,1] + G[2,2])

```

Though not technically a kernel, this algorithm works like one as it is ‘sild’ over each pixel in turn like one just like Smoothing, and these algorithms have very similar effects as they both rely on taking an average. In order to take a median the pixels must first be sorted however. The sort I wish to use is the insertion sort:

```

for i ← to length(kernelarray-1)

```



```

begin
  value ← A[i]
  j ← i - 1
  done ← false
  repeat
    if A[j] > value then
      begin
        A[j + 1] ← A[j]
        j ← j - 1
        if j < 0 then
          done ← true
        end
      end
    else
      done ← true
    end
  until done
  A[j + 1] ← value
end
end

```

this is a standard algorithm and as such I have edited a version from [wikipedia](#).

### Grey-Scaling

```

Red ← PixelColor.Red
Green ← PixelColor.Green
Blue ← PixelColor.Blue
Gray ← Round(0.3 * Red + 0.6 * Green + 0.1 * Blue)

```

In the grayscale algorithm there is no kernel, instead each pixel must have its individual colour values extracted, multiplied by constants which represent how important each colour of light is to the human eye, and then the resulting weighted “average” of the 3 colour values describes the shade of gray that pixel must become.

### Red Eye Reduction

```

if RedEyeTolerance.min < RGBvalue < RedEyeTolerance.max then RGB value :=
777777

```

### General Algorithm

```

For Gx ← 1 to xlength-1
  For Gy ← 1 to ylength-1
    Begin
      ApplyKernel[Gx,Gy]
    End
  End
End

```

This algorithm goes through the image applying the kernel to each column of 3x3 pixel blocks in turn. In this version it omits the outer edge of pixels as 3x3 kernels do not work on

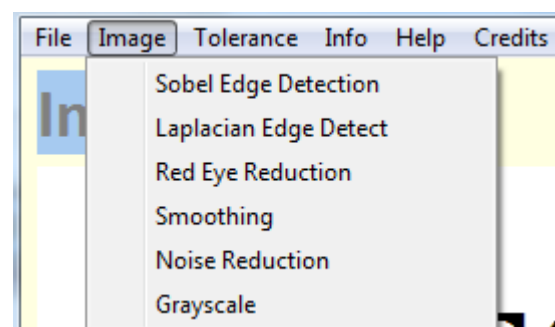
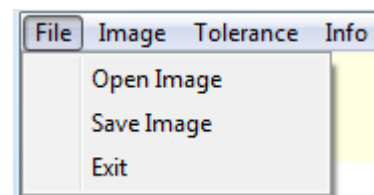
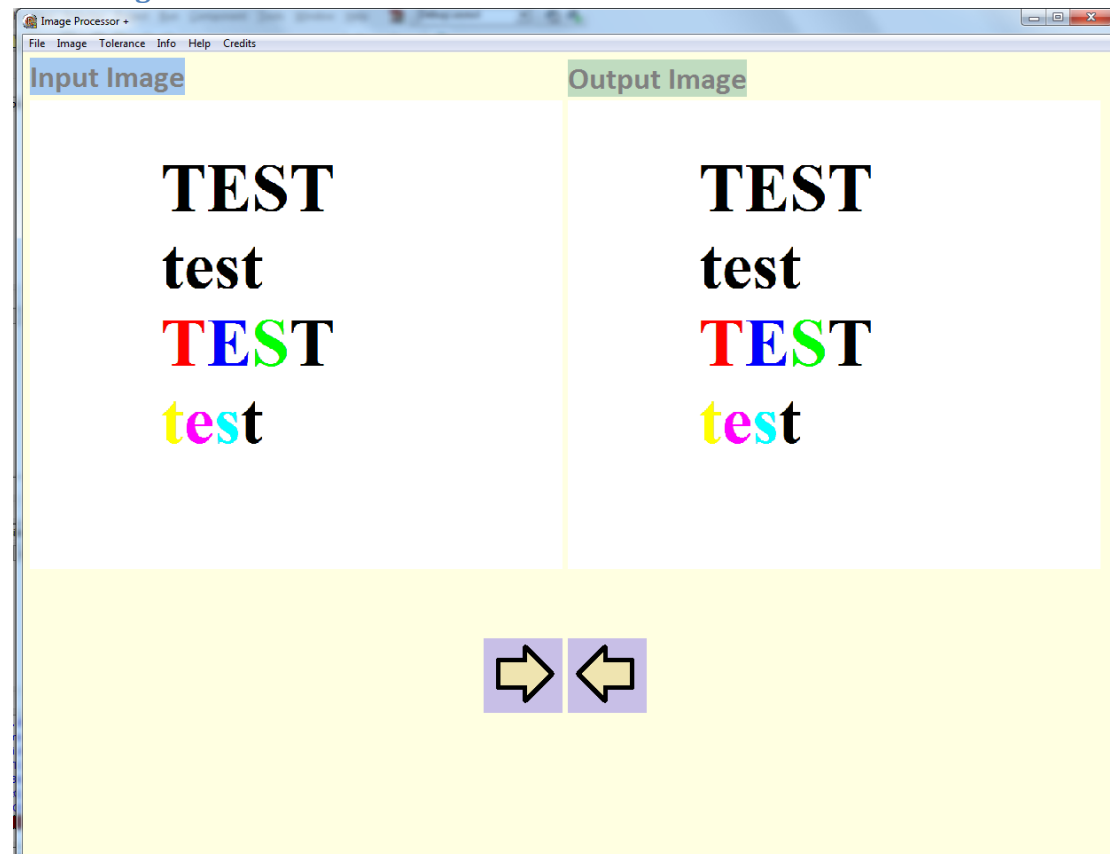
edges so they must be removed afterwards. For the Red Eye, Grayscale and Negative functions this will not be necessary as they do not use kernels.

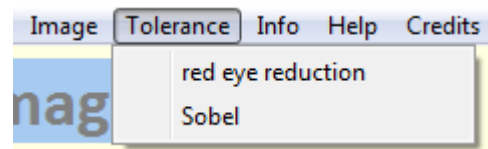
## Graphic User Interface Design

### Prototyping

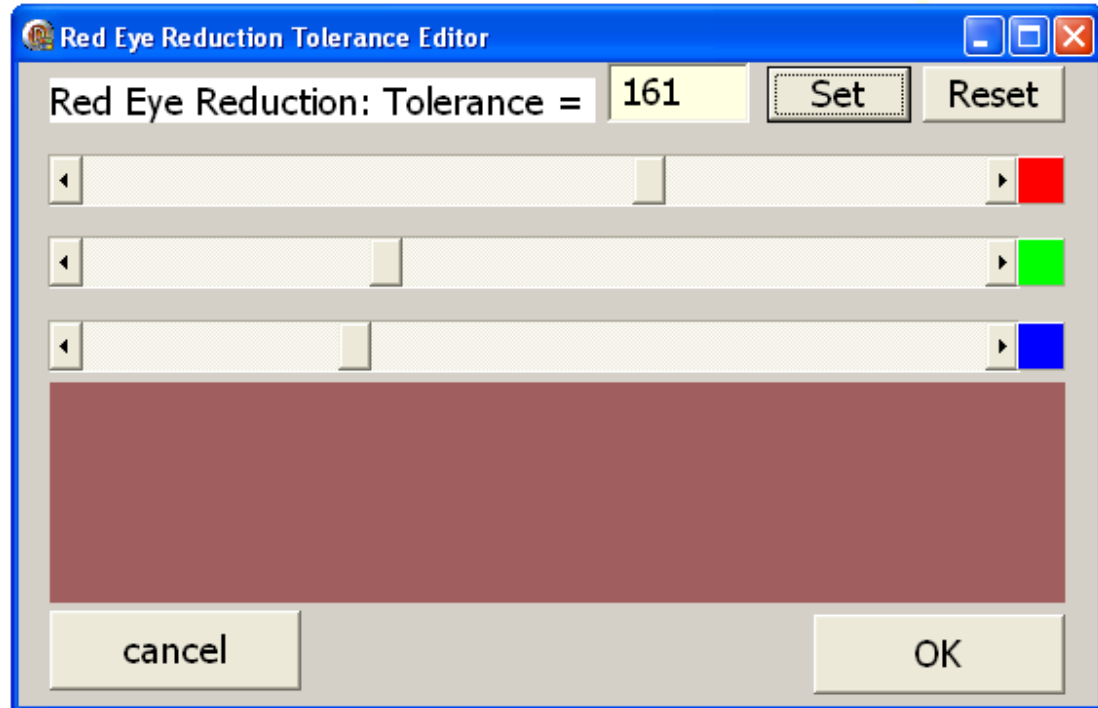
I have made prototype forms for each section of the GUI:

#### Main Program

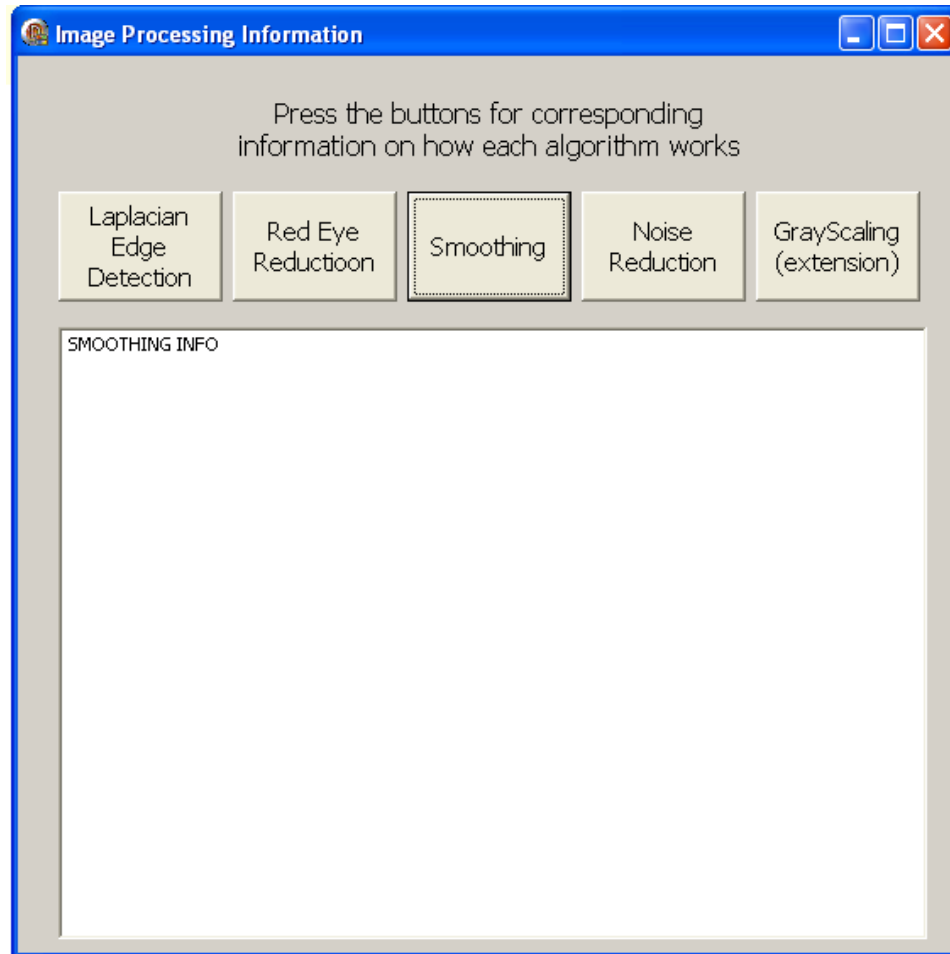




### Red Eye Reduction Tolerance Editor



## Educational Information Display



## Usability

The best way to present the system to make it easy to use would likely be drop down menus, the structure of which would be thus:

- File
  - Open Image
  - Save Image
  - Exit
- Edit
  - Undo (only one temporary image will be saved at a time so can only be carried out once)
  - Redo (only one temporary image will be saved at a time so can only be carried out once)
- Image
  - Sobel Edge Detection
  - Laplacian Edge Detection
  - Smoothing
  - Noise Removal
  - Red Eye Reduction
  - Grayscale
  - Negative
- Tolerances
  - Sobel

- Red Eye Reduction
- Info (will display Educational Information)
- Help
- Credits

There should then be a display of the original image and the edited image in the form below. Quick buttons for undo, redo, and help may be useful; however I have not included them in my prototype. I have however included a left and right button at the bottom of the page for setting the input image to the output image and vice versa on the main form. N.B. Open Image will always open to Input image and Save Image always saves the Output Image.

The tolerance editor for red eye reduction consists of three bars for editing each colour and a preview box displaying the chosen tolerance. The SET button previews the colour and enables the OK button after checking inputs are in range.

The “Virtual Textbook” will display written information about each of the relevant algorithms when prompted by button presses.

## Colour Scheme

Being an educational program the scheme should be fairly sober, and should follow usual text on background rules where text is dark on light and no similar colours are used for text and background. There is currently no need for high contrast support for the visually impaired.

## Input/output devices

The only user inputs and outputs supported by the program will be the computer monitor, keyboard and mouse. Due to the nature of the program there is no need for any other peripherals (n.b. It may be desirable for a print procedure to be included, but this is not essential, so I have currently omitted it).

## Error messages, feedback and dialogue boxes

Basic error messages such as “error: invalid input” and “error: file too large” should suffice for the system planned, as there are no complex inputs for the user to have to deal with. It must be necessary to inform the user if the image file they are attempting to open is not supported by the program, however it has not yet been decided which file types will be supported. However error messages for when there is a problem with the system must also be devised, which may include error codes which the user can reference against a help for example.

## Exits and actions

To prevent unwanted loss of data, “are you sure you want to overwrite this file”, “do you want to save changes before exiting” and “are you sure you want to exit” type error messages may be useful.

## Description of measures for planned system security

Because I have no plans to sell my system as a product I will not be adding any licensing keys into the product or any other systems for stopping duplication of the system. I also see no

need to protect the source code as it may be useful for advanced users to analyse it, and as such I may include a copy of the source code with my program so that interested users can analyse it. There will also be no sensitive data used in the program so I see no reason to add any data encryption into the system especially as it does not involve communicating data from one device to another. To these ends I feel there are few security measures I will need to include within the program

## Test Plan

I plan on testing the system with both white box and black box methods. To do this I will do testing during the creation of the program of course, but it is unfeasible to document all this, so I will describe the general strategy here. I will test each function and procedure I create in the program as I create them, making changes by looking at both the code and the output to iron out errors early on. I will also test how the system fits together as a whole as the program becomes more complete.

Once the program is completed I will then begin black box testing different areas of the program. The main areas will be:

- Image Editing Algorithms
  - I will test individual algorithms on simple images to check results against manually calculated results
  - I will test the effect of the software on larger, more realistic images, against subjective user expectations and the results of other image editing software
- User Inputs/Outputs
  - I will test the effects of erroneous, boundary and typical data on the system
- User Interface
  - I will test to make sure the User Interface works as intended
  - I will test the subjective usability of the interface according to the end user
- File Handling
  - I will test the opening/saving functionality of the program on different files, some of which may be erroneous

# Technical Solution

---

*The technical solution is included in Appendix A*

# Testing

---

*See evidence of testing (Appendix B) for screenshots*

*Appendix B is referenced to this table by the test code in the far left column*

Test	Data Entered	Reason for Choice	Expected Result	Actual Result	Corrective action required?
<b>Image Handling</b>					
1a	Attempt to open a *.JPG file	To test whether or not the program can handle invalid files which do not end in the *.bmp tag when opening images	Error Message: " "*.JPG" is not a valid bitmap"	Error Message: ""N:\My Picture\TEST.JPG" is not a valid bitmap"	None
1b	Attempt to open a *.GIF file	See 1a	Error Message: " "*.GIF" is not a valid bitmap"	Error Message: ""N:\My Picture\testgif.GIF" is not a valid bitmap"	None
1c	Attempt to open a *.PNG file	See 1a	Error Message: " "*.PNG" is not a valid bitmap"	Error Message: ""N:\My Picture\testpng.PNG" is not a valid bitmap"	None
1d	Attempt to open a *.BMP file	To test whether or not the program can handle valid Bitmap files	Open the Image Successfully and display on screen as Image1	Opens the Image Successfully and display on screen as Image1	None



Test	Data Entered	Reason for Choice	Expected Result	Actual Result	Corrective action required?
2	Valid Filename Check on save	To test whether or not the program can handle invalid files which do not end in the *.bmp tag when saving images	Automatically revert to '*.bmp' if an invalid file extension is entered	Automatically reverts to '*.bmp' if an invalid file extension is entered	None
3a	B/W bitmap image	If The Image is a b/w bitmap the program must tell the user that Red Eye Reduction cannot be applied to b/w bitmaps	Display Error Message "operation not valid for b/w bitmaps"	"I/O Error 105"	Bug Fix: "Writeln" not "Showmessage" " function used
3b	Valid BitsPerPixel value when performing Red Eye Reduction  RETEST	See 3a	See 3a	Displays Error Message  "operation not valid for b/w bitmaps"	None
4	Menu Click Image // Grayscale	To test whether the program correctly performs the grayscale function	Image2 := The Gray scaled version of Image1	Image2 := The Gray scaled version of Image1	None
5	Image Click "Image 3"  {a left pointing arrow}	To Test Whether the program correctly assigns the Bitmap Property of Image1 to the Bitmap Property of Image2	The picture property of image1 is assigned to be the picture property of image2	The picture property of image1 is assigned to be the picture property of image2	None

Test	Data Entered	Reason for Choice	Expected Result	Actual Result	Corrective action required?
6	Image Click "Image 5" {a right pointing arrow}	To Test Whether the program correctly assigns the Bitmap Property of Image2 to the Bitmap Property of Image1	The picture property of image2 is assigned to be the picture property of image1	The picture property of image2 is assigned to be the picture property of image1	None
7a	Menu Click: "Image//Noise Removal" using a checkerboard image(see evidence of testing)	To test whether the algorithm is applied correctly	No change to the image	No Change to The Image	None
7b	Noise Removal: Image with a single noise pixel	To test whether the algorithm is applied correctly	The noise pixel should become the colour of the median of it and the surrounding pixels (white)	The noise pixel became the colour of the median of it and the surrounding pixels (white)	None
7c	Noise Removal: Image with a 2x2 block of noise pixels on an otherwise white image	To test whether the algorithm is applied correctly	All the noise pixels should become white, and the white pixels remain white	All the noise pixels became white, and the white pixels remained white	None

Test	Data Entered	Reason for Choice	Expected Result	Actual Result	Corrective action required?
7d	Noise Removal:  Image with a 3x3 "cross" shape of black noise on white background	To test whether the algorithm is applied correctly	All but the middle noise pixel should become white and all other pixels remain white	All but the middle noise pixel became white and all other pixels remained white	None
7e	Noise Removal:  Image with a 3x3 block of noise on a white background	To test whether the algorithm is applied correctly	A 3x3 "cross" of black should remain in the centre of the image	A 3x3 "cross" of black remains in the centre of the image	None
8a	Grayscale:  Image all pixels gray	To test whether the algorithm is applied correctly	All pixels remain the same	All pixels remain the same	None
8b	Grayscale: a spectrum	Compare grayscaling function with that of another Image Processing Software (Paint.Net)	Both programs produce the same grayscaled image	Both programs produce the same grayscaled image	None
9a	Red Eye Reduction tolerance:  Memo1 entry  256	Boundary data	The text and scrollbar position should revert to 255	The text and scrollbar position reverted to 255	None

Test	Data Entered	Reason for Choice	Expected Result	Actual Result	Corrective action required?
9b	Red Eye Reduction tolerance:  Memo1 entry  356	Erroneous data	The text and scrollbar position should revert to 255	The text and scrollbar position reverted to 255	None
9c	Red Eye Reduction tolerance:  Memo1 entry  255	typical data	The text and scrollbar position should become 255	The text and scrollbar position became 255	None
9d	Red Eye Reduction tolerance:  Memo1 entry  120	typical data	The text and scrollbar position should become 120	The text and scrollbar position became 120	None
9e	Red Eye Reduction tolerance:  Memo1 entry  0	typical data	The text and scrollbar position should become 0	The text and scrollbar position became 0	None
9f	Red Eye Reduction tolerance:  Memo1 entry  'TEST'	Erroneous data	Error messages  "Invalid Input"  "ERROR: unknown"	Error messages  "Invalid Input"  "ERROR: unknown"	None

Test	Data Entered	Reason for Choice	Expected Result	Actual Result	Corrective action required?
9h	Red Eye Reduction tolerance:  Memo1 entry  '-1'	Boundary data	The text and scrollbar position should revert to 0	The text and scrollbar position reverted to 0	None
10a	Info: button press "Laplacian Edge Detection"	To test the button	Memo := "EDGE DETECTION INFO"*	Memo := "EDGE DETECTION INFO"*	None
10b	Info: button press "Red Eye Reduction"	To test the button	Memo := "RED EYE INFO"*	Memo := "RED EYE INFO"*	None
10c	Info: button press "Smoothing"	To test the button	Memo := "SMOOTHING INFO"*	Memo := "SMOOTHING INFO"*	None
10d	Info: button press "Noise Removal"	To test the button	Memo := "NOISE REMOVAL INFO"*	Memo := "NOISE REMOVAL INFO"*	None
10e	Info: button press "GrayScaling (Extension)"	To test the button	Memo := "GRAYSCALING INFO"*	Memo := "GRAYSCALING INFO"*	None
11a	Smoothing:  Salt and Pepper Noise	Compare smoothing function with that of another Image Processing Software (Paint.Net)	Both programs produce the same smoothed image	Both programs produce the same smoothed image	None

Test	Data Entered	Reason for Choice	Expected Result	Actual Result	Corrective action required?
11b	Smoothing: A noisy image	Compare smoothing function with that of another Image Processing Software (Paint.Net)	Both programs produce the same smoothed image	Both programs produce the same smoothed image	None
12a	Edge Detection: A normal Image	Compare edge detection function with that of another Image Processing Software (Paint.Net)	Both programs produce the same smoothed image	Both programs produce the same smoothed image	None
13a	Negative: A Spectrum	Compare Negative function with that of another Image Processing Software (Paint.Net)	Both programs produce the same smoothed image	Both programs produce the same smoothed image	None

*\*for testing purposes only*

# System Maintenance

---

## System Overview

My system consists of 4 Units, of which 3 are forms allowing users to modify images and view information on how the algorithms for image manipulation work.

The main form shows two images the input and the output image as well as a menu bar with 4 tabs for loading and saving images, manipulating the images, manipulating the algorithms, and for viewing educational information. Using file/save or file/load opens up a windows explorer dialogue box so that files can be browsed. The image tab has options for edge detection, red eye reduction, smoothing, noise reduction and greyscaling, where the algorithm is applied to the Input image on the left, and the result shown as the output image on the right. The tolerance tab has an option which will open up the secondary form for changing the tolerance of the red eye algorithm, and the info tab will open up the form displaying educational information about each algorithm. There are also left and right arrows at the bottom of the screen for setting the input image to the output image, or the output image to the input image respectively.

One of the secondary forms is used to edit the tolerance of the red eye reduction algorithm and consists of a slider which can be moved up and down to automatically change the number in a text box above it, which can be changed manually. When the set button is pressed the slider will change position to that indicated by the number if necessary and a preview of the colour tolerance will be displayed. If the value in the text box is an invalid input or out of the acceptable range an error message will be shown or the text box value will become the nearest in range value respectively. The Reset button reverts the tolerance to the default (144), and updates the slider and number as the set button would. The number in the text box represents the RED value of the RGB colour tolerance. The OK button sets the tolerance to the one chosen by the user, whilst the cancel button keeps the tolerance the same as it was before the form was opened. Both buttons also close the form.

The Info secondary form consists of 5 buttons and a large text box. There is one button for each of the 5 possible processes which can be carried out, with the greyscaling marked extension as this is not part of the Physics course but is relevant when using the edge detection algorithm. Pressing the buttons shows the corresponding information in the text box.

Unit 2 is the Unit in which all the code is contained for the actual processing of images, and the listings for functions and variables are provided later in the maintenance.

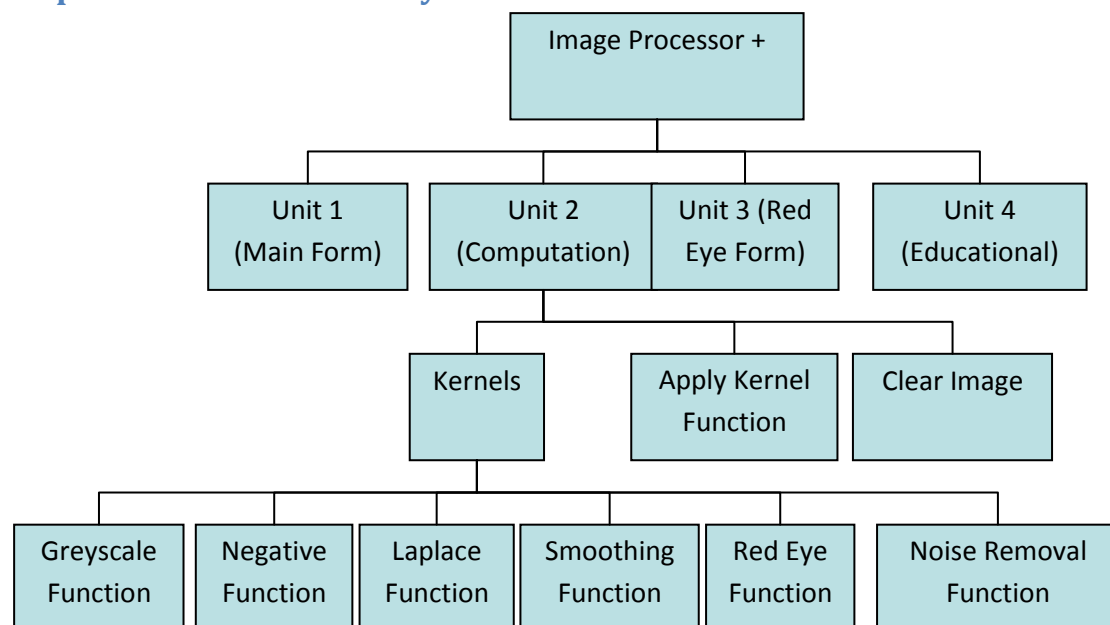
In the System the Units are broken up as such:

- UNIT 1 : The Main Form in which the images are displayed and user inputs are taken and processed
- UNIT 2 : The Unit with the code for all the image manipulation algorithms which actually edits the images. It also contains code for the Red Eye tolerance editing and stores some information for the Educational Tool
- UNIT 3 : The Form which handles inputs to the red eye tolerance editing
- UNIT 4: The Form which handles inputs and outputs for the Educational Tool

The most important part of the program is UNIT 2 which contains all the actual code for image editing. The Modular Structure of this part of the system is only subtly different from the original design. The Major Differences are:

- Time constraints led to Sobel Edge Detection being excluded

### Implementation Hierarchy Chart

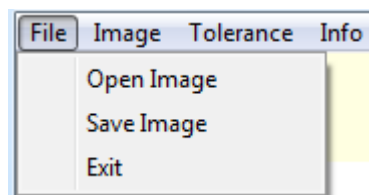
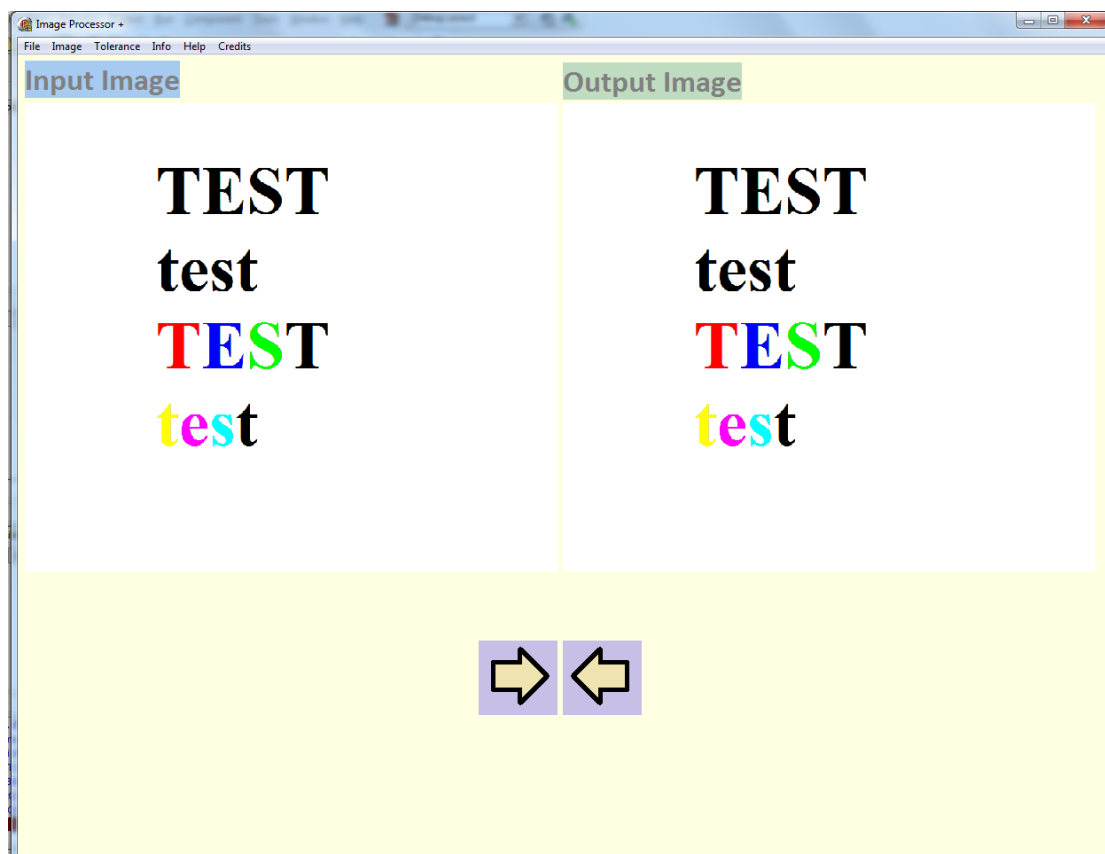


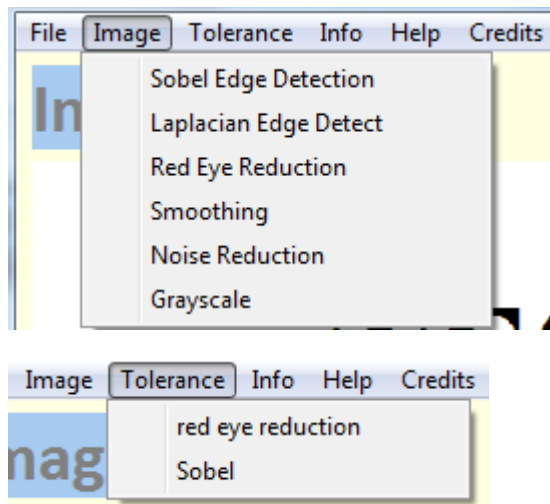


## GUI Implementation

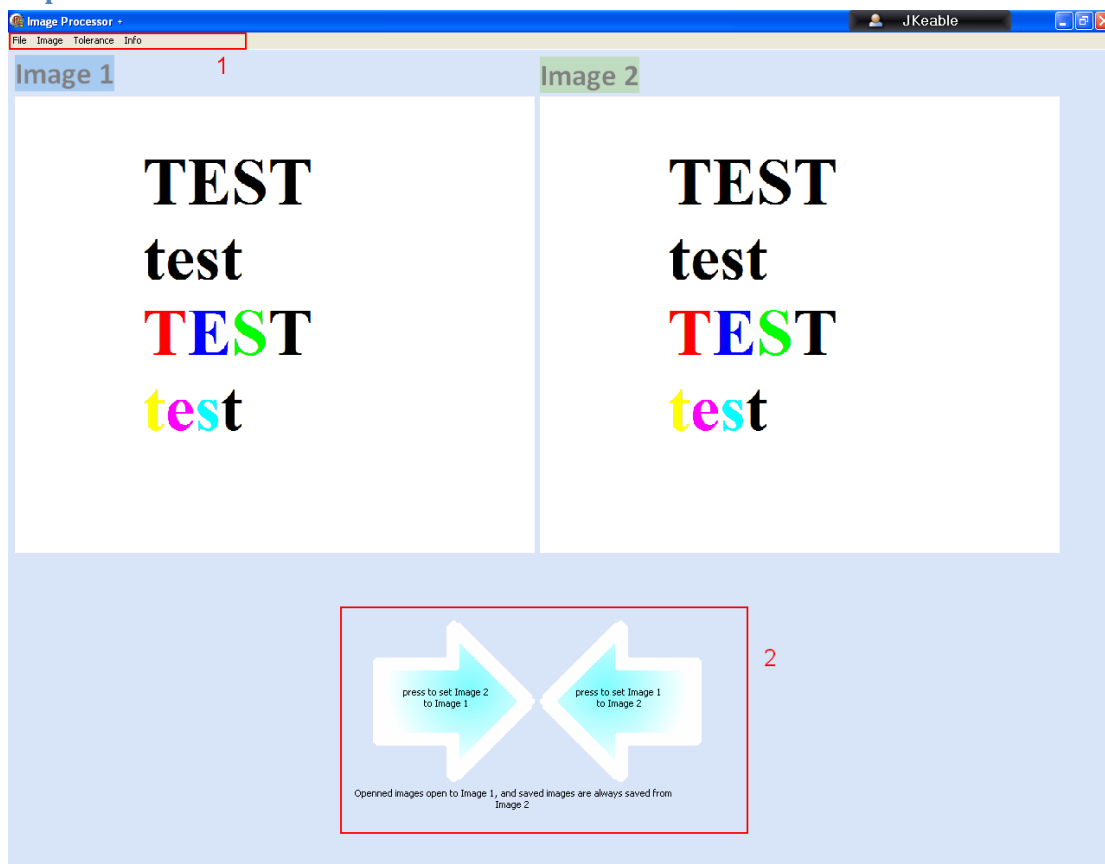
### Original Design

As was stated in the design, the system had to be user friendly, easy to navigate and relatively efficient to execute. As the system was implemented it became clear that certain changes to the GUI would need to be made in order to meet these aims, which I will discuss alongside the Implemented Forms.





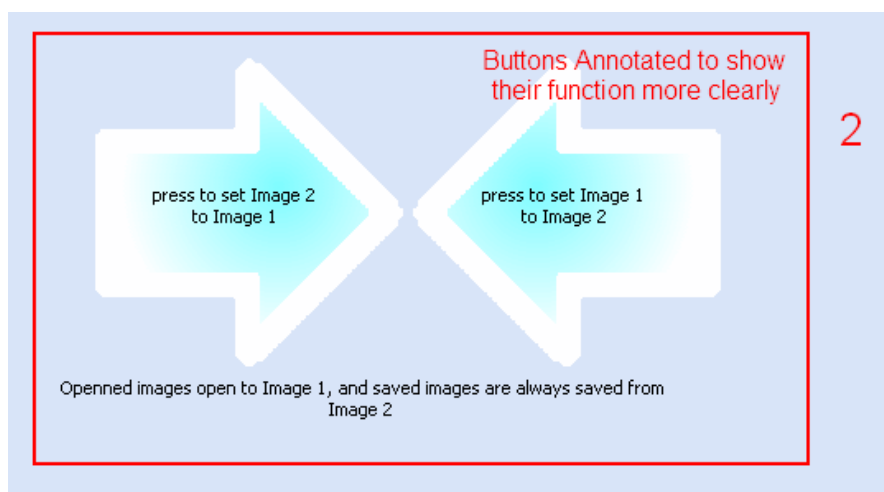
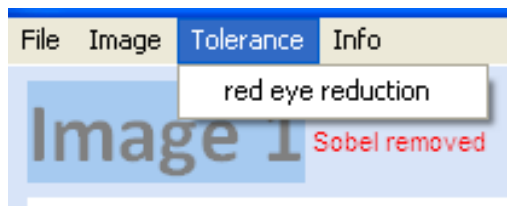
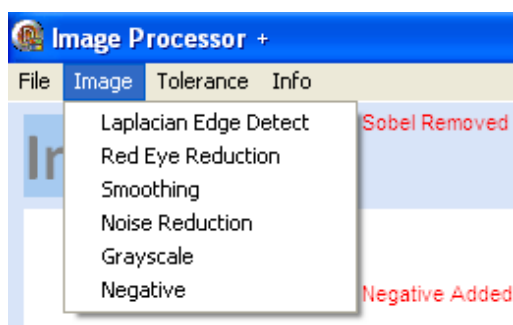
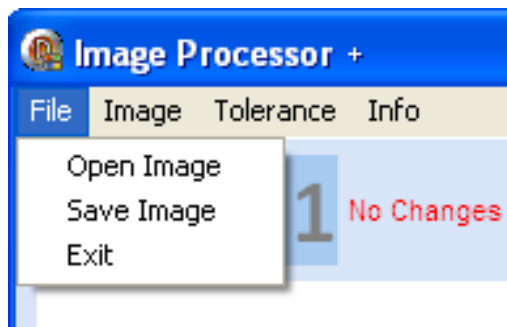
## Implemented



## Changes

- Colour changed from Cream to blue to give higher contrast between elements on the form
- Arrows made larger and their purpose is explained more clearly in that they Set each Image to each other
- All reference to the Sobel algorithm are removed

- Menu entries for the Negative facility are added
- Credits removed as deemed unnecessary
- Help removed from program menu and moved to the User Manual



## Detailed Algorithm Design

This section will relate the relevant algorithms to the coded procedures or functions.

n.b. in the pascal code '//TEXT' means an in line comment whilst '{TEXT}' and '(\*TEXT\*)' mean a block comment

Pseudocode Algorithm	Pascal code
<pre> Procedure ApplyKernel BEGINPROCEDURE  OutputImage ← InputImage  FOR i ← 1 TO ImagHeight DO FOR i ← 1 TO ImagWidth DO CASE 1: RGBImagePixel ← Laplace 2: RGBImagePixel ← Smoothing 3: RGBImagePixel ← RedEye 4: RGBImagePixel ← NoiseRemoval 5: RGBImagePixel ← Grayscale ENDPROCEDURE </pre>	<pre> procedure ApplyKernel(Width, Height: Integer; kernel : integer; Var Image1, Image2: TImage); VAR countw,counth : integer;  kernel3x3 : array [1..3,1..3] of integer;  count, count2: integer; median : integer; begin//procedure Image2.Picture := Image1.Picture; for counth := 1 to Height-2 do for countw := 1 to Width - 2 do begin//kernelTYPE case {begin} kernel of 1: begin //Laplace Image2.Picture.Bitmap.Canvas.Pixels [countw, counth] := Laplace(Image1,countw,counth) end;//laplace 2: begin //smoothing Image2.Picture.Bitmap.Canvas.Pixels [countw, counth] := smoothing(Image1,countw,counth); end;//smoothing 3: </pre>

	<pre> begin //red eye  //begin  image2.Picture.Bitmap.Canvas.Pixels [countw,counth]:=red_eye(Imagel,cou ntw,counth);  end;  4:  begin //Noise Removal  Image2.Picture.Bitmap.Canvas.Pixels [countw,counth] := Noise_removal(Imagel,countw,counth)  end;//Noise Removal  5:  begin //GrayScale  image2.Picture.Bitmap.Canvas.Pixels [countw,counth]:=Grayscale(imagel.P icture.Bitmap.Canvas.Pixels[countw, counth])  end;//GrayScale  End;//case  end;//kerneltype  end; </pre>
<pre> Function Grayscale BEGINFUNCTION  Red← RGBImagePixel REDVALUE  Green ←RGBImagePixel GREENVALUE  Blue ←RGBImagePixel BLUEVALUE  Gray← 30% Red + 60% Green + 10% Blue  ResultRed ←Gray  ResultGreen ←Gray  ResultBlue←Gray  ENDFUNCTION </pre>	<pre> function grayscale(PixelColor:longint): integer;  var  Gray, Red, Green, Blue: Byte;{the RGB values for the original pixel and the  value to which they will all be set after the algorithm has been applied}  begin  Red    := PixelColor;//red := to the first 8 bits of the rgb value  Green  := PixelColor shr 8;//blue := to the second 8 bits of the rgb value  Blue   := PixelColor shr 16;//green := to the third 8 bits of the rgb value </pre>

	<pre> Gray := Round(0.3 * Red + 0.6 * Green + 0.1 * Blue);{0.3,0.6,and 0.1 are the  approximate values for how much of each type of colour make up gray }  Result := RGB(Gray, Gray, Gray);  end;</pre>
<pre> Function RedEye BEGINFUNCTION  RedTolerance ← RedToleranceTextBox  GreenBlueTolerance ←255 - RedTolerance  Red ← RGBImagePixel REDVALUE  Green← RGBImagePixel GREENVALUE  Blue ← RGBImagePixel BLUEVALUE  If (Red &gt;RedTolerance) AND (Green&lt;GreenBlueTolerance) AND (Blue&lt;GreenBlueTolerance) THEN result ← Grey ELSE Result ← RGBImagePixel ENDFUNCTION</pre>	<pre> function red_eye(Image:Timage;x,y:integer):i nteger;  Var Red,Green,Blue :Byte;  redtolerance, gbtolerance :integer;  begin  try redtolerance := strtoint(unit3.Form1.memol.text);  except  begin  showmessage('ERROR: red tolerance invalid');  redtolerance := 144  end;  end;  gbtolerance := 255-redtolerance;  Red := image.Picture.Bitmap.Canvas.Pixels[ x,y] shr 0;  Green := image.Picture.Bitmap.Canvas.Pixels[ x,y] shr 8;  Blue := image.Picture.Bitmap.Canvas.Pixels[ x,y] shr 16;  if (Red &gt; redtolerance) and((Green &lt; gbtolerance) and (Blue &lt; gbtolerance))  then red_eye := strtoint('\$'+777777)  else result:=image.Picture.Bitmap.Canvas</pre>

	<pre> .Pixels[x,y]  end;//red eye </pre>
<pre> Function NoiseRemoval      Function GetMedian      BEGINUNCTION      REPEAT      swapped  false      FOR i ← 0 TO     length(list) - 2 DO      IF list[i-1] &gt; list[i]     THEN      swap( list[i-1], list[i]     )      swapped ← true      ENDIF      ENDFOR      UNTIL NOT swapped      Result ← list[4]      ENDFUNCTION  BEGINFUNCTION  SomeArray[1] ←RGBImagePixelCentre,  SomeArray[2] ← RGBImagePixelTop SomeArray[3] ← RGBImagePixelBottom  SomeArray[4] ← RGBImagePixelTopright  SomeArray[5] ← RGbImagePixelBottomright  SomeArray[6] ← RGBImagePixelTopleft  SomeArray[7] ← RGBImagePixelBottomLeft  SomeArray[8] ← RGBImagePixelRight </pre>	<pre> function noise_removal(Image:Timage;x,y:inte ger):integer;  function bubblesortMEDIAN (list : array of integer):integer;  Var  swapped: boolean;  i,temp: integer;  begin  repeat  swapped := false;  for i := 0 to 7 do  if list[i-1] &gt; list[i]then  begin  temp:= list[i];  list[i] := list[i-1];  list[i-1] := temp;  swapped := true  end  until not swapped;  Result := list[4]  end;//bubblesortmean  Var  UnSortedArray: array [1..9] of integer;  begin  UnsortedArray[1] := Image.Picture.Bitmap.canvas.pixels[ x,y];  UnsortedArray[2] := Image.Picture.Bitmap.canvas.pixels[ x,y+1]; </pre>

<pre> SomeArray[9] ← RGBImagePixelLeft  Result:=GetMedian(SomeArray)  ENDFUNCTION </pre>	<pre> UnsortedArray[3] := Image.Picture.Bitmap.canvas.pixels[ x,y-1];  UnsortedArray[4] := Image.Picture.Bitmap.canvas.pixels[ x+1,y+1];  UnsortedArray[5] := Image.Picture.Bitmap.canvas.pixels[ x+1,y-1];  UnsortedArray[6] := Image.Picture.Bitmap.canvas.pixels[ x-1,y+1];  UnsortedArray[7] := Image.Picture.Bitmap.canvas.pixels[ x-1,y-1];  UnsortedArray[8] := Image.Picture.Bitmap.canvas.pixels[ x+1,y];  UnsortedArray[9] := Image.Picture.Bitmap.canvas.pixels[ x-1,y];  //then sort  result := bubblesortmedian(unsortedarray)  end;//noiseremoval </pre>
<pre> Function Smoothing  BEGINFUNCTION  SumRed←0  SumGreen←0  SumBlue←0  SomeArrayRed[1] ←RGBImagePixelCentre RED,  SomeArrayRed[2] ← RGBImagePixelTop RED  SomeArrayRed[3] ← RGBImagePixelBottom RED  SomeArrayRed[4] ← RGBImagePixelTopright RED  SomeArrayRed[5] ← RGBImagePixelBottomright RED  SomeArrayRed[6] ← RGBImagePixelTopleft RED </pre>	<pre> function smoothing(Image:Timage;x,y:integer) :integer;  var  red,blue,green :byte;  count, sumr, sumg, sumb : integer;  kernelR, kernelg, kernelb : array [1..9] of byte;  begin  sumr := 0;  sumg := 0;  sumb := 0;  kernelR[1]:=Image.Picture.Bitmap.ca nvas.pixels[x,y];  kernelR[2]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y];  kernelR[3]:=Image.Picture.Bitmap.ca </pre>



<pre> SomeArrayRed[7] ← RGBImagePixelBottomLeft RED  SomeArrayRed[8] ← RGBImagePixelRight RED  SomeArrayRed[9] ← RGBImagePixelLeft RED  FOR i ← 1 to 9 do SumRed ← SumRed + SomeArrayRed[i]  ENDFOR  Red ← SumRed DIV 9  SomeArrayGreen[1] ←RGBImagePixelCentre GREEN,  SomeArrayGreen[2] ← RGBImagePixelTop GREEN  SomeArrayGreen[3] ← RGBImagePixelBottom GREEN  SomeArrayGreen[4] ← RGBImagePixelTopright GREEN  SomeArrayGreen[5] ← RGbImagePixelBottomright GREEN  SomeArrayGreen[6] ← RGBImagePixelTopleft GREEN  SomeArrayGreen[7] ← RGBImagePixelBottomLeft GREEN  SomeArrayGreen[8] ← RGBImagePixelRight GREEN  SomeArrayGreen[9] ← RGBImagePixelLeft GREEN  FOR i ← 1 to 9 do SumGreen ← SumGreen + SomeArrayGreen[i]  ENDFOR  Green ← SumGreen DIV 9  SomeArrayBlue[1] ←RGBImagePixelCentre BLUE,  SomeArrayBlue[2] ← RGBImagePixelTop BLUE  SomeArrayBlue[3] ← RGBImagePixelBottom BLUE </pre>	<pre> nvas.pixels[x,y+1];  kernelR[4]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y+1];  kernelR[5]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y];  kernelR[6]:=Image.Picture.Bitmap.ca nvas.pixels[x,y-1];  kernelR[7]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y-1];  kernelR[8]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y+1];  kernelR[9]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y-1];  for count := 1 to 9 do  begin  sumr := sumr + kernelR[count]  end;  Red := sumr div 9;//smoothred  kernelg[1]:=Image.Picture.Bitmap.ca nvas.pixels[x,y]shr 8;  kernelg[2]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y]shr 8;  kernelg[3]:=Image.Picture.Bitmap.ca nvas.pixels[x,y+1]shr 8;  kernelg[4]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y+1]shr 8;  kernelg[5]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y]shr 8;  kernelg[6]:=Image.Picture.Bitmap.ca nvas.pixels[x,y-1]shr 8;  kernelg[7]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y-1]shr 8;  kernelg[8]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y+1]shr 8;  kernelg[9]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y-1]shr 8;  for count := 1 to 9 do  begin </pre>
--	--

<pre> SomeArrayBlue[4] ← RGBImagePixelTopright BLUE  SomeArrayBlue[5] ← RGBImagePixelBottomright BLUE  SomeArrayBlue[6] ← RGBImagePixelTopleft BLUE  SomeArrayBlue[7] ← RGBImagePixelBottomLeft BLUE  SomeArrayBlue[8] ← RGBImagePixelRight BLUE  SomeArrayBlue[9] ← RGBImagePixelLeft BLUE  FOR i ← 1 to 9 do SumBlue ← SumBlue + SomeArrayBlue[i] ENDFOR  Blue ← SumBlue DIV 9  ResultRed ← Red  ResultGreen ← Green  ResultBlue ← Blue </pre>	<pre> sumg := sumg + kernelg[count] end;  green := sumg div 9;//smoothgreen  kernelb[1]:=Image.Picture.Bitmap.ca nvas.pixels[x,y]shr 16;  kernelb[2]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y]shr 16;  kernelb[3]:=Image.Picture.Bitmap.ca nvas.pixels[x,y+1]shr 16;  kernelb[4]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y+1]shr 16;  kernelb[5]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y]shr 16;  kernelb[6]:=Image.Picture.Bitmap.ca nvas.pixels[x,y-1]shr 16;  kernelb[7]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y-1]shr 16;  kernelb[8]:=Image.Picture.Bitmap.ca nvas.pixels[x-1,y+1]shr 16;  kernelb[9]:=Image.Picture.Bitmap.ca nvas.pixels[x+1,y-1]shr 16;  for count := 1 to 9 do begin sumb := sumb + kernelb[count] end;  blue := sumb div 9;//smoothblue  result := RGB(red,green,blue);//smoothcolour end;  ENDFUNCTION </pre>
<pre> ON BUTTON 'GrayScale' CLICK DO ApplyKernel(Grayscale) </pre>	<pre> procedure TForm2.GrayScale1Click(Sender: TObject);  begin  ApplyKernel(Image1.Picture.Bitmap.W idth, Image1.picture.Bitmap.Height,5, </pre>

	<pre> Image1, Image2)  end;</pre>
<pre> ON BUTTON 'RedEyeTolerance' CLICK  DO VIEW RedToleranceForm  ON RedToleranceForm BUTTON 'OK' click  DO OldTolerance ← NewTolerance FROM RedToleranceForm</pre>	<pre> procedure TForm2.redeyereduction2Click(Sender : TObject);  begin unit3.Form1.Visible := true;  unit3.Form1.Button1Click(Sender);  unit3.olddtolerance := strtoint(unit3.Form1.Memol.text)  end;</pre>
<pre> ON BUTTON 'SET' CLICK  ScrollBar1Pos ← TextBox  PreviewImageRED←TextBox  PreviewImageGREEN←255 - TextBox  PreviewImageBLUE←255 - TextBox</pre>	<pre> procedure TForm1.Button1Click(Sender: TObject);  var x,y : integer;  begin if Memol.text &lt;&gt; '' then Begin  Try ScrollBar1.Position := strtoint(memol.Text)  Except  ShowMessage('Invalid Input');  End;  Try  for y := 0 to Image1.height do for x := 0 to Image1.width do  image1.Picture.Bitmap.Canvas.Pixels [x,y]  := RGB(strtoint(memol.Text),255- strtoint(Form1.memol.text),255- strtoint(Form1.memol.text)) Except  showmessage('ERROR: unknown')  End  End;</pre>

## Annotated Listings

### Procedures/Functions

#### Unit2

<b>function grayscale(PixelColor:longint): integer;</b>  <i>transforms a single pixel to gray, where PixelColour contains Red,Green and Blue values which can be separated by msb.</i>
<b>function Laplace(Image:TImage;x,y:integer):integer;</b>  <i>applies the Laplacian Edge Detection algorithm to a 3x3 kernel based on a central pixel</i>
<b>function smoothing(Image:TImage;x,y:integer):integer;</b>  <i>applies the smoothing algorithm to a 3x3 kernel based on a central pixel</i>
<b>function red_eye(Image:TImage;x,y:integer):integer;</b>  <i>checks to see if a single pixel is 'red' based on a tolerance and if the pixel is red will transform it to a medium shade of grey</i>
<b>function noise_removal(Image:TImage;x,y:integer):integer;</b>  <i>will apply the noise reduction algorithm to a 3x3 kernel based on a central pixel</i>
<b>function bubblesortMEDIAN (list : array of integer):integer;</b>  <i>a subfunction of 'noise_removal' which bubble sorts the kernel then finds the median value once passed the kernel by the parent function</i>
<b>function ClearImage(Image:TImage):TImage;</b>  <i>clears a specified image to white</i>
<b>procedure ApplyKernel(Width, Height: Integer; kernel : integer; Var Image1, Image2: TImage);</b>  <i>Is initialized by a button click which also gives it the algorithm to apply, and the function then steps through each pixel of the image (except the outer 1 pixel border as a 3x3 kernel cannot be applied to these pixels) and calls the specified algorithm which will apply itself to each pixel and pass back the new value for each pixel. This procedure then sets each output image pixel to the result from the called function as it steps through the pixels.</i>
<b>procedure TForm2.Grayscale1Click(Sender: TObject);</b>  <i>initializes the AppyKernel Procedure with the grayscale function</i>

**procedure TForm2.Image3Click(Sender: TObject);**

*sets input image to output image*

**procedure TForm2.Image5Click(Sender: TObject);**

*sets input image to output image*

**procedure TForm2.Info1Click(Sender: TObject);**

*makes the educational information form visible*

**procedure TForm2.Laplacianedgedetect1Click(Sender: TObject);**

*initializes the AppyKernel Procedure with the Laplace function*

**procedure TForm2.Load1Click(Sender: TObject);**

*opens a windows dialogue box for opening image files*

**procedure TForm2.NoiseReduction1Click(Sender: TObject);**

*initializes the AppyKernel Procedure with the noise\_reduction function*

**procedure TForm2.RedEyeReduction1Click(Sender: TObject);**

*initializes the AppyKernel Procedure with the red\_eye function*

**procedure TForm2.redeyereduction2Click(Sender: TObject);**

*makes the red eye reduction tolerance editor form visible allowing the user to edit the tolerance*

**procedure TForm2.Smoothing1Click(Sender: TObject);**

*initializes the AppyKernel Procedure with the smoothing function*

**procedure TForm2.SaveImage1Click(Sender: TObject);**

*opens a windows dialogue box for saving image files*

### Unit3

**procedure TForm1.Button1Click(Sender: TObject);**

*sets the scrollbars value to the value in the text box if possible, and sets the color of the preview image to the tolerance indicated by that number*

**procedure TForm1.Button2Click(Sender: TObject);**

*sets the tolerance to the one selected by the user, and hides the tolerance editor form*

**procedure TForm1.Button3Click(Sender: TObject);**

*cancels the change to the tolerance, so the old tolerance is kept, and hides the tolerance editor form*

**procedure TForm1.Button4Click(Sender: TObject);**

*resets the text box value in the tolerance editor to '144', and has the same effect as the 'SET' button click afterward*

**procedure TForm1.FormCreate(Sender: TObject);**

*creates the redtolerance editor form*

**procedure TForm1.ScrollBar1Change(Sender: TObject);**

*sets the text box value to that indicated by the scroll bar when it is moved*

#### Unit4

**function assigninfo(code:char):string;**

*gets the educational text from text files so that it can later be placed in the form*

**function getinfo(code:char):string;**

*using the assigninfo function this actually places text in the text boxes on the form*

### Constants/Variables

#### Unit2

#### Variables

Identifier	Type	Description
Red	Byte	An 8 bit value used in the Grayscale function to store the Red part of the colour of a pixel
Blue	Byte	An 8 bit value used in the Grayscale function to store the Blue part of the colour of a pixel
Green	Byte	An 8 bit value used in the Grayscale function to store the Green part of the colour of a pixel
Gray	Byte	An 8 bit value used in the Grayscale function to store the gray value calculated from a catenation of the three RGB values of a pixel
Red	Byte	An 8 bit value used in the Negative function to store the Red part of the colour of a pixel

Identifier	Type	Description
Blue	Byte	An 8 bit value used in the Negative function to store the Blue part of the colour of a pixel
Green	Byte	An 8 bit value used in the Negative function to store the Green part of the colour of a pixel
Red	Byte	An 8 bit value used in the Smoothing function to store the Red part of the colour of a pixel
Blue	Byte	An 8 bit value used in the Smoothing function to store the Blue part of the colour of a pixel
Green	Byte	An 8 bit value used in the Smoothing function to store the Green part of the colour of a pixel
count	Integer	An Integer used by the smoothing function to add up the numbers in the array kernel
sumr	Integer	An Integer used by the smoothing function as the sum of the 9 red values for the pixels in the 3x3 kernel
sumb	Integer	An Integer used by the smoothing function as the sum of the 9 blue values for the pixels in the 3x3 kernel
sumg	Integer	An Integer used by the smoothing function as the sum of the 9 green values for the pixels in the 3x3 kernel
kernelR	Array[1...9] of Integer	An array to store the 9 Red values for the pixel kernel in the smoothing function
kernelG	Array[1...9] of Integer	An array to store the 9 Green values for the pixel kernel in the smoothing function
kernelB	Array[1...9] of Integer	An array to store the 9 Blue values for the pixel kernel in the smoothing function
Red	Byte	An 8 bit value used in the Red_eye function to store the Red part of the colour of a pixel
Blue	Byte	An 8 bit value used in the Red_eye function to store the Blue part of the colour of a pixel
Green	Byte	An 8 bit value used in the Red_eye function to store the Green part of the colour of a pixel
redtolerance	integer	A value used by the red_eye function to test whether or not the red RGB value of a pixel is large enough for it to be red

Identifier	Type	Description
gbtolerance	integer	A value used by the red_eye function to test whether or not the green and blue RGB values for a pixel are small enough for it to be considered red
count	integer	An integer used by the BubbleSortMEAN function to bubblesort the kernel of RGB values
Count2	integer	An integer used by the BubbleSortMEAN function to bubblesort the kernel of RGB values
temp	Integer	An integer used by the BubbleSortMEAN function to bubblesort the kernel of RGB values to store temporary values when swapping values in the list
UnSortedArray	Array[1..9]	The kernel in array form used by the NoiseReduction used to find the median
countw	Integer	Used by ApplyKernel to step through the rows of pixels
counth	integer	Used by ApplyKernel to step through the columns of pixels



# User Manual

---

The User Manual is included in Appendix C

# Appraisal

---

## Introduction

Once the system is complete, it is important to analyse how the finished product compares to the original brief agreed by the customer and producer. To this end I have made my own judgements on the performance of the system against the original “SMART” objectives, as well as collecting feedback from the customer (proof of which is included in the appendix) and discussed what the implementation has succeeded in doing and what it has failed to do. I have also discussed what possible changes and additions could be made to the system and how easy it would be to expand the project.

## Comparison of Implementation with “SMART” objectives

In the analysis section a number of “SMART” objectives were laid out for the system. These objectives are compared to the finished system here, explaining to what extent they were met, to what extent they were not met, and why.

### *1.1.1 The Laplacian method must be implemented*

The laplacian method was included in the system. It was decided that the 3x3 pixel kernel method was to be used as opposed to the 5 pixel ‘+’ kernel method as this allows for edge detection in 4 directions as opposed to two.

### *1.1.2 A Sobel method could be implemented*

The Sobel method was not included in the system. It was decided not to include this as it was not part of the AS physics course, so time and resources would be better spent on algorithms that were in the course, such as the Laplacian algorithm.

### *1.1.3 For the sobel method the user should be able to change for what value it is said an edge has been detected.*

N/A (see 1.1.2).

### *1.1.4 The user must be able to choose whether the edges are white on black, or black on white.*

Originally this was to be applicable to the sobel method, however this was not implemented so this functionality was not relevant in the way it was meant to have been. However, it became clear that for some users the laplacian procedure, which is always white on black edges, caused an unclear result for some users. As a result an algorithm that would flip the colours of an image, i.e. a negative function, was added. As with the greyscale procedure this could be used in conjunction with or independently of the edge detection algorithm

### *1.1.5 The system must be able to convert an image to greyscale before any edge detection is carried out*

A separate greyscale function was added to the system which can be used independently of the edge detection algorithm if necessary.

### *1.1.6 The conversion of images to edge detection images should take no longer than 30 seconds*

When executed on hardware that met reasonable specifications using images within a reasonable size this was achieved. On very sub par hardware or very large Images this sometimes may not be the case, but these cases should not occur to the normal user.

### ***1.2.1 The system should be able to smooth images using the mean technique***

This method was implemented into the program as it is described in the AS physics course.

### ***1.2.2 It could be possible to enter how many times the smoothing process should be repeated, if any.***

It was decided that as the program was designed for educational use not for commercial image editing that this repeating function was not essential so it was not implemented in the release. It is possible to manually apply the algorithm multiple times to the same image by applying the algorithm once, then executing the output image of the input image box, and then applying the algorithm again however, which is useful to see compare the difference on the strength of smoothing achieved by repeating the algorithm. In future it would be fairly simple to implement the repeated smoothing functionality if the situation changed.

### ***1.2.3 It should take no longer than 30 seconds for images to be smoothed***

When executed on hardware that met reasonable specifications using images within a reasonable size this was achieved, however the algorithm did complete slower in comparison to the other algorithms. On very sub par hardware or very large Images this sometimes may not be the case, but these cases should not occur to the normal user, though even relatively small images could take long times compared to other algorithms.

### ***1.3.1 The system should be able to remove noise using the median method***

This method was implemented into the program as it is described in the AS physics course.

### ***1.3.2 There should be as little loss of data as possible during noise removal***

From testing it appeared to the human eye that it was mainly noise that was removed from the image, and the original appearance of the image was maintained. The only exception is one pixel thick drawn lines, but this should not normally occur in reasonably high quality photographs

### ***1.3.3 It could be possible to enter how many times the noise removal process should be repeated, if any.***

This was not implemented for the same reasons as it was not implemented for smoothing (see 1.2.2)

### ***1.3.4 It should take no longer than 30 seconds for noise to be removed from an image***

When executed on hardware that met reasonable specifications using images within a reasonable size this was achieved. On very sub par hardware or very large Images this sometimes may not be the case, but these cases should not occur to the normal user.

#### ***1.4.1 It should be possible for the system to convert red pixels in an image to grey pixels***

This was implemented into the program using a 50% grey value to convert pixels above some red threshold into. The red threshold is a combination of a >Red and <Green, <Blue check.

#### ***1.4.2 It should be possible for the user to change the red tolerance of the process***

A separate form was implemented into the program to facilitate this. The form allows users to use a slide bar or text box to enter a value for the red tolerance, and preview what this tolerance is before setting it.

#### ***1.4.3 It could be possible for the user to select an area for which red eye is to be reduced***

Along with all other selection tools, this was not implemented into the final product. The reason for this is that these features would require time to implement that was thought to be better spent on improving the quality of the rest of the program. As an educational tool the program still shows very well how red eye reduction works, and so functionality more suited to users who wanted a program for photo editing was not incorporated.

#### ***1.4.4 It should take no longer than 30 seconds for red eye to be reduced.***

When executed on hardware that met reasonable specifications using images within a reasonable size this was achieved. On very sub par hardware or very large Images this sometimes may not be the case, but these cases should not occur to the normal user.

### ***2.1 The user should be able to save an edited image without overwriting the original***

A save feature where the file name can be chosen using a dialogue box was implemented, and so images can be saved without overwriting the original.

### ***2.2 The user could be able to undo changes to an image for at least 10 image edits***

Instead of an undo function where multiple 'History' images had to be stored a two image system was implemented into the system. The image on the left is 'Image 1', the input image, and is the image that the program will always open to. The Image on the right is 'Image 2', the output image, and is the image that the program will always save to. There are right and left arrows at the bottom of the screen allowing the user to set either image to the other image. This system allows the user to see a direct before and after comparison, as well as keeping the original image stored in the program. I feel this helps the system function as an educational tool, and also makes it more compact as only two images are stored at any one time.

### ***2.3 The user should be able to revert to the original image***

This idea was replaced by the system described above (see 2.2)

### ***2.4 There could be certain selection tools***

As is described in the red eye reduction objective (see 1.4.3), the selection tool feature was omitted as it was thought this would be better suited to a commercial image editing package as opposed to an educational image processing system.

### ***2.5 The user could be able to preview changes before they are made***

This aim was also replaced by the two image solution (see 2.2)

### ***3.1 It should not crash or have problems frequently***

From testing, the system did not appear to crash or have errors usually, however very large images could cause the system to become unresponsive, especially on lesser hardware. This was dealt with by specifying minimum system requirements, and a maximum recommended image size

### ***3.2 It should be clear where different tools are***

Feedback from the end-user suggests that this is generally the case, however until the system is distributed to a larger user base, this objective may be unclear as to its completeness. This being said, no glaringly obvious flaws appeared in the end-user feedback.

### ***3.3 It should not be possible to easily lose an edited image or delete an existing one***

The two image system makes it difficult to lose an edited image within the program, however on exit of the program there is no prompt as to whether or not you wish to save your work. Because the system contains sample images and is designed to be used as an educational tool, it was decided not to include major safeguards against loss of data, however it may be desirable to implement these in the future

### ***3.4 The system must contain brief summaries of which each function does***

A separate form contains all educational information in the program. Each algorithm (except 'Negative', which was not in the original brief) has its own entry of text in this resource, and users can have the educational form open along with the main form for image editing.

## ***4 The system should not damage other systems***

Every effort was made to make sure the finished system did not contain anything that would unintentionally damage other systems, or any viruses that may have leaked into the distributable zipped folder. To date neither I nor my end user have had our systems at all damaged by the program.

## ***5 The System should be compact enough to be sent by e-mail or to be put on a CD-ROM, DVD, or Flash memory Stick (it does not have to fit onto a Floppy Disk).***

The size of the whole system including the executable, user manual and sample images is very easily small enough to fit onto a memory stick, CD or DVD, and most email clients allow attachments to be sent which are the sizes of the zipped file.

## **End User Feedback**

And authenticated photocopy of notes made by the end user on the system can be found in *Appendix D*

## **Analysis of User Feedback**

I used a feedback form as well as a copy of the finished system with the user manual. Mr. Cottrell used the system and completed the feedback form. Having analyzed his responses I have summarized the main points, and my responses to them.

## Imaging

Feedback was generally positive about the image manipulation techniques, however there were some criticisms.

### Laplacian Edge Detection

Unfortunately little feedback was given about this part of the program, though all feedback that was given seemed positive. It seems safe to assume that the lack of criticism about this algorithm shows it worked correctly.

### Smoothing

Much like with edge detection the user did not comment extensively on this, however again the limited feedback was positive. It is not entirely surprising seeing as the smoothing effect can be difficult to detect.

### Noise Removal

In this case more feedback was given. The algorithm was reported to work correctly and quickly, but had varied effect. The fact that the noise removal process worked better on some images than others is more due to the algorithm than the implementation of it, as though it is excellent at detecting salt and pepper noise, more subtle noise cannot be detected.

### Red Eye Reduction

Mr. Cottrell was generally pleased with the functionality of the algorithm, confirming that it was fast enough and effective, and that the tolerance editor worked correctly. However he also commented that the process did not remove enough red from the image. This may have been because the red tolerance was set to high; however it seems more likely that the single tolerance slider meant that certain tones of "red" were difficult to remove. Using three sliders for Red, Green and Blue might improve this problem.

## User Interface and Usability

The main feeling I have taken from the comments here is that the user interface did what it should do well, but was a little spartan in places. There was also one technical issue, which I will discuss here also

The opinion on the file handling of the program was that it worked correctly, but there was some confusion over which image was being saved. Though the functionality of basic image manipulation was "limited" according to Mr. Cottrell it also did what it needed to well. In future maybe more tools could be added. Mr. Cottrell also thought that the imaging tools were laid out well and were easy to use. In terms of the interface when an algorithm was being applied he did suggest that the program should have some time of indicator, such as an egg timer icon, to show it was in activity.

The one major concern with the user interface was that the End User was unable to access the educational information due to an error accessing the text files which store this information. Unfortunately this means he was also unable to comment on the usefulness of this information. I believe this problem may be as a result of incorrect installation, as I

encountered no such problems in testing, and to avoid it happening in future it would be prudent to improve the installation instructions or provide an installer wizard.

## Possible Extensions

Based on user feedback and my own thoughts on the finished system I see a number of improvements that could be made in different areas.

### Imaging

As the main part of the program it is important that the algorithms are effective, and for the most part they were, but there are some ways that they could be improved:

#### Improved red eye tolerance editor

It was suggested by the end user that more “Red” needed to be removed from the image. The point of the tolerance editor is so that the end user can change how powerful the algorithm is, however this seems to have failed so steps could be made to remedy this. Currently only the red value can be changed in such a way that if a RGB pixel colour’s Red value is above this value, and the G and B channels less than the equivalent Blue and Green values, it is considered “red”. Instead the user could choose a value above which the red channel must be for a colour to be accepted, and the green and blue values the respective channels would have to be less than for the colour to be accepted, all as separate sliders to be combined into some RGB tolerance. This would allow the user to have more control over the tolerance so they could remove the shades of red they wanted to.

#### Improved file handling interface

According to Mr. Cottrell there was so confusion over which file was being saved from or opened to, and as such I feel adding in more clear instructions on how image files are handled in the program may be needed. I also personally feel that adding in support for more file types other than bitmaps would be useful for users who do not understand file types as well as others.

#### Improved installation

Though Mr. Cottrell was able to install the software, due to issues with the educational information in the program not loading correctly, the installation instructions for the program might be improved. Alternatively an installation wizard for the program could be created to make it easy for less advanced users than Mr. Cottrell to use the system.