Making Test Case Images Searchable

BTECH 451 Mid-Rear Report

Tianyun Gu
Supervisor: Dr Xinfeng Ye

Abstract

This report covers my work over the course of the first semester on the BTECH 451 project with Kiwiplan, which aims to produce a system to increase the ease of locating images for test cases used in the company's automated testing system. The motivation for the project is described in the first section along with an overview of the scope and goals. Functionalities required for the system are discussed, including optical character recognition for extracting text from images to be used for indexing and search. Different options for fulfilling these requirements are considered with respect to challenges faced. I conclude with an illustration of the currently implemented system architecture as well as work to be completed in the future.

Contents

1 Introduction

This section provides context and introduces the aims for this BTECH 451 project.

1.1 Project Overview

Kiwiplan is an industry software solutions development company, which produces software for use in many areas of the supply chain in manufacturing of rigid packaging. Automated testing of their software is performed by an in-house developed system specifically for their products, named Droid. Tests are performed by checking for discrepancies between expected results in the form of a screen capture image and the screen produced from running the test case. While this approach to automated testing may not be considered the best solution today, Kiwiplan was an early adopter of an automated approach to testing, such that off the shelf solutions were not readily available at the time of Droid's development. Justification for the continued use of Droid comes from the system being built to meet the specific needs of the company, and regardless of whether other approaches may be adopted in the future, there is benefit in continuing the support of legacy applications.

One challenge currently posed by Droid is the difficulty in matching the output produced by tests results to the test cases which produced them, due to there being quickly recognisable connecting indicators. Manually locating individual test case images from the large (and continuously growing) number of entries present in directory structures is tedious so in order to facilitate continued growth, a solution to perform automated search of test cases is proposed. This project aims to develop an application to address such needs

1.2 Solution Concept

In order to address the challenge described above, the proposed solution is to develop a system which can be used to search for and retrieve base case images through the text strings which appear on the image. A user should access the search interface in the form of a web based service and search results should be returned in a reasonably quick time frame. A number of tasks must be performed by the application as part of this:

1) **Convert image file type:** All Kiwiplan's test case images are encoded in a proprietary file format, CBMP, which provides very efficient image compression. While this format can be used by Droid, it must be converted to a commonly useable format for processing.
2) **Extract text from image:** Text that is present on an image should be extracted and enabled for search. This is performed through optical character recognition (OCR).
3) **Images indexed with search terms:** After extracting text from images, this information needs to be stored in a method that will allow for search in an efficient manner.
4) **Communication through web based service:** Other machines on the network should be able to access the search interface to locate images not stored on the local machine.

The most significant components of the project fall under tasks 2) to 4) and the majority of work during the first half of the year has been focused on the second and third tasks. Because the project is primarily interested in achieving a functioning system at this stage, I focus on examining the use of third party packages to achieve these goals.

1.3 Related Work

A similar pipeline to that identified in the goals for this project has been used to make collections of text fully searchable, such as for online libraries or document archives. I considered two examples to examine the tools utilised in achieving this, the first from the World Digital Library [1] and the second from the New Zealand Journal of History at the University of Auckland [2].

The World Digital Library created a system to facilitate search of text in digitalised books from scanned pages using the Tesseract OCR engine and Apache Solr for indexing and search. It was noted that inaccuracies in the OCR results were acceptable as the original page scans were presented to the user, rather than the OCR text itself. OCR was only used for the purpose of indexing pages, and while inaccuracies do lead to a decreased hit rate for search results, the amount of existing content which would otherwise be completely unsearchable means that the benefits outweigh the loses. In comparison to the scope of my project, incorrect OCR results may be far less desirable.

The New Zealand Journal of History is an academic journal which began publication in 1967 and offers online access to issues. Issues older than 2003 have been made available through scans of printed copies, with the content extracted through OCR. The engine utilised is ABBYY Finereader, with a reported accuracy rate of 99.5%. It is difficult to determine the significance of this value without other comparisons.

2 Optical Character Recognition

Optical character recognition, or OCR, is the process through which images of text are converted into a machine readable format, for the purposes of digitally editing or searching, for example. Images can be sourced from a variety of methods including photographs or document scans and for the purposes of this project, a capture of what would be screen display. Common applications of OCR include transcribing printed documents to digital form, data entry from records in physical form, and conversion of handwritten text.

2.1 OCR Engines

2.1.1 Evernote

Evernote is an application used for digitally creating and storing ‘notes’ with support for a range of media types, including both text and images. OCR is included in the functionalities, as part of extending the ability to perform search on content to all forms of media which may be contained on a note. As part of the initial project concept discussion, demonstrations of using a Droid test image with Evernote without any image preprocessing appeared to be

surprising successful. Further research into how Evernote performs OCR reveals a fairly complex process. The 'Evernote Indexing System' is responsible for making all content of a note searchable and is invoked each time a user upload one containing some data matching the MIME type of an image. Images then go through a series of 'passes' of OCR, each with different assumptions about how text maybe appear on the image, and each set of assumptions gives rise to different techniques used during preprocessing the image [2].

Where many traditional applications of OCR place high value on the accuracy of results, the end goal from this is to extract as many text strings as possible from the content of an image such that indexing and search will provide comprehensive cover. The end result from this process is a set of possible alternatives for each word identified on the image, each associated with a weight value, w, representing the confidence in its correctness. This information is embedded in the metadata of an image-containing note such that when a user performs a search for a phrase, each alternative listed for all text identified on an image is checked for a match [3]. In this way, Evernote's approach to OCR introduces a high level of fault tolerance as it does not rely on the decisive results from a single pass of character recognition.

Evernote's API is publically available however the user license agreement forbids the exclusive use of the OCR services. For the purposes of this project, building a system of this complexity is unrealistic due to time and resource constraints. One factor which may be of interest to consider if time permits in the latter part of the project is the use of a probabilistic approach when indexing images, with differing techniques used during preprocessing to produce alternative character recognitions.

2.1.2 ABBYY Finereader

ABBYY Finereader software is geared towards converting scanned images into an editable form and includes capabilities to recognise and maintain document formatting during the conversion. While this may be useful if we are interested in keeping the integrity of the document content, such as in [2], this extra information is not useful for the purposes of this project, where we are only interested in using OCR for indexing. Furthermore, because the images to be worked with are not scanned documents, I cannot be certain that this would produce the same level of accuracy as described earlier. It is likely that such an OCR solution which is aimed to be used with scanned documents will not produce the best results when applied to screen images, as will be illustrated in section 2.2. Another drawback of this system is the limited technical documentation available, due to the software falling under a proprietary license, so it overall its usefulness in this project seems limited.

2.1.3 Tesseract OCR

The Tesseract OCR engine was original developed by Hewlett Packard and is currently open source. One key feature in the way Tesseract performs OCR is the use of an 'adaptive classifier', which actively learns from the content as it is processed, such as estimating character widths, to improve accuracy of information which is later processed. Character recognition is performed in two steps. The first pass sends successfully classified characters to the adaptive classifier as training data and the second pass will attempt to reclassify words which were uncertain on the first pass [6].

Another feature of Tesseract OCR is its ability to detect and adapt to the baseline of text. Because Tesseract is purely an OCR engine and does not perform any preprocessing on images before character recognition, this capability may be useful if text on images do not follow a perfect vertical or horizontal alignment. This feature is not very valuable for the purposes of this project however, as input images will not suffer from such problems. The lack of preprocessing performed by Tesseract OCR engine does provide one benefit in that there is greater control over how images are handled before character recognition is performed, which proves to be useful due to the unconventional nature of the images used. This is elaborated on in the next section.

One drawback of Tesseract OCR is the use of polygon approximations for characters identified in an image. The typeface of text in Droid images is a uniform thickness of one pixel, with serifs, as shown in Fig. 1. Because even slight alterations to characters in this style may have a very large impact on the character apparently appearing on the image, any inaccuracies introduced by approximations of the shape of characters may compromise the accuracy of OCR results. All things considered, the flexibility provided by this engine seems the most appropriate for use with this project.

2.2 Challenges

Several factors influence the accuracy of OCR results, aside from the algorithms used. These come from qualities of the image content, more specifically, the integrity of the text which OCR intends to capture. In common applications of OCR, such as with scanned documents or photographs of text, factors which may reduce the quality of OCR results include the presence of noise or artifacts in the image, distortions to the text [5]. To achieve better results, pre-processing and image before OCR aims to correct these issues.

Because images used by Droid are originally digital, much of these factors will not be present. However, other challenges are encountered due to the nature of such images. Most OCR packages are not created to be used with images generated from on screen display. For example, a minimum resolution of 300dpi is recommended for images used in OCR, compared to the standard 72dpi used on computer monitors. This means that although images used by Droid contained perfectly clean text, performing OCR on these images without any preprocessing will still produce very poor results.

Fig. 1 is an example of an image used by Droid before performing any preprocessing, and Fig. 2 shows the output given by Tesseract OCR from this image. Despite the text contained in this image being very easily readable for a human, OCR results are extremely inaccurate. Other issues demonstrated here include:

- Presence of undesirable words which appear on the interface.
- False positives caused by interface elements.
- Image border interfering with character recognition (“Num” interpreted as “um” due to the left border).

Evidently, there is a strong need for preprocessing to improve accuracy to a usable level.

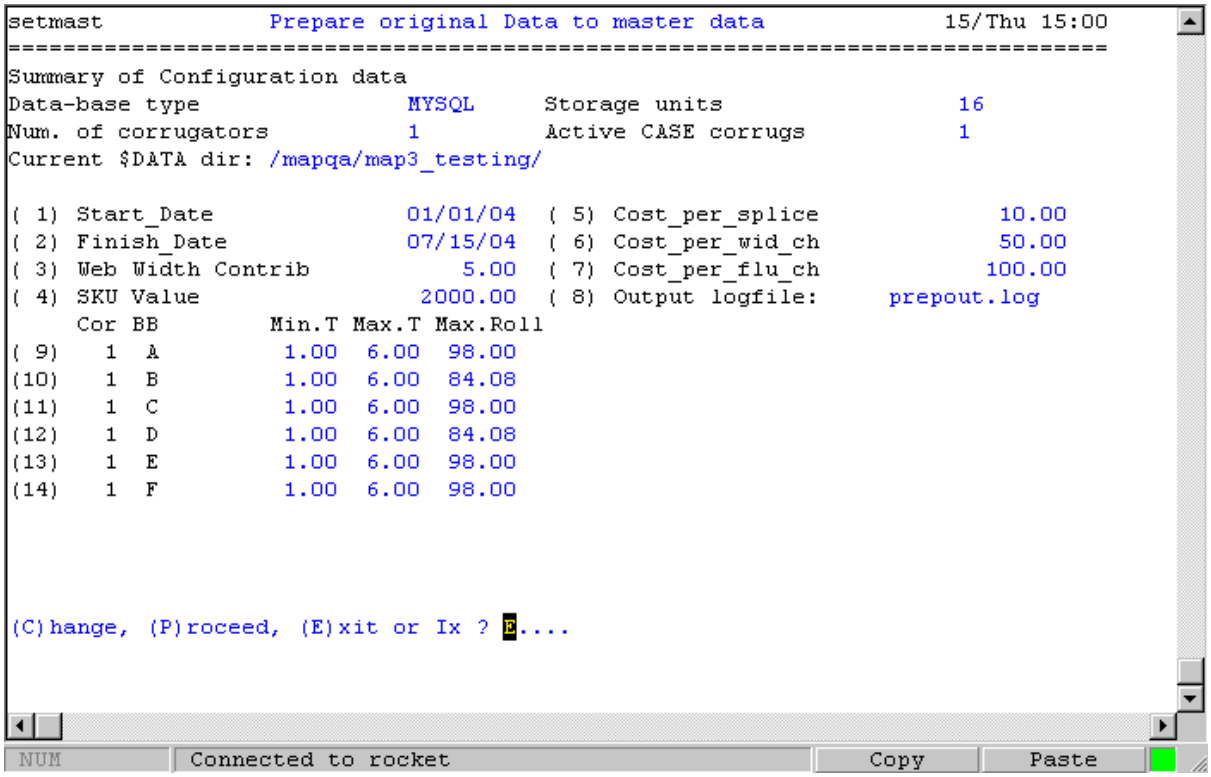

Figure 1. Example image before any preprocessing.

```
10
11  secmas: PrEpare urlglna) Data to master data JS/Thu 15 an
12  Summary of Configuraclun data
13  Dacarhase type HYSQL Storage unlcs 1
14  um. of corrugacurs 1 Acclve case corrugs 1
15  Current snara a1 /mapfia/map37cesc)ng
16  L 1L Scarcibace u1/u1/u4 L 5L Cusciperispllce 1u.uu
17  L 2L Flnlshibace u7/15/u4 L 5L Cusciperiwldich 5n.uu
18  L 3L "eh "ldch Concrlh 5.uu L 7L Cusciperiiluich 1uu.uu
19  L 4L sxu Va1ua zuuu.uu L 5L Output Juginle. paapaaa.1ag
20  CD: as u1m.r Hax.T Hax.Rull
21  L 9L 1 a 1.uu s.uu 95.nu
22  L1uL 1 a 1.uu s.uu 54.ua
23  L11L 1 c 1.uu s.uu 95.nu
24  L12L 1 n 1.uu s.uu 54.ua
25  L13L 1 a 1.uu s.uu 95.nu
26  L14L 1 r 1.uu s.uu 95.nu
27  (CyhangEy LPLIDceed, LELx): or IX 7 a
28  4 »
29
30  17mm y—cammaaaaa ma tucker. Copy Paste l 1
31
```

Figure 2. OCR Result from Fig. 1.

2.3 Preprocessing

The most obvious first step to address the issues introduced above is to alter the resolution of the image, and the simplest solution to this is scaling up. Through experimenting I arrived at enlarging the image around 400% to be the limit of accuracy gains from simple scaling. Although this now creates blurriness in the image, this alone significantly increases accuracy over the original image, shown in Fig. 3 (refer to appendix for complete illustration of OCR results from this). Cropping the image to remove the borders and interface also solves a number of the previously mentioned issues.

This result is not perfect. In particular, certain characters appear to be consistently interpreted incorrectly, such as 'M' and bracket characters. Another noteworthy observation from this output is the differences in text formatting in comparison to Fig. 2, however, white space will not be considered when indexing images so it is not necessary to consider.

Beyond these there are no obvious indicators for how improvements to the OCR result could be achieved through editing the image during preprocessing, and it maybe be a matter of conducting continued experimentation to achieve the most accurate results. At this stage, this level of accuracy is deemed acceptable in the interest of advancing project progress. If time permits there is potential to return to this to make further refinements.

```
37
38  Start_Date 01X01X04
39  Finish_Date 07X15X04
40  Heb Uidth Contrib 5.00
41  SKU Value 2000.00
42  Cor BB Hin.T Hax.T Hax.Roll
43
44  1 A 1.00 6.00 98.00
45
46  1 B 1.00 6.00 84.08
47
48  1 C 1.00 6.00 98.00
49
50  1 D 1.00 6.00 84.08
51
52  1 E 1.00 6.00 98.00
53
54  1 F 1.00 6.00 98.00
55
56  (Clhange, (PJroceed, (Eint or Ix ? I....
57
58  Cost_per_splice
59  Cost_per_wid_ch
60  Cost_per_flu_ch
61  Output logfile:
62
```

Figure 3. A sample of OCR output from image scaled up to 400%.

3 Indexing

This phase of my project investigates the storing of OCR results for search. An important consideration is compatibility with Kiwiplan's existing systems, which utilise MySQL databases. Although many indexing tools offer their own databasing options, this is not considered ideal due to the need to keep systems consistent for in-house applications for ease of maintenance. Indexing is not fully implemented at this stage of the project and this section will be expanded on in the future.

3.1 Xapian

The first option considered was Xapian, an open source search engine library which supports parallel communication with multiple databases. The most attractive feature of Xapian is the capability for probabilistic search [7], which would be useful for implementing a system with similarities to Evernote's indexing of images. Xapian performs indexing by defining a set of documents, $\{D_1, D_2, \dots, D_i\}$, and index terms, $\{t_1, t_2, \dots, t_j\}$ [8]. Each index term has an associated list of documents indexed by it so that when a search is performed using a specific term, the results are the entries in this list. This is an efficient method of storing index mappings so queries can be processed in a fairly quick amount of time.

One drawback of Xapian is the lack of a Java interface. Although java bindings exist, the use of JNI removes platform independency. Furthermore, because there are no dependencies available online, to use Xapian would require the entire library be included with the application and potentially rebuilt where ever it is to be run. This, along with the lack of complete documentation for its use, may cause future issues if used.

3.2 Hibernate Search

A pure java implementation provides the most convenience for both building the system and future maintenance. Hibernate Search, build on Hiberate ORM, allows java objects to be indexed for search using object properties as index terms, as defined by the developer through either annotations or an XML configuration file. Java objects are first converted to database objects through ORM. One advantage of this is that there is direct control over the structure of the relations used to store indexes, as the method for creating mappings are defined by the java application, along with which object properties are indexed.

3.2.1 ORM

Object-Relation Mapping, or ORM, is the conversion of data objects in an object-orientated paradigm to relations in the relational model for data. ORM will add additional computation overheads to the system, however this a trade-off for removing the need to consider additional data structures when sending and retrieving information from the system and database.

The use of Hibernate ORM will often denormalise data, so some consideration must be given to the intended structure of relations to reduce redundancy in the resulting relations. Because the objects used by the system are few and simple; images and index terms with a single many-to-many relationship between them, this is not expected to pose too many problems.

4 System Architecture Overview

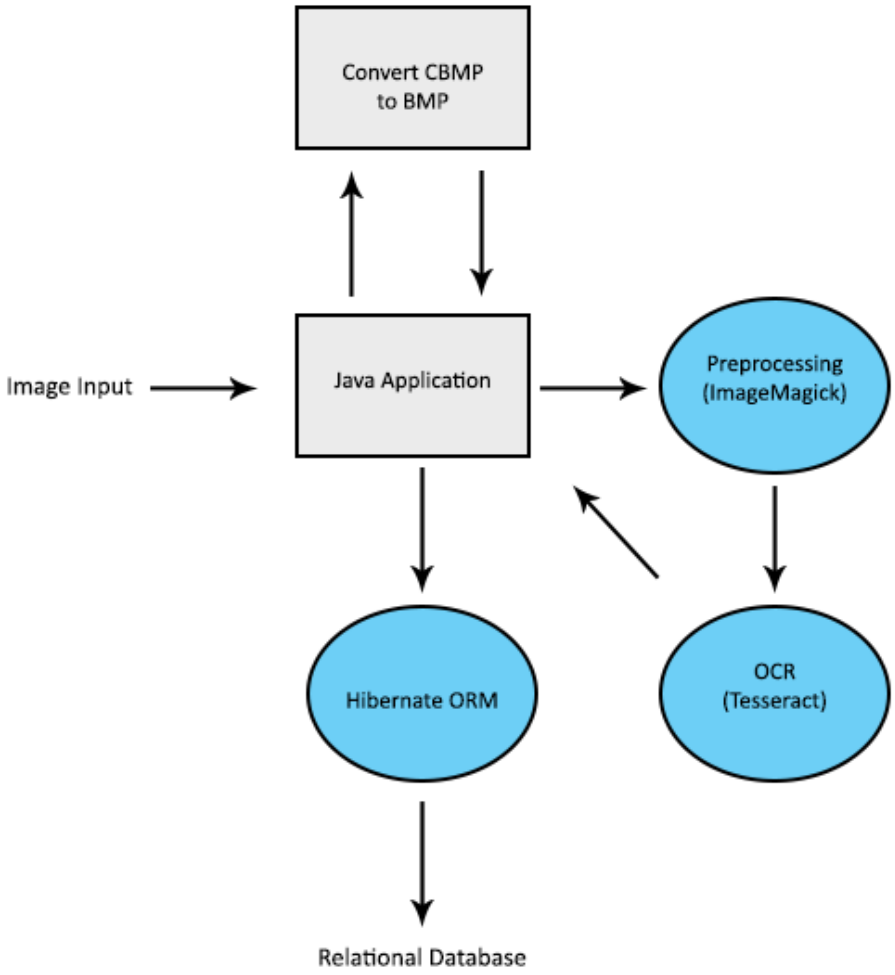

Figure 4. Image processing architecture.

Fig 4. illustrates the implemented system architecture and interactions between components.

4.1 Image Preprocessing

When the system receives an image to be indexed to the database, the first step is to decompress the CBMP file. Because this file type was developed by Kiwiplan to be used with Droid, there is no common method for decompression and a conversion program is called by the system as a new process to perform the conversion. This image is then read by the system and passed on to ImageMagick [9] to continue preprocessing, as described in section 2.3.

4.2 Data Representation

A class is defined by the system to represent image data. The image itself is stored as a file path String, and has an associated set of index terms containing the output produced by

Tesseract OCR. Hibernate ORM converts these objects into relational form to be stored in an SQL database.

5 Future Work

The first semester's work has focused on implementing basic features for processing and extracting text from test images to be used for indexing. The next stage of the project will be geared towards the web-based portions of the system, so that an end user may query the database through a graphical user interface to retrieve images relevant to the search terms. Indexing must be fully implemented, including a finalised design for the tables to be used by the database and the representation of mappings between indexed images and index terms.

A web-based GUI will be implemented to facilitate searching of images. Results from the search must be handled such that they are easily interpreted by a user. If the number of hits exceeds an amount reasonable to displayed at once, some method of paging should be included to improve usability. Initially, results returned will present only the file path associated with the image to the user, however if time permits improving functionality of the system might extend to allowing the user to view the image from the interface.

6 Appendix

```
Prepare original Data to master data

15XThu 15:00

Summary of Configuration data
Data-base type

Num.

of corrugators

HYSQL

1

Current $DATA dir: Xmapqafmap3_testing/

I 11
I 21
I 31
I 41

I 91
I101
I111
I121
I131
I141

Storage units
Active CASE corrugs

I 51
I 61
I 71
I 8)

Start_Date 01X01X04
Finish_Date 07X15X04
Heb Uidth Contrib 5.00
SKU Value 2000.00
Cor BB Hin.T Hax.T Hax.Roll

1 A 1.00 6.00 98.00

1 B 1.00 6.00 84.08

1 C 1.00 6.00 98.00

1 D 1.00 6.00 84.08

1 E 1.00 6.00 98.00

1 F 1.00 6.00 98.00

(Clhange, (PJroceed, (Eint or Ix ? I....

Cost_per_splice
Cost_per_wid_ch
Cost_per_flu_ch
Output logfile:

16
1
10.00
50.00
100.00

prepout.log
```

7 References

[1] C. Adams, "Making Scanned Content Accessible Using Full-text Search and OCR", *Blogs.loc.gov*, 2014. [Online]. Available: https://blogs.loc.gov/digitalpreservation/2014/08/making-scanned-content-accessible-using-full-text-search-and-ocr/.

[2] "New Zealand Journal of History", n*zjh.auckland.ac.nz*, 2016. [Online]. Available: http://www.nzjh.auckland.ac.nz/.

[3] A. Pashintsev, "Evernote Indexing System", *Evernote Blog*, 2011. [Online]. Available: https://blog.evernote.com/tech/2011/09/30/evernote-indexing-system/.

[4] B. Kelly, "How Evernote's Image Recognition Works", *Evernote Blog*, 2013. [Online]. Available: https://blog.evernote.com/tech/2013/07/18/how-evernotes-image-recognition-works/.

[5] J. White and G. Rohrer, "Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction", *IBM Journal of Research and Development*, vol. 27, no. 4, pp. 400-411, 1983.

[6] R. Smith, "An overview of the Tesseract OCR engine". In *icdar,* pp. 629-633, IEEE, 2007

[7] "The Xapian Project : Features", Xapian.org, 2016. [Online]. Available: https://xapian.org/features.

[8] "Theoretical Background", Xapian.org, 2016. [Online]. Available: https://xapian.org/docs/intro_ir.html.

[9] I. LLC, "ImageMagick: Convert, Edit, Or Compose Bitmap Images", Imagemagick.org, 2016. [Online]. Available: http://www.imagemagick.org/script/index.php.