University of Dublin

TRINITY COLLEGE

Book Finder

Jason Hogan
B.A.(Mod.) Computer Science
Final Year Project May 2017
Supervisor: Dr. Kenneth Dawson-Howe

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

05/05/2017

Name

Date

Acknowledgements

I would like to thank Dr. Kenneth Dawson-Howe for all his help, insight, and guidance over the course of this project.

I would like to thank Hodges Figgis for allowing me to test my application in their bookshop and publish the resulting images.

Abstract

Bookshops and libraries often contain bookcases where hundreds of books are displayed. It can be tedious and slow for a human to pick out one specific book from a large bookcase, especially in academic libraries where many of the book spines are similar in appearance to one another.

This report presents a project whereby software was developed for finding specific books in images of bookshelves. The book-finding system uses computer vision techniques to identify shelves in an image, and search each shelf for the given book. An iOS application was developed alongside the book-finding software, in order to deploy the software in a useful and usable format. The application uses the smartphone's camera to photograph bookshelves for subsequent searching. The developed system is over 90% accurate, and has an average search time of approximately 0.95 seconds.

Going forward, this software could be implemented as part of a larger virtual bookshop/virtual library application, or as part of a system which uses mounted cameras to search every single shelf in a library or bookshop at once.

Table of Contents

Code & Resources CD attached on back cover

List of Figures

Chapter 1

Introduction

This report presents the final year project titled Book Finder, a project based in computer vision.

1.1 Project Overview

The proposed project was to develop software, using computer vision techniques, for recognising specific books in images of bookshelves, with the idea that the final product might be implemented in a mobile application. The purpose of this system would be to aid users in quickly locating books on large bookcases.

Given an image containing one or more bookshelves, and the name of a book, the system should analyse the image of the bookshelves, find out whether or not the given book is on one of the shelves, and if so its exact location in the image should be highlighted. Figure 1.1 shows a visualisation of the two inputs and the resulting output for the proposed system.

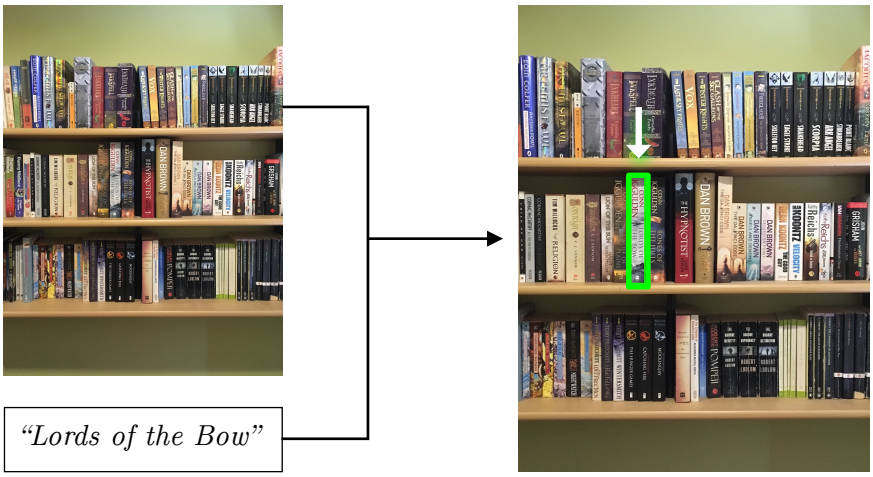


Figure 1.1: Overview of proposed system

Some of the issues that need to be taken into account when searching for books include; the effects of lighting and shadows in the scene, camera resolution, book orientation, book spine design, text, and material, layout of shelves, crooked photographs, and perspective skewing.

During the development of this software 106 book spines were scanned and a local database of scanned book spine images was created. This database allows an image of the spine of a book to be retrieved and used when carrying out a search for that book.

1.2 Motivation

The motivation for this project stems from the problem of trying to find specific books in book shops or libraries, where manually searching through large bookcases can be tedious and slow, especially if the books are not ordered. In these settings there are often hundreds of similar looking books grouped together, making the task more difficult. Library methodologies such as the Dewey Decimal System can also make it hard to know exactly where to look when trying to find a book on a large bookcase. Figure 1.2 shows such a scenario, where trying to find a specific volume of one of those books would require checking every spine one by one.

Figure 1.2: Library bookcase

The idea behind this project is to solve this problem by creating a novel, efficient, useful piece of software packaged in a smartphone application that can use the phone's camera to locate books on shelves quickly and accurately. This would allow a user to simply point their phone camera at the bookcase, and within a few seconds the location of the book would be highlighted for them.

The software could also be employed by bookshops and libraries as part of a larger virtual book-finding system, where a digital map of the library or shop could be created, so a user could be directed to the right section/aisle using geolocation, then the book-finding system could direct them from that point to the exact location of the book. Another possibility would be to have cameras pointed at every bookcase, so that every shelf in the entire library or bookshop could be searched at the same time. This would also help staff find books that have been put back in the wrong place.

For the above reasons, the proposed book-finding system would be a novel and useful piece of software.

1.3 Report Structure

The structure of this report will be as follows:

- Chapter 2 will cover the background of the project area; related work, and the theory behind the major vision techniques used.
- Chapters 3, 4, and 5 will discuss the process that was developed for taking an image of a bookcase and the name of a book, and finding the exact location of that book.
- Chapter 6 will detail the analysis of test results for the completed book-finding system, and the derivation of a classification function based on those tests.
- Chapter 7 will give an overview of the iOS application created alongside the book-finding system, and will show the interface and functionality of the application through screenshots.
- Chapter 8 will provide a collection of images taken using the iOS application to show the results and output of the book-finding system.
- Chapter 9 will present the conclusions and a discussion of potential future work in this area.
- Chapter 10 contains the appendices where more test data is included, and a number of vision techniques which were investigated but not used are outlined, and the reasons why they were not used are discussed.

Chapter 2
Background

2.1 Related Work

There has been some previous work done on recognising books on bookshelves, most notably a project from Stanford university, and a feature in the mobile application Evernote.

Chen, David M., et al.[1] outline a design for an application that uses SURF feature detection to analyse images of bookshelves against a database of book spines and identifies each book in the given image. This method starts by extracting individual book spines from an image, then works backwards to find which spine from the database each book matches most closely. Once it has segmented out a single book spine, it creates a shortlist of the 50 most similar spines from the database using SURF (Speeded Up Robust Features) feature detection, then uses RANSAC (Random Sample Consensus) to narrow down this list to find the best match. This method relies on extracting comprehensive edge data from the image, and in the test images provided in the paper every book spine is a different colour, there are no shadows, and the image is taken at close range, all of which results in strong edges between books. Experiments detailed in Appendix 2, chapter 10.2.1, show that the conditions in photographs taken in scenarios that this project aims to address, namely lower resolution photos, taken from further away from the bookshelf, that may contain shadows or irregular lighting, rarely permit such ideal edge data.

Evernote[2] has a built in functionality that automatically performs optical character recognition on any images saved in notes in the application, and catalogues any data that it can find, making the images searchable. It has been demonstrated that this feature could be used to catalogue shelves of books [3]. If a user searches for a book title it will tell the user which image in their notes contains the given title, and will highlight the matching text in that image. This relies on a clear, high resolution photo of the shelf taken at close range, and only works statically on photos that have been saved in a note on the application as opposed to working in real time with the phone's camera. It also would not work with any books that do not have clear text on the spine, for example; old hardback classic books.

2.2 Vision Techniques

A large part of the background work for this project was researching many different computer vision techniques. This section outlines the theory behind some of the key vision and image processing methods that were employed in the book-finding system.

2.2.1 Edge Detection

Edge detection is a technique for finding edges in an image. Edges, in image processing, are places where there is a high rate of change of intensity, i.e. a contour where a strong change occurs in the image. The edge detection technique used in this project is Sobel first derivative edge detection. Sobel is a compass edge detector that uses Gaussian-

weighted convolution kernels (i.e. partial derivatives) to find the first derivative of a single channel image — the first derivative of an image being the rate of change of intensity at each point in the image. The two kernels used by Sobel to find the rate of change in two directions are:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Convolving the image with the Sobel kernels generates two partial derivatives, from which the edge gradients and edge orientations for the image can be found, where edge gradients represent the rate of change in intensity at a point, and orientation describes in which direction the change is occurring.

Figure 2.1: Sample photo of bookcase

Figure 2.2: Resulting Sobel edge gradient image

Non-maxima suppression can be employed to use these two elements to create an edge image. First, since a pixel has only eight immediate neighbouring pixels surrounding it, the edge orientations are quantised into eight directions. Then, for each edge point; if the gradient of either of the two orthogonal edge points is greater than its own gradient it is suppressed (its value is set to zero). This essentially combs the edge gradient image and leaves behind only the strongest edge pixels, which will be the central pixels for each edge contour. Since the edge gradient image is greyscale it must be thresholded after non-maxima suppression to create a binary edge image; where edge pixels are white and every other pixel is black. Thresholding here sets all pixels with a value greater than zero to be equal to the maximum value of 255.

Figure 2.3: Binary edge image

2.2.2 Template Matching

Template matching is a technique for trying to find where a template image shows up in a larger target image. In the context of this project, a book spine would be a template, and an image of a bookshelf would be the target image. There are several different matching metrics that can be used when generating a matching score for the template in an overlaid position in the target image. The metric most closely examined for this project is normalised correlation coefficient. This is a variation on normalised cross correlation. Cross correlation template matching computes the sum of the products of all pixels in the template image and their corresponding pixels in target image based on the position of the overlaid template. The position in the target image that yields that maximum value for this sum is deemed to be the position of best match. The formula for cross correlation is as follows ($x,\ y$ = position of overlay in target image):

$$\mathrm{D}_{\mathrm{CrossCorrelation}}(x,\ y) \ = \ \sum_{x',y'} T(x',y') \cdot I(x+x',y+y')$$

$$T(x',y') = \text{value of the pixel at position (x', y') in the template.}$$
$$I(x+x',y+y') = \text{value of the corresponding pixel in the target image.}$$

Correlation coefficient template matching performs the same calculation, however this time the mean pixel value of the image is subtracted from the current pixel value at every position.

$$\mathrm{D}_{\mathrm{CorrelationCoefficient}}(x,\ y) \ = \ \sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))$$

$$T'(x',y') \ = \ T(x',y') \text{ - } 1/(w \cdot h) \cdot \sum T(x'',y'')$$
$$I'(x+x',y+y') \ = \ I(x+x',y+y') \text{ - } 1/(w \cdot h) \cdot \sum I(x+x'',y+y'')$$

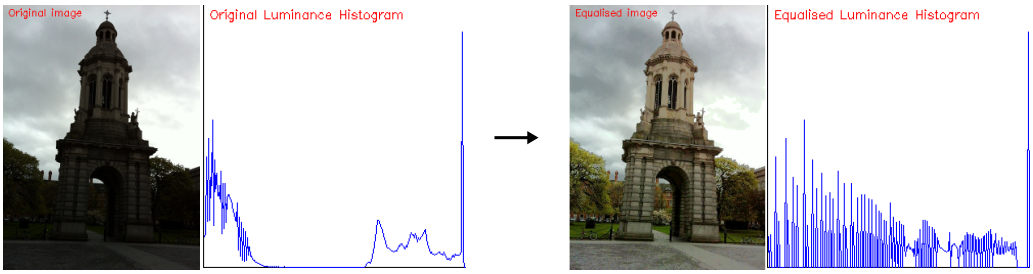
Introducing normalisation to the metric normalises the intensity variation between template and target, so it produces the same correlation even when there is a difference in overall brightness between the two images. This is very useful when dealing with photographs taken in libraries or offices where books might be in bright daylight or covered in shadow, whereas the template will be a scan of the book spine with no interference from outside light sources.

$$\mathrm{D}_{\mathrm{NormalisedCorrelationCoefficient}}(x,\ y) \ = \ \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

2.2.3 CLAHE

Contrast Limited Adaptive Histogram Equalisation (CLAHE) is a form of adaptive histogram equalisation. Histogram equalisation is a technique that takes a histogram of a single channel, for example a histogram of the luminance channel of an image in HLS (Hue, Luminance, Saturation) colour space, and redistributes the values evenly to

smooth out sharp troughs and peaks in the histogram to a certain extent. This has the effect of enhancing the contrast in the channel represented by the histogram. Figure 2.4 shows the effects of histogram equalisation on the luminance channel of an image. The luminance channel of a pixel contains information about the brightness of the pixel.



Histogram images from *A Practical Introduction to Computer Vision with OpenCV*, Kenneth Dawson-Howe, Copyright Wiley & Sons Inc. 2014. [4]
Reproduced with author's permission.

Figure 2.4: Histogram equalisation

Histogram equalisation suffers when images have regions that have very low or very high intensity compared to the rest of the image. For example the luminance channel of an image containing both bright lights and dark shadows. These outlier regions will not be sufficiently equalised when taken in the context of the whole image. Adaptive histogram equalisation addresses this problem by computing multiple sub-histograms, each representing a different region in the image, and equalises each sub-histogram independently. This increases contrast in the channel locally for each part of the image.

Finally CLAHE adds a contrast limiting dimension to adaptive histogram equalisation. Fully equalising a histogram can have quite drastic effects, but CLAHE equalises histograms only up to a specified contrast limit (referred to as the 'clip limit'). Any part of the histogram that exceeds the given clip limit has its values redistributed equally among all of the bins in the histogram. This allows for much more subtle enhancement of contrast in images. Figure 2.5 shows the subtle effects of CLAHE when equalising the luminance channel of an image in HLS colour space, compared to regular histogram equalisation.

original *regular luminance equalisation* *CLAHE*

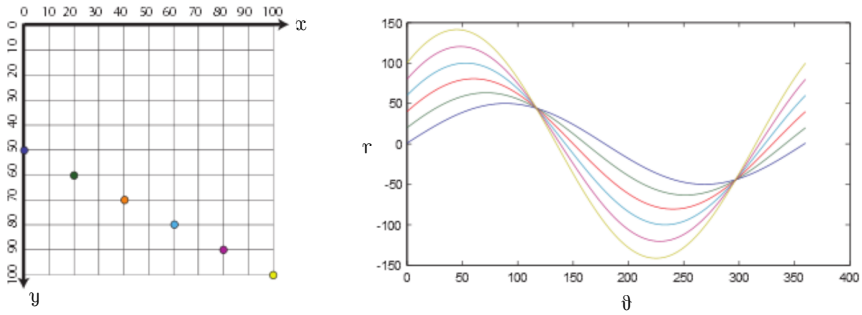
Figure 2.5: CLAHE compared to regular equalisation

2.2.4 Hough Transform

The Hough transformation is a method of extracting features from an image. It works by transforming images from image space into Hough space and then finding local maxima in the Hough space. In this project the probabilistic Hough transform for straight line segments is used.

The equation of a line can be represented in the polar coordinate system [5] as $r = x\mathrm{Cos}\vartheta + y\mathrm{Sin}\vartheta$, where r is the orthogonal distance from the line to the origin, and ϑ is the angle between the line and the x axis. After the transform, lines in the image will be represented in Hough space where the axes are r and ϑ.

To transform from image space to Hough space with a binary edge image, each edge point is considered individually. For each edge point in the image, every point in Hough space which represents a line through that edge point is incremented. In terms of the above line equation — for every pixel (x, y) in the image that is an edge point, compute a value for r for every possible value of ϑ. Since ϑ is an angle, there are 360° of possibilities. Working in 360° of Hough space will find every line twice — once in each direction. If the direction of the lines is not important then only 180° of Hough space needs to be considered. Figure 2.6 shows a mapping of six points from image space to Hough space.



Histogram images from *A Practical Introduction to Computer Vision with OpenCV*, Kenneth Dawson-Howe, Copyright Wiley & Sons Inc. 2014. [4]
Reproduced with author's permission.

Figure 2.6: Mapping from image space to Hough Space

All six curves intersect at two points in Hough space, these two points represent the straight line that would connect all six of the points in image space, going in each direction. From the lines in Hough space, and their intersections with each other, the equations of straight lines in the image can be extracted.

The probabilistic Hough transform is a more efficient variation on the Hough transform where a smaller subset of the edge points is used to extract the lines. The idea behind this is that a smaller, random subset of the set of edge points will still represent the features in the image sufficiently so straight lines can still be extracted accurately. With reference to figure 2.6, if only three of the six points in the image were considered when mapping to Hough space, the same straight line would still be found in the intersection of the lines in Hough space. Probabilistic Hough in OpenCV finds line segments and gives the two end points of a line as the output, rather than the equation of the line.

Chapter 3
Finding Shelves

Segmenting the individual shelves in an image of a bookcase is the first stage of processing done in preparation for searching for books. Knowing where the shelves are helps a great deal when trying to make sense of a photograph of a bookcase, so that steps can be taken to start looking for specific books. Being able to process and search each shelf independently makes the system more robust, and makes searching for books much faster and more feasible.

3.1 Edge Detection

The first step in finding the individual bookshelves in an image is finding the edges. Generating a binary edge image turns a colour photograph into a matrix of binary data that can be logically analysed and processed by the system. The vertical and horizontal edges in an image of a bookcase provide a lot of useful information when it comes to trying to make sense of the image and deduce where the bookshelves are.

Figure 3.1: Sample photo of bookcase

To get a reliable, accurate edge image without picking up too much interference or too many extraneous edge details, Sobel first derivative edge detection was used. Sobel operates on single channel images, so the two options for processing a colour image are either to convert the image to greyscale, or to split the image into the three individual channels — red, green, and blue channels for an image in RGB colour space — and carry out edge detection on each channel separately, then merge the three results into one aggregate edge image. In this case the latter option, multi-spectral edge detection, was chosen as it is more comprehensive and generates a more complete edge image than greyscale edge detection.

Figure 3.2: Sobel edge gradient image

Non-maxima suppression, followed by thresholding, is then used to create the binary edge image. For more detail about Sobel first derivative edge detection, non-maxima suppression, and thresholding, see chapter 2.2.1.

Figure 3.3: Binary edge image

3.2 Segmenting Shelves

Once the edge data for the given image has been generated, it is then used to find the shelves. As can be seen in Figure 3.3 the edges in an image of a bookcase are going to be primarily vertical edges; representing the sides of the books, and horizontal edges; representing the tops and bottoms of books and the shelves themselves. When moving down through such an image from top to bottom a pattern emerges — there will be a stretch where there are many rows of vertical edges beside each other, then a gap with very few edges at all, then another stretch of rows with many vertical edges, then another gap, and so on. The sections that have many vertical edges are rows of books, therefore it is relatively straightforward to deduce where the separating lines between shelves are by looking for the gaps between rows of books.

Since only the vertical edges are needed to deduce where the shelves are, all the horizontal edges are masked out of the edge image at this point. This makes the algorithm more robust and reliable, as it is only taking into account edges that are likely to be part of a book spine. Without this step other edges in the image outside of the bookcase would be more likely to interfere, or things like old wooden shelves could throw it off because lots of horizontal edges might show up in the grain of the wood.

The vertical edges are extracted using a probabilistic Hough transform for straight line segments. More information about the theory behind Hough transformations can be seen in chapter 2.2.4. First; all the straight line segments are extracted from the binary edge image using the probabilistic Hough transform. Straight lines are of the most interest because the sides of books are going to be represented by straight line edges, any other edges in the image are not useful here. Since Hough transformations are computationally expensive, and the direction of the lines does not matter in this case, only 180° of the Hough space, as opposed to the full 360°, is considered. This doubles the speed of the Hough transformation process. Then each of the Hough line segments is checked, and only vertical ones are kept. Finally the edge image is analysed and any pixels that do not lie along a vertical line segment are suppressed. This generates the binary image of vertical edges shown in figure 3.4

Figure 3.4: Vertical edges

A three-pass system was used to accurately find the rows in the image that separate the shelves.

- The first pass counts the number of edge pixels in each row and calculates an average number of edge pixels per row for the image.
- The second pass looks for rows that have significantly fewer edge pixels than the average row and marks them as possible shelf-separating rows.
- The third pass parses this list of possible shelves and groups them based on proximity to each other. If there is a large gap between a bookshelf and the tops of the books on the shelf below it then there could be many consecutive rows designated as possible shelves, but seeing as they are so close to each

other it is concluded that they all correspond to the same shelf divide. As a result only the top-most of these rows is kept, the rest are discarded. The reason for the top-most row being chosen, as opposed to say the middle row, will become clear in section 3.3 when the shelves are cropped. If there is a large enough gap between two possible shelf rows then they are concluded to correspond to different shelves. At the end of this pass the list contains only one separating row per shelf.

Some extra steps are taken alongside this algorithm to handle the very top and bottom of the bookcase. For the top of the bookcase; the image is taken row by row from the top down until it starts to hit the top row of books, once the top row of books is identified, it backs up to just above the top of that row of books — this is the starting point from which the algorithm can go on to find all the middle shelves. There are also checks in place to handle cases when there is no gap below the bottom shelf, in these cases an extra shelf line is added at the very bottom of the image to ensure the bottom row of books is not overlooked.

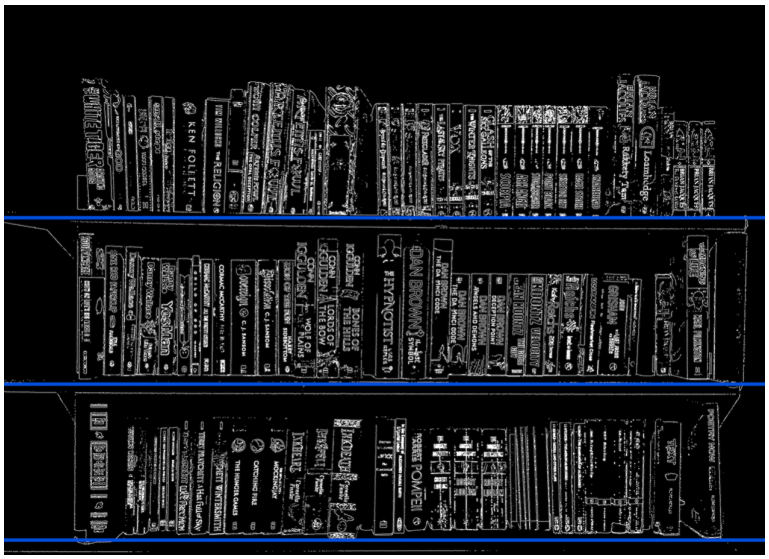


Figure 3.5: Lines separating bookshelves

Figure 3.5 shows an example of an edge image with the corresponding shelf-separating rows that were deduced represented by blue lines. Once all the separating lines between shelves have been identified, the image can be split into a number of sub-images, each containing one shelf. This allows each shelf to be processed in isolation from this point onwards.

3.3 Cropping Shelves

Before moving on to the search phase, each individual shelf image is cropped. This speeds up searching and allows for better handling of abnormal cases (such as images that do not contain any bookshelves). If the number of pixels in the target region can be reduced then search time can also be reduced proportionately. Since the template matching that will be used to search for the book in Chapter 4.2 is quite computationally expensive, it is worth reducing the target area, and thus the number of unnecessary template matching calculations, as much as possible.

The edge image is used to determine if there are barren regions of the shelf image, and if so these regions are cropped out — barren regions being regions of the image where there is no edge data. There is no need to search these empty spaces when looking for a book. Figure 3.6 shows a shelf before it has been cropped, barren regions can be seen above the books, and to the left and right. Due to how the separating lines between shelves are found in section 3.2, there will be no blank space underneath the books at the bottom of each shelf, so the bottom of the image does not need to be considered during cropping.

Figure 3.6: Bookshelf before cropping

The top of each shelf is analysed row by row from the top downwards. Every barren row is cropped out of the image until the first edges start to appear, then the cropping stops. This comprehensive search for edges is necessary for the region above the top of the books since there are likely to be many books, all of different heights, so the top-most edge of the tallest book must be found in order to determine the top of the target region for the search.

The left and right hand sides of the target region are each going to consist of one straight line representing the upright edge of a book, so a more efficient method can be used to find the x position of this upright edge on each side. Each side of the image is cropped by sending three evenly spaced prongs in from the very edge towards the centre of the image until one of the prongs hits the side of the first book. Three prongs are used, instead of just one, to allow for books that are leaning slightly, as opposed to being perfectly vertical in orientation. In these cases the top or bottom of the book will be closer to the side then the rest of it.

This is faster than checking every pixel, but also is less likely to be thrown off by outlying edge pixels. For example a single edge pixel somewhere out beyond the sides of the shelves would stop the cropping process prematurely if it is hit, even though it is not actually part of a book. However by only checking three different rows for edge

pixels as it works its way in towards the shelves, it is likely that such outlying edge pixels will slip through the cracks and go unnoticed. When the side of a book comes along, there will be a long vertical edge that cannot be missed by the search prongs, so the cropping process will be correctly stopped at this point.

Figure 3.7 is a visualisation of this cropping process, where the red denotes pixels that were examined when looking for edges.

Figure 3.7: Bookshelf during cropping

Figure 3.8: Bookshelf after cropping

The cropping is more effective in situations where only a section of the given image actually contains bookshelves, as opposed to situations where the shelves fill the whole image. In such cases some irrelevant parts of the image around the actual shelves will be cropped out. It also helps in dealing with abnormal cases, such as images with no bookshelves or very little edge data. In such cases a lot of the image is likely to cropped away, stopping unnecessary template matching calculations and allowing a relevant alert to be returned sooner.

Chapter 4
Searching for Books

The original image has now been reduced down to several target areas for searching. Each individual shelf has been identified, stored as a separate image, and cropped. A small amount of pre-processing needs to be done on each image in preparation for the template matching, and then the actual search for the book can commence.

4.1 Pre-Processing

Template matching operates in greyscale so at this point each shelf image is converted from RGB colour space to greyscale. The greyscale shelf images are then equalised using CLAHE — Contrast Limited Adaptive Histogram Equalisation. CLAHE is explained in detail in chapter 2.2.3. The equalisation better distributes the intensities in the image, which can help lessen the severity of very dark or bright zones in the image. This means that shadows and shines, both of which are often prevalent in photographs of bookshelves, can be mitigated to a certain degree. Figure 4.1 shows a bookshelf image before equalisation. The red squares highlight areas where shadows thrown by the shelf above can be seen at the tops of book spines. Without equalisation the shadows prevent those books from being found correctly by the template matching.

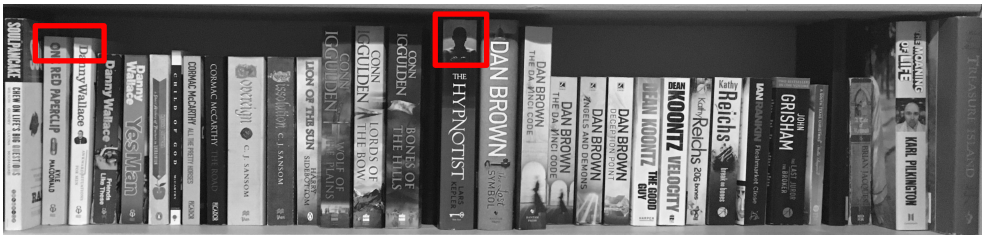


Figure 4.1: Bookshelf before equalisation

Equalising the image, with a low clip limit to keep the effect subtle, results in the image seen in figure 4.2. Here the contrast of the shadows has been diminished, as can be seen in the areas again highlighted by the red squares.

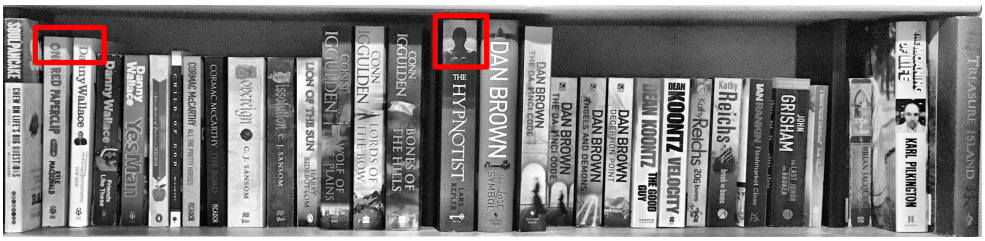


Figure 4.2: Bookshelf after equalisation

As a result the books in question can now be correctly found by the search methods in the next phase.

Finally, before the template matching commences, the cropped and equalised bookshelf image is scaled down in size. The iPhone SE, the smartphone model used in developing this system, produces photos 4032×3024 pixels in size, which means shelves in the image could be anywhere up to approximately 4000 pixels high if the photograph was taken at close range. It is unnecessary to have such large images, and slows down processing. Dealing with large images makes the template matching phase significantly slower. It was found that each shelf image could be scaled down to a height of 300 pixels without any detrimental effects to template matching accuracy. Scaling the shelves down any more than that meant a possible loss of template matching accuracy when dealing with small books, 300 pixels was found to be a safe height to scale down to without affecting results. This step speeds up the search process significantly.

4.2 Template Matching

Once the shelves are prepared, the actual search for books can begin. This is done through normalised correlation coefficient template matching. Template matching is a technique whereby a match for a template image, in this case the book spine, is looked for in a larger target image, in this case the bookshelf image. The template image of the book spine is loaded from a given database of book spine images, and converted to greyscale before being used for template matching. A confidence value is produced as a measure of how close the match is, based on what matching metric is used. Figure 4.3 shows some sample matches in a target image, with the resulting confidence levels.

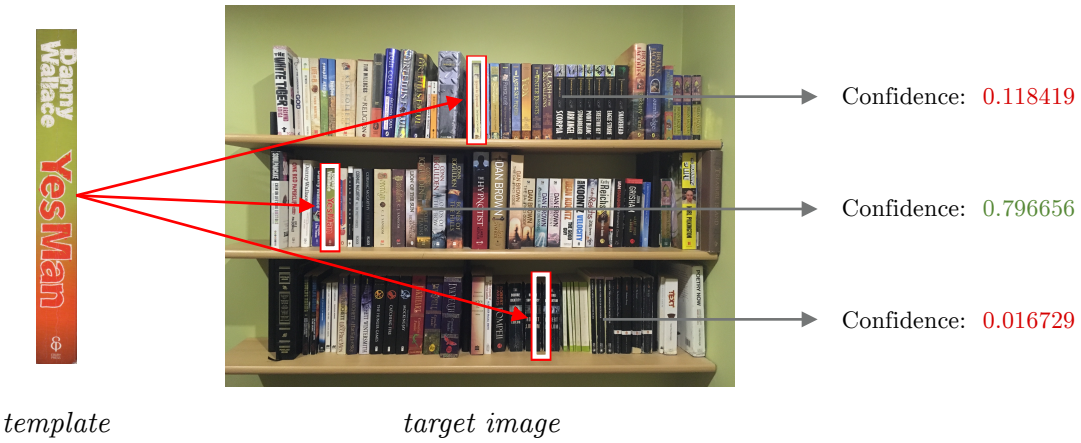


Figure 4.3: Template matching visualisation

Three dimensional template matching is employed to search for books, the three dimensions being; x coordinate, y coordinate, and template scale. In other words when looking for the book spine template in the target image, it is overlaid at every x position and every y position at every scale. The scale variable is necessary as the scale of books in the given photo of the bookshelves will vary every time depending on what distance the photograph is taken from, and the resolution of the camera. To get an accurate template match the size of the given spine in the target image must be the same as the template image of the spine. It would be both unnecessary and computationally infeasible to search for the template at every possible scale, so the scale variable is computed intelligently based on information derived from shelf image.

Figure 4.4 shows a visualisation of templates overlaid on a target image at various positions and various scales, where a red or green rectangle represents the outline of an overlaid book spine template. The green rectangle is at an appropriate scale and position to accurately overlay onto a book on the shelf, so this would yield a high template matching score if that was the book being searched for.

Figure 4.4: Visualisation of template matching variables

For the first round the template is scaled to be the same height as the shelf image. The cropping phase outlined in chapter 3.3 results in each shelf image being only as high as its tallest book, so the template will start out at the same scale as the largest book in the shelf image. Then, after match confidences for every position in the target image have been computed, the template is rescaled to be 95% of its previous size, and the shelf is searched again. 95% was found to be the optimum ratio by which to scale down the template image, considering the trade off between accuracy and efficiency. This process is repeated until the scale of the template image is less than half the scale used in the first round. It is assumed to be unlikely that a single shelf would contain such a range of book sizes that some books would be less than half the height of others, and in practice this has proved to be a fair assumption. However it is trivial to adjust this parameter if needs be. Scaling by 0.95, and stopping once the template reaches half its original size, means the template will be searched for at 13 different scales ($0.95^{13} = 0.5133$, $0.95^{14} = 0.4876$).

The confidence values for template matching are calculated using the normalised correlation coefficient metric. For more details on this matching metric see chapter 2.2.2.

Figure 4.5 shows an example of a correct result returned by normalised correlation coefficient template matching. In this case the confidence is 0.816721.

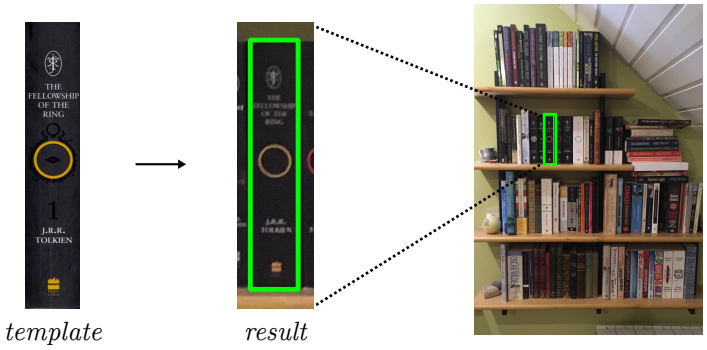


Figure 4.5: Example of correct template match

Using this template matching procedure, all the individual shelves obtained in the image are searched in parallel and the match that has the highest confidence out of all the shelves is taken to be a tentative result for the location of the book.

Template matching confidence values are not always reliable, there are many cases where incorrect locations in the image can yield high confidence values. Figure 4.6 shows an example of this. The confidence value here is 0.752992.

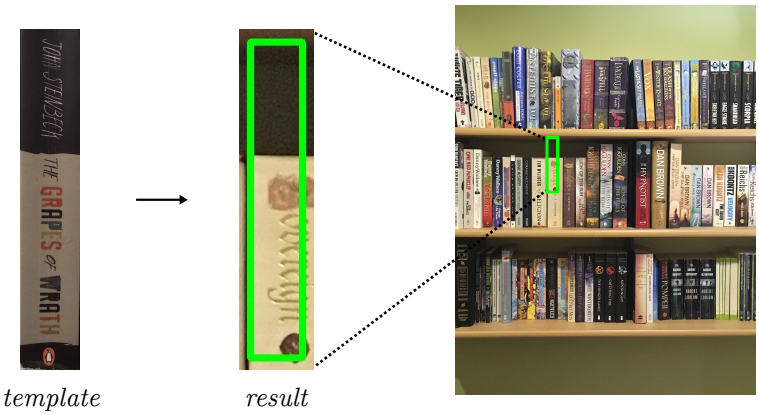


Figure 4.6: Example of incorrect template match

The region of the image returned here is not even centred on a book, but it resembles the given book spine in that the top third is dark, then there is a lighter section in the middle which contains text, and then another dark patch at the bottom of the spine. Therefore a high confidence value is produced.

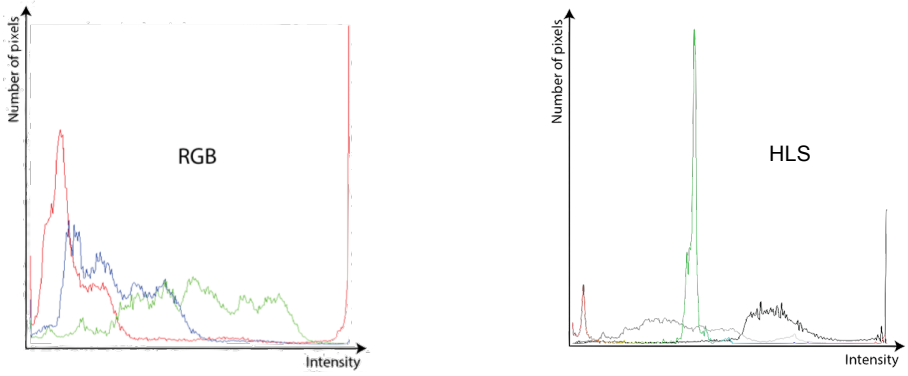
Since the template matching confidence values are not always reliable, further analysis is necessary to ensure that the result is indeed correct. The procedure for this further analysis and validating of the tentative result returned by the template matching is discussed in chapter 5.

Chapter 5
Validating Match

At this point a tentative match for the book in question has been found by the template matching, but further steps need to be taken to ensure that it is a correct result. Since template matching works in greyscale, the colour data associated with the book spine has not been used yet, so a colour histogram comparison can be done to verify that the colour information is consistent between the book spine loaded from the database and the potential book spine on the shelf returned by the template matching. It could be considered that *a*) the colours and *b*) the actual patterns and features on a book spine are the two most prominent pieces of information when it comes to distinguishing one book from another. Therefore, if those two sets of information can be tested independently of each other when comparing book spines, a comprehensive two step verification method can be implemented. Greyscale template matching followed by colour histogram comparison achieves this.

5.1 Colour Histogram Comparison

Colour histogram comparison is a computer vision technique whereby histograms are computed to represent the colours in two images, and then a comparison metric is employed to determine how similar the two distributions of colours are. A colour histogram describes the number of pixels at each colour value in the image. How this is represented depends on the colour space.



Histogram images from *A Practical Introduction to Computer Vision with OpenCV,*
Kenneth Dawson-Howe, Copyright Wiley & Sons Inc. 2014. [4]
Reproduced with author's permission.

Figure 5.1: Colour histogram examples

The two colour histograms in figure 5.1 represent images in RGB and HLS colour spaces. The RGB histogram shows the number of pixels at each intensity of red, green, and blue values, the HLS histogram shows the number of pixels at each intensity of each hue, luminance value, and saturation value. When comparing the colours in two book spines both of these colour spaces are utilised. HLS is used when equalising luminance (see section 5.2), and RGB is used for the actual colour histogram comparison calculation.

When computing the histogram of an image, the colours in the image must first be quantised into a number of bins. The histogram then represents the number of pixels that fall into each colour bin. It was found that using 8 bins yielded the best results for comparing book spines here. Figure 5.2 shows the average, lowest, and highest histogram comparison scores found when testing different numbers of bins for both correct and incorrect book matches. It is desirable that the average and low values for correct matches would be as high as possible, without making the average and high values for incorrect matches too high.

Histogram Bins:	2 Bins	8 Bins	16 Bins
Correct Match — Average	0.93	0.59	0.59
Correct Match — Low	0.43	0.12	0.09
Incorrect Match — Average	0.69	0.25	0.27
Incorrect Match — High	0.99	0.70	0.55

Figure 5.2: Histogram bins statistics

As the table shows, 8 bins yields the best trade off in terms of high results for correct matches and low results for incorrect matches. Using 8 bins maximises the difference between the average values for correct matches and incorrect matches.

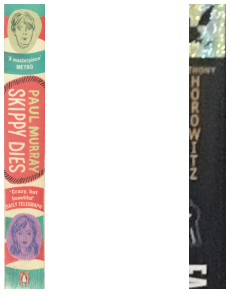
The correlation colour histogram comparison metric is used to generate a colour comparison score. This gives a score between -1 and 1, where 1 is a perfect match.

$$\mathrm{D}_{\mathrm{Correlation}}(h_1,\ h_2) \quad = \quad \frac{\sum_i (h_1(i) - h_1)(h_2(i) - h_2)}{\sqrt{\sum_i (h_1(i) - \bar{h_1})^2 \sum_i (h_2(i) - \bar{h_2})^2}}$$

Figure 5.3 below shows the template of a book spine and the result returned from searching a photograph for that book. This is a correct match, and the resulting colour histogram comparison score is 0.958765. Figure 5.4 shows a book spine template and a region of a bookshelf image that does not match, in this case the search did not correctly find the book, and the resulting colour comparison score is 0.001970.

template *search result*

Figure 5.3: Colour comparison for correct match



template *search result*

Figure 5.4: Colour comparison for incorrect match

These are two extreme cases, where the correct result returns a very high colour comparison score and the incorrect result returns a very low score. The average score for a correct match at this point is 0.59, and the average score for an incorrect match is 0.25. There may be outlier cases if the colour in the region returned by an incorrect template match, by chance, happens to resemble the colour in the book spine and yields a high colour comparison score. There is not much that can be done to rectify the colour comparison score in such cases. Nevertheless, colour comparison is a useful metric when trying to classify a search result, especially when combined with the template matching confidence. More tests for average template matching and colour comparison scores, as well as details of how a search result is classified as either correct or incorrect using the two scores, can be seen in chapter 6.

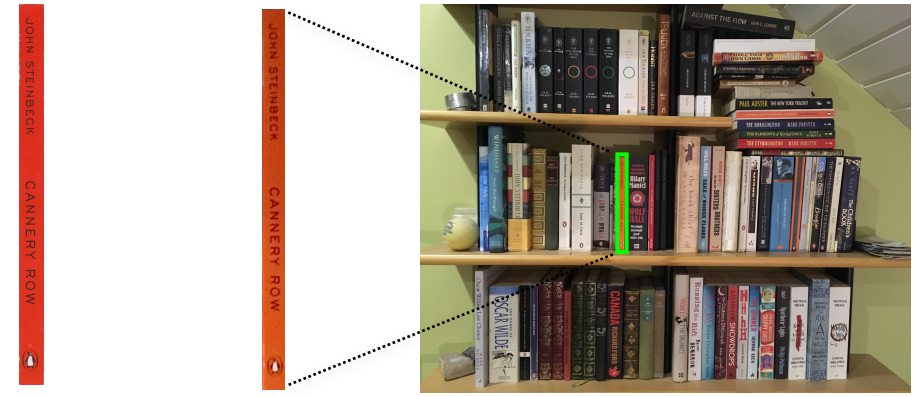
5.2 CLAHE

Before the colour histogram comparison is carried out, some work is done to prepare the book spine image obtained from the given photograph. The colour in photographs, especially from cameras on smartphones, as opposed to professional SLR cameras, often suffer from the effects of lighting in a scene. For example; colours can appear washed out when photographed in daylight, or shadows can make shades of red look like dark grey or black. The image of the book spine loaded from the database, on the other hand, is a scan, so the colour has not been interfered with from any outside light sources. Figure 5.5 shows comparison of a scanned booked spine to an image of the same book spine taken from a photograph. The colour is much more intense and vibrant in the scan, whereas the photograph is darker and less vibrant. Results such as this would yield a low score for colour histogram comparison.

Figure 5.5: Comparison of colour vibrancy

In order to combat this, Contrast Limited Adaptive Histogram Equalisation (CLAHE) is used to mitigate the damage to the colour information, so that the colour histogram comparison can return a higher value when comparing a book spine match that is correct but has distorted colour information. See chapter 2.2.3 for details about how CLAHE works.

In preparation for colour histogram comparison, the image of the book spine found in the photograph of the shelves is converted to HLS colour space, which separates the brightness values in the image from the colour information, and the brightness, or luminance, channel is equalised using CLAHE. The luminance channel represents the brightness component of each pixel in the image, so this helps brighten colours in the given photographs without changing the actual hue of the colour. This can significantly increase colour histogram comparison scores for correct book matches, especially when dealing with spines that contain vivid colours. Figure 5.6 shows the appearance of a book spine in a photograph of a bookcase compared to the scan of that spine.



scan *photograph*

Figure 5.6: Scan of book spine compared to photo of book on shelves

In this case, even though the colours look similar to the human eye, the distorted colour in the photograph leads to a low colour histogram comparison score of 0.027839. Figure 5.7 shows the same two images of the book spine after they have been converted to HLS and had their luminance channels equalised using CLAHE.

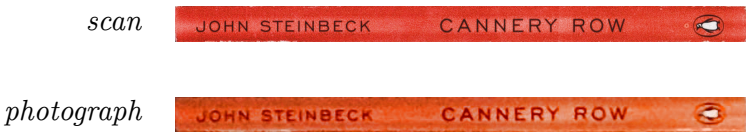


Figure 5.7: Scan and photograph after CLAHE

Now the colour histogram comparison results in a score of 0.313009, over ten times higher than the previous score. It is still a relatively low score but is high enough that it would permit this to be deemed a correct match when combined with the template matching confidence. With the previous score of 0.027839 this would never be classified as a correct match. The main benefit of using CLAHE is to handle extreme cases like this where a correct match yields a very low colour comparison score, so while it does also increase the average score, correcting the lowest recorded scores for correct matches is the main focus when implementing CLAHE and choosing clip limits.

The CLAHE done on the on the photograph of the above book spine has a clip limit of 2, the scan has a clip limit of 1. This was the configuration of clip limits chosen for the validation procedure. For the relevant test data and details on how this clip limit configuration was chosen, see Appendix 1, chapter 10.1.1.

The process of analysing the image of the bookshelf, searching for the book, and validating the match is now complete. At this stage a region of the image has been identified as a potential location for the book, and two scores — template matching confidence, and a colour histogram comparison value — have been generated as confidence values to help decide whether the search result is correct or not. Details of search result classification, and analysis of the performance and accuracy of the system, are contained in chapter 6. Examples of the results of searches for various books can be seen in chapter 8. For details of other computer vision techniques that might have been used in this process but were not, such as character recognition, see Appendix 2, chapter 10.2.

Chapter 6
Combining Metrics

Having developed a system to load book spine images and find specific books in photographs of bookshelves, testing was then carried out in order to evaluate the accuracy and performance of the system. The book-finding system was packaged in an iOS application, a sample database of book spines was created, and test searches were carried out. Details of the iOS application are outlined in chapter 7. The following tests were done on an iPhone SE, which has a 12-megapixel camera with $f/2.2$ aperture and a five-element lens.

6.1 Tests

A sample database containing 106 scanned book spines was created, and each book was searched for using the iOS application. The 106 books were located across three different bookcases. Doing the tests through the application meant a new photograph of the relevant bookcase was taken for every search. Since the ground truth for the tests was provided manually, on a per-search basis, it was infeasible to run thousands of tests looking for every spine on every shelf in every condition at every orientation. Therefore the test data presented in this chapter is instead a cross section comprised of a single search for each book, and over the 106 searches a variety of conditions and phone orientations were covered. Figure 6.1 shows photographs of the two bookcases that contained most of the books for these tests. The photographs are taken from approximately the same distance as they were during testing.

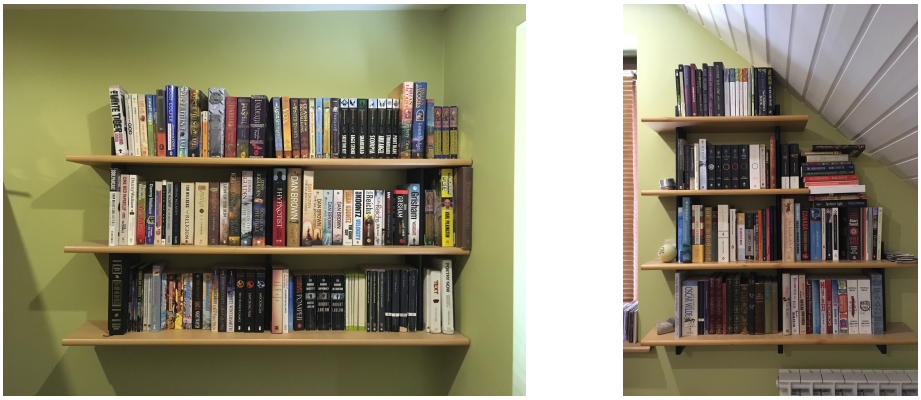


Figure 6.1: Bookcases used for testing

Figure 6.2 shows a graph of the template matching and colour comparison scores for each of the 106 searches. Green points plot the scores for successful searches where the book was found correctly, red points plot the scores for unsuccessful searches where the book was not found correctly.

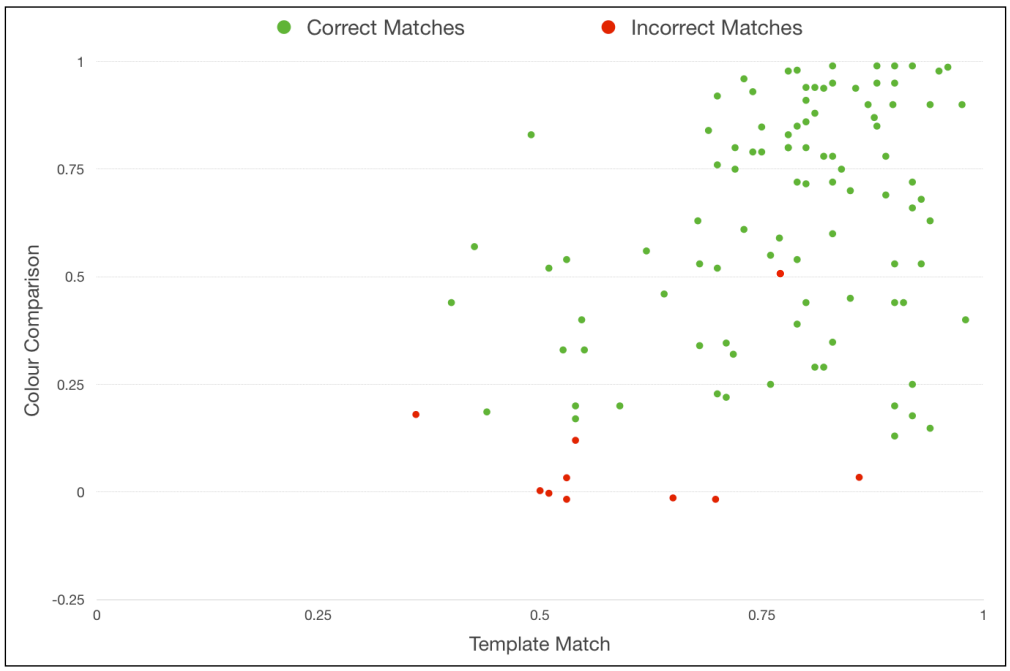


Figure 6.2: Graph of test results

Out of the 106 test searches, the book was found correctly 96 times, and not found correctly 10 times, resulting in an accuracy level of 90.5%. The failed searches here are cases where the given book was not found in that particular search, as opposed to cases where the book *could not* be found. It is often possible to redeem a failed search attempt by simply taking a new photograph with the application and trying again. The most dense cluster of correct matches is in the top right of the graph, where both template matching and colour comparison scores are high.

Most of the incorrect matches in the test data have low colour comparison scores, and a wider range of template matching confidences. There is one outlier where both scores for the incorrect match were higher. This was a case where the book was not found correctly, but the area that was returned by the template matching had a similar distribution of colours to the given book spine. A further examination of this outlier can be seen in the results in chapter 8.2.

For correct matches the average template matching confidence was 0.779 and the average colour comparison score was 0.643. For incorrect matches the average template matching confidence was 0.595 and the average colour comparison score was 0.0826.

The average search time was 0.95315 seconds. This is quite fast considering vision techniques such as template matching and Hough transformations are computationally expensive. Parallelising the shelf searches, Cropping the shelves, resizing each shelf image, and intelligently computing the scale variable for the template matching all contributed to speeding up the process. Searches on bookcases with fewer shelves than the examples shown in figure 6.1 would be faster than this again. See Appendix 1, chapter 10.1.2, for a listing of the recorded search times used to calculate the average.

6.2 Classification

Having collected the test data, a classification function could be derived from the graph. The classification function allows the system to discern whether a search result is correct or not based on the template matching and colour comparison scores.

For this test data the two sets of results, correct and incorrect, are mostly linearly separable, with the exception of the one outlying incorrect match. The line that was interpolated to separate the two classes of results has the equation:

$$y = -0.2262x + 0.27$$

This line is shown in yellow in figure 6.3. The function eradicates 90% of the incorrect matches, without losing any correct matches. Out of the 106 results, one result would have been misclassified by the function, so it follows that, based on this test data, the function can correctly classify search results 99.0566% of the time.

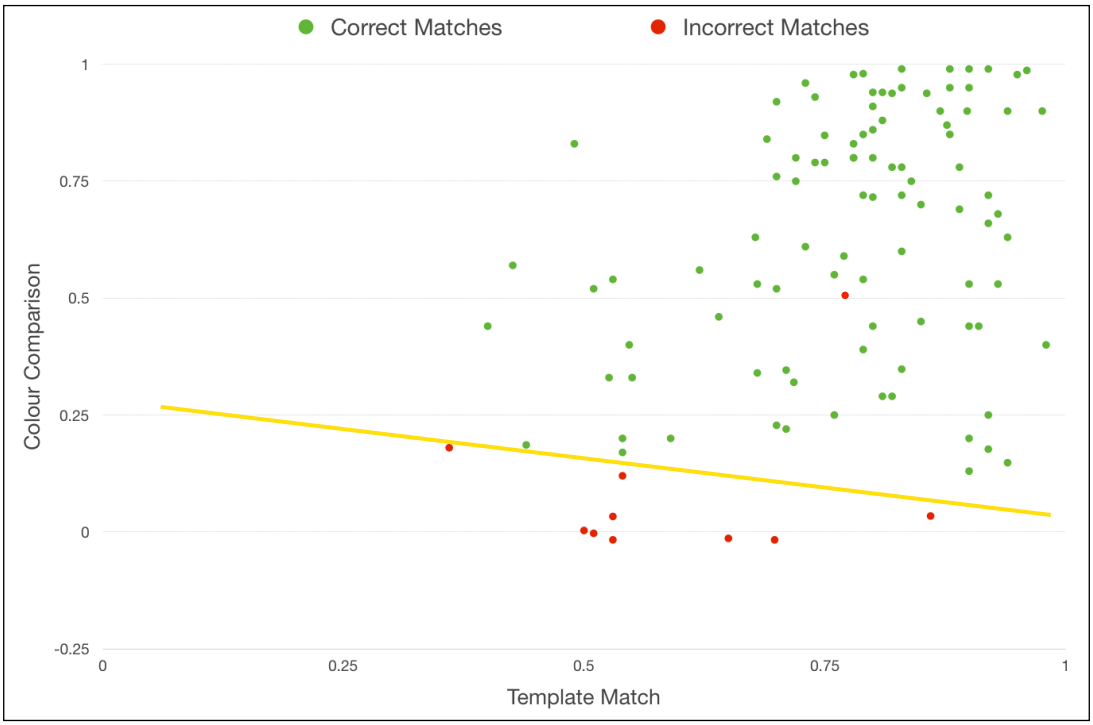


Figure 6.3: Graph with classification function

Chapter 7
iOS Application

An iOS application was developed as part of the project, in which the book-finding system was packaged, as a way of deploying this tool in a useable format. The application is named *shelfie* and carries out the core task of finding books using the phone's camera, as well as having some extra functionality to make it more useful.

7.1 Core Features

The application was developed in Swift 3, with the computer vision processes embedded inside an Objective C++ wrapper. The OpenCV computer vision library [6] was used to implement many of the vision techniques. The application takes advantage of the multi-core architecture found in modern iPhone models by parallelising the search for books, so all shelves are searched at the same time. The parallelisation is implemented through Swift's Grand Central Dispatch [7] multithreading protocol.

On opening the application the user is presented with a screen showing a blurred view of the output from the camera, with a text entry field overlaid. Once the user enters the name of a book, they are taken to the main screen where they are able to take a photograph of the bookcase they wish to search. At this point the blur is removed, and grid lines are added to aid the user in taking straight photographs of bookcases. The camera component was implemented using Apple's AV Foundation software [8]. Image capturing functionality is embedded within this screen of the application. The application permits high resolution photographs to be taken, with automatic image stabilisation. Photographs taken by the user are passed through to the book-finding module, where images are converted to OpenCV matrix format, the book-finding process is carried out, and if the search is successful the image is returned and displayed on the screen to the user with the location of the book highlighted. The book is highlighted by outlining it in green, and an animated bouncing arrow hovers above or below (depending on the position of the book in the image) pointing to the book. Resulting images with the book highlighted can be saved to the user's local photo library on their iPhone.

Figure 7.1 shows some screen shots of the core functionalities of the application. These screenshots are all in portrait orientation but the application also functions in landscape orientation.

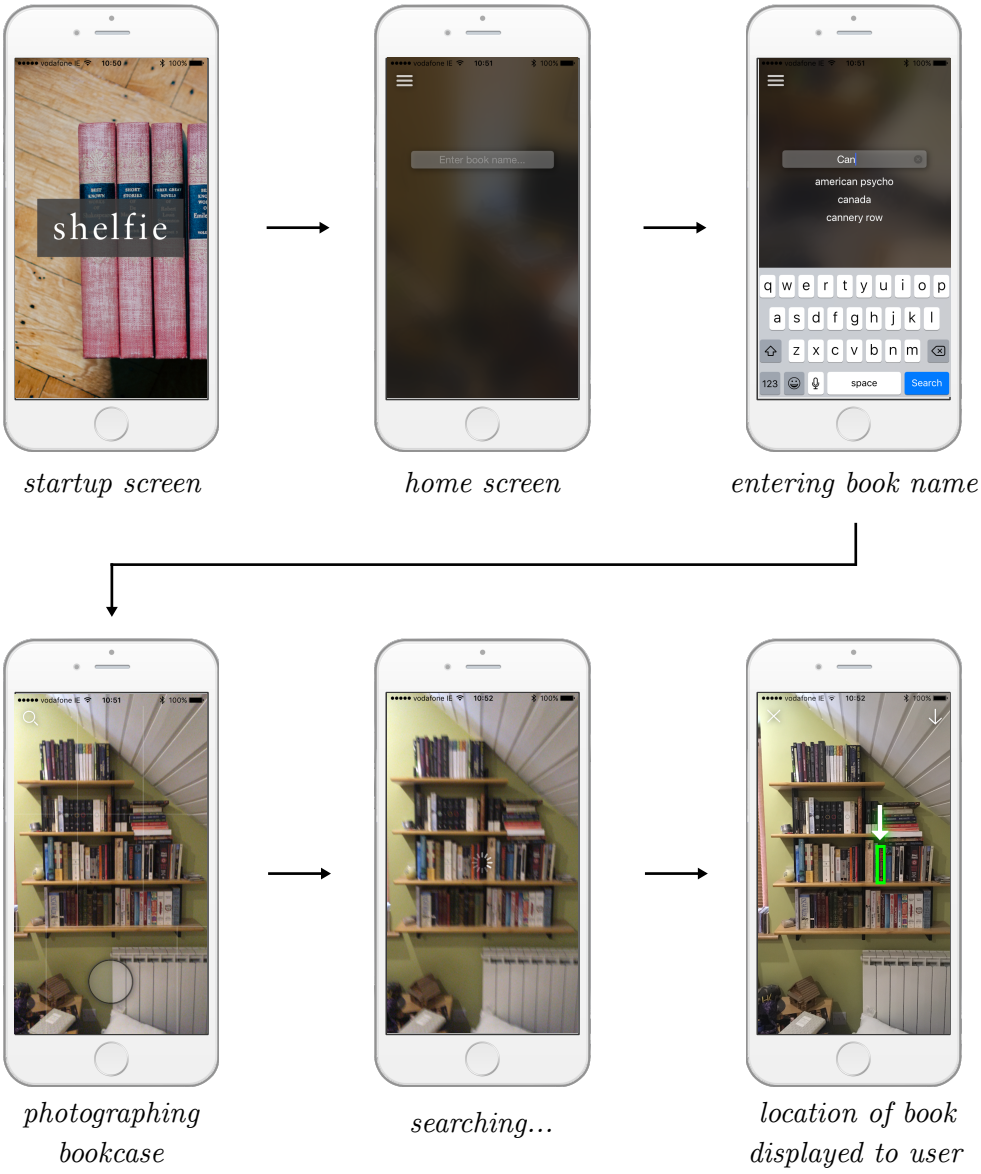


Figure 7.1: iOS application core features

A number of catches were put in place to prevent the application from encountering errors when dealing with troublesome images, such as photographs that do not contain any books etc., this makes the application robust. If no bookshelves can be found in the image an exception will be thrown and the user will be alerted and will have an opportunity to take another photograph. The application also uses the classification function derived in chapter 6.2 to determine whether a result is correct or not. If an error occurs due to a bad photograph (such as a photograph that does not contain any books, or has very little edge data), or the result is not correct, the message in figure 7.2 is displayed to the user.

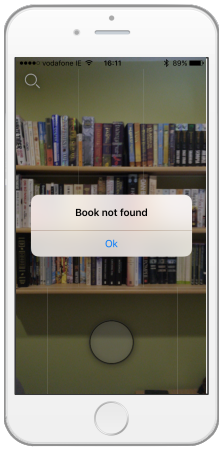


Figure 7.2: “Book not Found” screen shot

7.2 Extra Features

Some extra features were developed in order to make the application more useful and usable, including; a suggested books list appearing when typing in the name of a book to search for, the ability to add a new book spine to the database, a list of all book spines currently in the database, and the option to download a result image to the user’s photo library. Some of these features are shown in figure 7.3.

suggested book names

adding a new book spine to database

list of books in database

Figure 7.3: iOS application extra features

When a user starts typing the name of a book, the list of suggested books appears. This list is recalculated for every character the user types, so it narrows down as the user continues to enter more characters into the text input field. If any of the book names in the list of suggested books are clicked then that book spine is loaded and the user is instantly taken to the main camera screen where they can photograph the bookcase and subsequently find the selected book. The current version of the application stores all the book spine images locally, but in the future it would be intended that a remote database of book spine images would exist, which could be added to and queried by users.

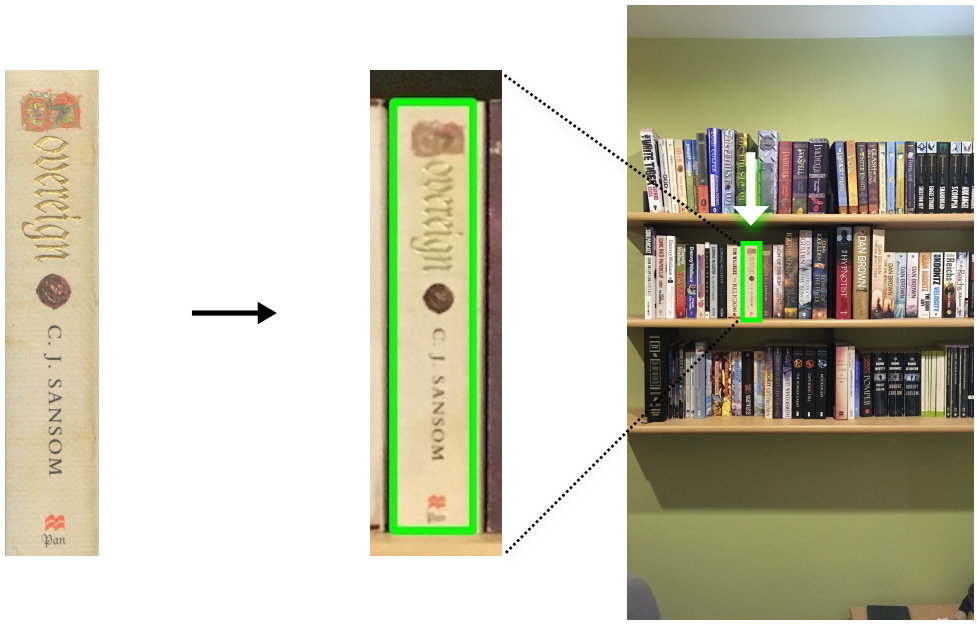
Chapter 8

Results

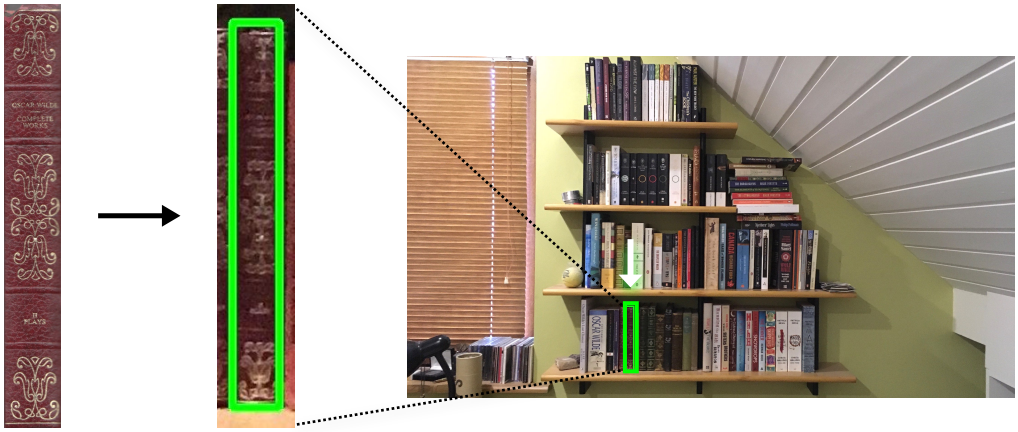
The following result images show the output of the final product — the book-finding software deployed in an iPhone application. These results were produced through the iOS application running on an iPhone SE. Examples of successful searches carried out in a home setting, a library setting, and a bookshop setting are shown, as well as examples of failed searches.

8.1 Successful Searches

Figures 8.1 and 8.2 show the results of several successful searches from a home setting.



Template Matching Confidence: 0.859588
Colour Comparison Score: 0.749521



Template Matching Confidence: 0.637479
Colour Comparison Score: 0.379201

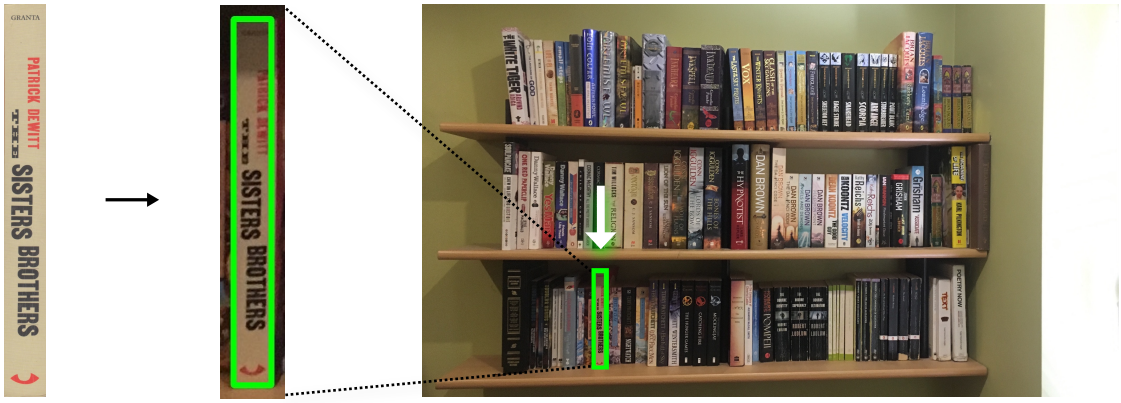
Figure 8.1: Home bookcase results I

Template Matching Confidence: 0.919895
Colour Comparison Score: 0.751823

Template Matching Confidence: 0.706669
Colour Comparison Score: 0.717445

Figure 8.2: Home bookcase results II

Figure 8.3 shows an example of how the system can cope with relatively severe shadows across a book spine and still find the book correctly.



Template Matching Confidence: 0.760999
Colour Comparison Score: 0.148110

Figure 8.3: Coping with shadows

The colour comparison score is fairly low, due to the low lighting and strong shadow in the photograph, but it is high enough that it is still classified as correct.

Figure 8.4 shows two examples of the application in use in a library setting where the book was correctly found.

Photos taken in Hamilton Library, Trinity College Dublin

Figure 8.4: Library bookcase results

In this case, where the application was used in a university library, there are white labels on each of the books, presumably placed by the library for administrative purposes. In this case the images of the book spine in the database were taken with the labels on, but if books were searched for using a global database of stock book spines where the spine images did not include these specific labels then it may cause problems, and would possibly prevent the books from being found correctly.

Figure 8.5 shows several examples of the application in use in a bookshop setting, again where the book was correctly found. These results are from tests carried out in the bookshop Hodges Figgis in Dublin.

Photos taken in Hodges Figgis Bookshop. Used with permission.

Figure 8.5: Bookshop bookcase results

8.2 Failed Searches

The tests discussed in chapter 6 show that there can be outliers when it comes to template matching confidences and colour comparison scores, and there will be occasions where a result is misclassified. Figure 8.6 shows such an occasion where the book was not correctly found, but the search result was mistakenly deemed to be correct.

Figure 8.6: Misclassified search result

Looking at this closer in figure 8.7, the intensity variation in the region of the image returned by the search resembles that of the the given book spine template, and the colours in the region also happen to be very similar to the colours in the book spine.

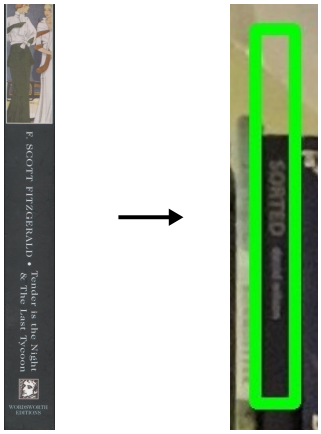


Figure 8.7: Inspection of misclassified search result

It is hard to legislate for cases like this as both the template matching confidence and colour comparison score are reasonably high. The colour comparison score especially is well above the average for an incorrect result. Since template matching does not take colour into account, it is just unlucky that in this case it happened to land on a region where the colours are also similar to the given spine template.

Another situation where the book is not found is when there is a very plain book spine with few distinguishing details. Figure 8.8 shows an example of such a spine. The black border line has been added to the figure to make the perimeter of the spine clearer.

Figure 8.8: Book spine with little distinguishing detail

Spines like this, where the colour is plain all over, and the text is small and has low contrast with the background, give the template matching algorithm very little to work with.

Damaged books can also cause problems. If the scanned image of a book spine in the database is of a brand new, pristine copy of the book, but the copy of the book on the bookcase in question is old and very worn or damaged, then that can prevent the book from being found correctly. Figure 8.9 shows an example of this.

Figure 8.9: Damaged book spine on shelf

In this case the book is not found. This is not so much a flaw with the system, as merely just unfortunate circumstances in which it should not be expected to work.

Chapter 9
Conclusion

Software which uses computer vision techniques to find specific books in images of bookcases has been developed, along with an iOS application in which it has been deployed, tests have been carried out and the findings evaluated, and results from use of the application have been presented. This chapter presents a conclusion to the work done and the outcome, as well as a discussion of topics for further work in the future.

9.1 Conclusion

In conclusion, a book-finding system has been developed which is 90.5% accurate. It has been packaged in an iOS application, and takes advantage of the iPhone's camera and multi-core architecture. It can search images containing hundreds of books and find the location of a specific book in an average time of approximately 0.95 seconds. It can cope with the effects of shadows and harsh lighting in images to a certain extent, and can successfully distinguish very similar looking book spines from one another.

The book-finding software itself was the main focus of the project, but a significant amount of effort also went into developing an elegant, efficient, user friendly iOS application in which to package it. Effort was also put into researching many other computer vision techniques which were not implemented in the final system, details of some of these unused approaches can be found in Appendix 2, chapter 10.2.

9.1.1 Strengths and Limitations

The strengths of the system include its accuracy and speed. If the specified book is present in the image, it finds it correctly over 90% of the time. An average search time of approximately 0.95 seconds is quite fast, considering computer vision techniques such as template matching and Hough transformations are computationally expensive. Other strengths include the ability to locate classic books, and distinguish between similar looking spines, such as multiple volumes of the same book. Other approaches to finding books, such as reading the text from the book spine, would fail when it comes to locating some classic books whose spines contain no text.

The limitations of the system include; dealing with books that are slanted on shelves, or not in an upright orientation, crooked photographs of bookshelves, and searching for books with damaged spines, or very little detail on the spine. Dealing with crooked or skewed photographs, and coping with non-vertical book orientation on shelves are potential areas where the system could be developed further in the future, see section 9.2 for more on this.

9.1.2 Potential and Future Prospects

In terms of prospects and potential uses for this software, the iOS application is usable and useful right now. It gives users the ability to add new book spine images to a local database within the application, and then search for those books. This would be useful in a home office or any home that has a large collection of books. When it comes to applying the system in bookshops or libraries, there is further potential for the software

to be used within a larger virtual bookshop or library application, where the user could be led to the correct section of the bookshop/library, then shown the exact location of their desired book on the shelves in front of them. It could also be used in conjunction with a network of fixed cameras pointed at bookcases so that every shelf in the whole bookshop/library could be searched at once. There is also potential for this software to be used in other domains. For example; the software could be modified for use in supermarkets — to search for grocery items on supermarket shelves.

9.2 Future Work

Topics for further research and development in this area include; dealing with crooked photographs, correcting perspective skewing in photographs, dealing with books that are not upright on shelves, revisiting colour comparison, and live highlighting of the book in an augmented reality format.

Dealing with crooked photographs, and books that are sideways or upside down on shelves, are both relatively straightforward developments. Simply searching for the book spine with the template at four different orientations would allow sideways and upside down books to be found, however this would also make the search routine four times slower. It was not deemed a worthy sacrifice to make the search time so much slower for this purpose, as in libraries and bookshops it is reasonable to assume that almost all of the books should be upright on the shelves, however if search time could be reduced further then this adjustment may become more viable. Correcting crooked photographs might involve finding the orientation of the given image that yields the most vertical Hough line segments when carrying out edge detection, and rotating the image accordingly before segmenting shelves and searching for books.

Handling books that are slanted on shelves, leaning to one side slightly as opposed to lying fully sideways, is more complicated. It may be possible to deal with this by finding regions of the image where the edges of the books are not vertical, and transforming just those regions to fix the slanting.

While CLAHE helps improve colour comparison scores for correct book matches, and the resulting classification function is accurate, there are still some correct matches that produce low colour comparison scores. With further research and experimentation it may be possible to improve the colour comparison process. Looking at different ways of pre-processing the spines to prepare them for colour histogram comparison, or investigating other colour comparison techniques such as Earth Mover's Distance, may produce higher results for correct matches.

Currently the application highlights the location of the book in a static image on screen. Live highlighting of the book in the camera feed on the phone screen using augmented reality would be a nice feature to add to the iOS application. This would involve continuous processing of a video feed from the camera, and live feature tracking.

Chapter 10

Appendices

10.1 More Test Data

This appendix contains more in-depth details of tests and experiments carried out during development of the book-finding system.

10.1.1 CLAHE Clip Limits

This section presents test results and evidence used when choosing the clip limit configuration for CLAHE in chapter 5.2.

Clip limit values can range from 0 to 255. The table in figure 10.1 shows test results for the average, highest, and lowest colour comparison values using CLAHE with different combinations of clip limits for correct and incorrect matches respectively. There is not a huge disparity between some of the combinations when using such low clip limits, as the effects are quite subtle, but it was decided to use a clip limit configuration of 2 - 1 for search result - template image. Configurations of 2 - 0 and 4 - 2 would both also have been suitable choices, but 2 - 1 keeps the average score for an incorrect match lower than either of these two options.

The more extensive tests of the final system presented in chapter 6 show that this configuration yields a good separation between the average scores of correct and incorrect matches. The highest recorded score for incorrect matches is more random and unpredictable than the other metrics — it is based on what colours happen to be in the area landed on by an incorrect greyscale template match. so it was not given much weight when choosing the best CLAHE configuration.

CLAHE Clip Limits: (Template — Search Result Clip Limit)	Correct Match Average	Correct Match Low	Incorrect Match Average	Incorrect Match High
Without CLAHE	0.61	0.024	0.309	0.742
1 — 0	0.66	0.14	0.36	0.76
2 — 0	0.711	0.244	0.417	0.68
4 — 0	0.799	0.38	0.5	0.82
8 — 0	0.85	0.51	0.57	0.9
16 — 0	0.87	0.56	0.63	0.91
2 — 1	0.69	0.26	0.39	0.78
4 — 2	0.74	0.4	0.44	0.82
8 — 2	0.7	0.4	0.46	0.81
8 — 4	0.799	0.46	0.53	0.88
16 — 4	0.7	0.34	0.54	0.82
16 — 8	0.8	0.48	0.59	0.87

Figure 10.1: CLAHE clip limit test results

The above tests show that, without using CLAHE, the average colour comparison score for a correct match was 0.61, and the lowest recorded score for a correct match was 0.024, whereas using CLAHE with the aforementioned clip limit configuration (2 - 1) increases the average score to 0.69, and, more relevantly as far as CLAHE and clip limits are concerned, the lowest recorded score was increased by more than a factor of 10, to 0.26 for correct matches.

10.1.2 Recorded Search Times

Figure 10.2 shows a sample set of search times which were recorded once the system was fully optimised. These searches were carried out using the iOS application. The searches cover both vertical and horizontal phone orientation, multiple different bookcases, and both instances where the was found and instances where the book was not found. These results were used to calculate an average search time for the system.

Time (Seconds)				
1.31	0.23	0.37	0.78	0.81
0.765	1.3488	1.9087	0.8	1.11
1.33	0.89	0.476	0.87	1.168
1.338	0.43	0.8	0.85	1.12
1.6	0.208	0.79	1.0	0.8
0.786	0.19	0.789	0.53	0.824
1.2265	0.59	0.837	0.56	0.82
0.98	1.504	0.573	0.36	1.2
1.45	0.829	0.664	0.653	1.157
0.558	1.039	0.45	0.65	1.47
1.54	0.823	2.076	0.54	1.2
1.54	1.433	0.85	1.187	1.097
1.446	0.672	0.85	1.326	0.657
1.187	0.723	1.676	1.062	1.0435

Figure 10.2: Search times for final system

Average search time = 0.95315

The average search time is 0.95315 seconds.

10.2 Other Approaches

Many different computer vision techniques were investigated, experimented with, and implemented during the course of this project. A lot of these techniques and approaches were not employed in the final system. This appendix details the most influential and informative approaches that were ultimately unsuccessful and not utilised, the reasons why they did not work, and what was learned from experimenting with them.

10.2.1 Segmenting Individual Books

One of the initial plans for this system involved using the edge image to segment each individual book spine on the shelves in the image. That way each book could be isolated and independently examined in various ways. Finding the shelves was the first step in achieving this. However when it came to analysing each shelf and trying to separate out the spines, it became clear that the edge data was simply not good enough to do so. Due to the effects of light and shadows in the environments in which this system is designed to work, as well as the low resolution of smartphone cameras, many book spine edges would be missed in the edge detection phase. Figure 10.3 shows examples of this.

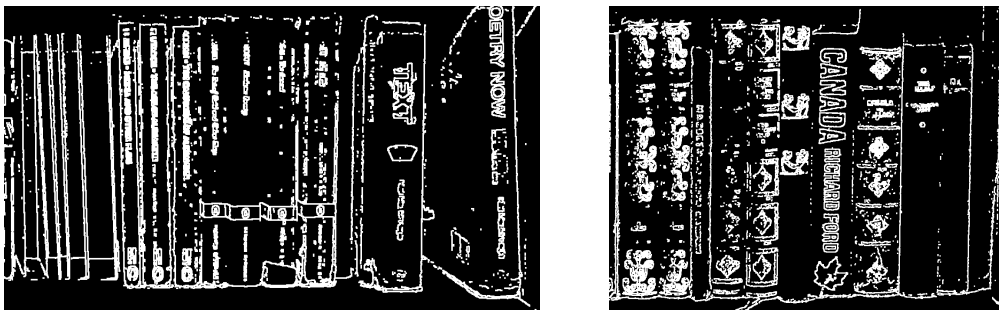


Figure 10.3: Poor edge data

The spines in the two images above are missing a lot of edges, especially where two books meet. The edges are also very fragmented along the tops of the books. Cases like this were found to be very common when examining edge images of bookshelves. This makes accurate segmentation of every book spine infeasible. Figure 10.4 shows an attempt at segmenting book spines using the edge image and a probabilistic Hough transformation for straight line segments. This is a shelf that has comparatively good edge data, and even still less than 50% of the spines are fully and accurately outlined.

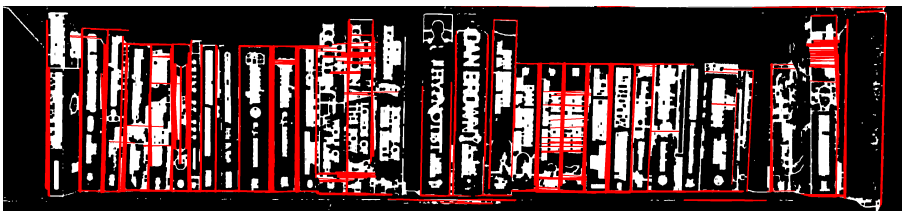


Figure 10.4: Attempt to segment books

As a result, it was decided that a more workable approach would have to start with an image of the book in question, and work towards the location of a spine on the shelves, as opposed to starting with each spine on the shelves and then figuring out what books they match.

10.2.2 Optical Character Recognition

Optical character recognition (OCR) is a computer vision technique that recognises characters in images, so that words and writing can be interpreted from a photograph. When considering how to recognise a book in an image, this would seem like the ideal solution — simply read what it says on the spine. However, the first problem with using OCR is that it requires each book to be segmented so that the text can be read from each spine in the image individually. Section 10.2.1 shows that book spine segmentation was not feasible for this project.

Even if the books could be segmented, when it comes to photographs of book spines with varying designs, graphics and images, and unusual fonts, OCR proved to be sensitive to noise and incapable of coping with non-standard fonts. When experimenting with OCR for this project the Tesseract OCR library was used [9]. Figure 10.5 shows the Tesseract OCR output when given the corresponding book spine image.

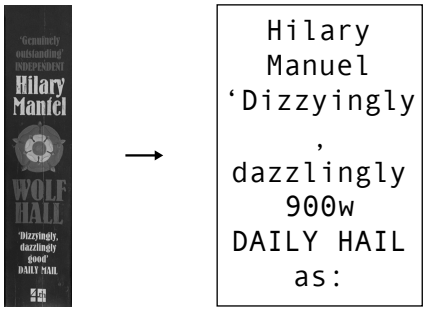


Figure 10.5: OCR output for greyscale spine image

This is a relatively simple book spine, comprised mainly of large, clear text, yet the resulting text recognised by Tesseract OCR is inaccurate and incomplete, and half of the words, including the title of the book, ‘*Wolf Hall*’, do not contrast enough with the background colour of the spine to be recognised at all. Even when the book spine is manually pre-processed to be a binary image with the text in white and the background in black, the OCR still fails. Figure 10.6 demonstrates this.

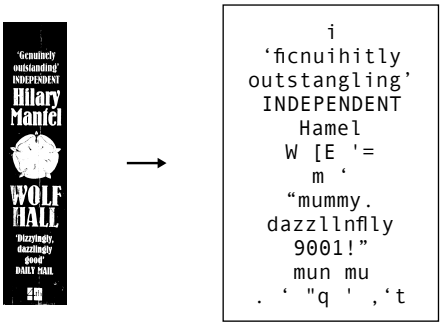


Figure 10.6: OCR output for binary spine image

Even if the output from the pre-processed image was reasonable, it would still follow that each book spine image would need to be pre-processed differently in order to yield the most accurate output from the OCR. It will also not be known what the orientation of the text on the spine is before carrying out OCR, and Tesseract requires the text to be correctly oriented in the given image.

10.2.3 Chamfer Matching

Chamfer matching is a method for comparing images, where the chamfer distance, or dissimilarity, between the template and various regions in a target image, is computed in order to find the least dissimilar overlay position. It is similar to template matching in that it looks for where a template image shows up in a target image, however it typically uses a binary edge image as opposed to a greyscale image, and looks at the distance to the nearest edge in the target image for each edge pixel in the overlaid template image. In some ways this is more robust than template matching, it can better deal with distortion in the target image as it only considers the edge pixels. However, as section 10.2.1 shows, the edge data in photographs of bookshelves taken in the kinds of scenarios that this project addresses — homes, libraries, bookshops, etc. — is not comprehensive enough for accurate chamfer matching. Experiments proved chamfer matching to be to far less successful than template matching when it came to finding books. In light of this, a template match with edge gradients was implemented, the premise being that it was, to some extent, a cross between chamfer matching and regular template matching. Using edge gradients provided the template matching with more object pixels to work with than just using the binary edge image. This was more successful than chamfer matching, but still only 72.5% accurate, whereas template matching with the greyscale spine image was approximately 90% accurate when tested at that time. Therefore greyscale template matching was deemed to be the better matching technique for this project.

It is possible that with further work and experimentation the above techniques might have eventually proven useful, but considering that the other techniques outlined in chapters 4, 5, and 6 produce a book-finding system that is 90.5% accurate, it was not deemed necessary to investigate book segmentation, OCR, or chamfer matching any further during the course of this project.

Chapter 11

Bibliography

[1] Chen, David M., et al. "Building book inventories using smartphones." *Proceedings of the 18th ACM international conference on Multimedia*. ACM, 2010.

[2] Evernote note taking application — https://evernote.com

[3] Cataloguing books with Evernote — http://www.jamierubin.net/2013/04/17/going-paperless-my-virtual-bookcases-in-evernote/

[4] A Practical Introduction to Computer Vision with OpenCV, Kenneth Dawson-Howe, Wiley & Sons Inc. 2014 — https://www.scss.tcd.ie/publications/book-supplements/A-Practical-Introduction-to-Computer-Vision-with-OpenCV/

[5] Stover, Christopher and Weisstein, Eric W. “Polar Coordinates.” From MathWorld A Wolfram Web Resource — http://mathworld.wolfram.com/PolarCoordinates.html

[6] OpenCV Computer Vision Library — http://opencv.org

[7] Swift 3 Grand Central Dispatch documentation — https://developer.apple.com/library/content/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html

[8] AV Foundation Library — https://developer.apple.com/av-foundation/

[9] Tesseract OCR Library — https://github.com/tesseract-ocr/tesseract