Automating the process of dividing a map image into sections
- using Tesseract OCR and pixel traversing

Automatisering av processen att dela in en kartbild i sektioner
- Med hjälp av Tesseract OCR och pixel traversering

Lukas Vikström, Jesper Skoglund

Tutor : Dennis Persson
Examinator : Jody Foo
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/her own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Lukas Vikström, Jesper Skoglund
Automating the process of dividing a map image into sections using Tesseract OCR and pixel traversing

Lukas Vikström
Linköpings university
Linköping, Sweden
lukasvikstrom@gmail.com

Jesper Skoglund
Linköping university
Linköping, Sweden
jesper.skog@hotmail.com

ABSTRACT
This paper presents an algorithm with the purpose of automatically dividing a simple floor plan into sections. Sections include names, size and location on the image, all of which will be automatically extracted by the algorithm as a step of converting a simple image into an interactive map. The labels for each section utilizes tesseract-OCR wrapper tesseractJS to extract text and label location. In regards to section borders pixel traversing is employed coupled with CIE76 for color comparison which results in the discovery of size and location of the section. Performance of the algorithm was measured on three different maps using metrics such as correctness, quality, completeness, jaccard index and name accuracy. The metrics showed the potential of such an algorithm in terms of automating the task of sectioning an image. With results ranging between lowest percentage of 48% and highest of 100% on three different maps looking at correctness, quality, completeness, average jaccard index and average name accuracy per map.

Author Keywords
Interactive Map; OCR; Angular; Floor plan; sectioning.

INTRODUCTION
Early maps although rather basic in their functionality, contained needed information about the general surrounding area used during navigating. Modern technology now allows us to revitalize these primitive physical maps into interactive applications with wider scale alongside intuitive user interaction. Examples of interactive maps found online are Google Maps¹ and Bing maps ². The definition of an interactive map was stated by A. Çöltekin et al. [3] to consists of two different elements, the map itself and an interactive GUI used to enable user interaction.

World maps such as Google Maps or Bing maps contain not only information about the world in a general geographical aspect, it contains details as far down as to a vendor on a street level. Despite these online maps containing large amounts of locational data there are still cases where smaller and more specific maps are needed. Common types for these specific maps are those categorized as floor plans, which show the static location of objects or areas for each level in a building. Although not dynamic nor interactive these floor plans gives the user a graphical understanding of where an object might be located. Our project will be focused on automating the process of converting these floor plans to interactive elements, which allow user interaction.

Motivation
The conversion of a simple image into an interactive element usually requires the user to initially spend time sectioning and configuring the parts of the image which should be intractable before it can actually be used as an interactive map. Sectioning will be the term we define as the initial step in our system where the image is converted into an interactive map. Instead consider an option where an algorithm automatically sectioned and named the sections based on an image, allowing the system to create interactivity based on these automatically generated elements.

Consider a scenario where an office building consists of 30 floors where each floor consists of multiple sections. In our system this would translate to 30 different images each of them requiring sectioning. In order to prepare the images of different floors a user would be required to spend time sectioning every floor manually within our system. Although this process only takes place once for each floor, the time spent manually configuring every floor could be significant. With an automated algorithm the time the user would have to spend preparing the system would be limited to confirming the findings of the algorithm.

Aim
Using standard images of indoor maps we aim to convert these into interactive elements, creating a sort of hybrid between simple images of maps and the modern interactive maps. We will be exploring an automated solution to this conversion which will be further explained in later sections. Using the floor layout of a factory to improve ease of navigation or to optimize the work flow could be one of the use cases for such a hybrid system.

We will use the Angular framework³ to develop a basic interactive map. The development will take place in collaboration with IFS⁴. Our focus will not be to create an extensive interactive map in terms of functionality, rather to create a base which we can test our automated conversion method. The map image should be no different than a simple image of the indoor

¹https://www.google.com/maps
²https://www.bing.com/maps
³https://angular.io
⁴http://www.ifsworld.com
location such as floor plan. Our aim is to develop an algorithm inside our system to parse the map image and automatically add sections based on the image. We will then compare the results from different maps using metrics show the performance and viability of such an algorithm.

Research questions

How does an automated algorithm for identifying sections on an image perform compared to the manually defined gold standard in terms of:

- Measuring the results in terms of discovered sections.
  - Correctness
  - Quality
  - Completeness
- The attributes of the discovered sections:
  - Jaccard index - How much of the automatically discovered section overlaps with the section in the gold standard in terms of coordinates and size.
  - Name accuracy - How accurate the discovered name was compared to the gold standard.
- Time taken for our algorithm to achieve the results.

The metrics will show the performance of our algorithm in terms of the produced sections. Due to the algorithm containing multiple steps in asserting location and size of a potential section we require different metrics for evaluating these sections. Correctness, quality and completeness validates the amount of sections found compared to what the image actually contains. Jaccard index and name accuracy will calculate the algorithm’s performance in terms of the section attributes such as size, placement and name.

LIMITATIONS

Our algorithm demands the presence of certain assumptions in order to properly explore the image and retrieve sections. Creating a more general algorithm with a wider application area deemed to be outside the scope of our project, instead we focus on the viability of such an algorithm as a proof of concept.

Map design

In terms of map design we decided to limit the map choices we tested. We will be using maps which for the most part contains clear section borders in terms of color, the text on the map should not constitute anything other than section name and the section shapes should mostly consists of rectangular shapes.

THEORY

Color measuring

Formulas for measuring differences between colors has been researched for many years. G. Xiong et al. [9] examined different ways of comparing differences between two colors. In which they stated that the CIE\(^3\) has throughout the years released different recommended formulas for this purpose. In 1976 they recommended the CIE76, which uses Euclidean distance and is defined as

\[ \Delta E_{ab}^* = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2}. \]

CIE later recommended the CIE94 and then the CIED2000 which contrary to its predecessors is not Euclidean distance based.

\[ \Delta E_{94}^* = \sqrt{\left(\frac{\Delta L^*}{K_L} + \frac{\Delta C^{ab}}{K_C} + \frac{\Delta H^{ab}}{K_H}\right)^2} \]

\[ \Delta E_{00}^* = \sqrt{\left(\frac{\Delta L^*}{K_L} + \frac{\Delta C^{ab}}{K_C} + \frac{\Delta H^{ab}}{K_H}\right)^2 + \left(\frac{\Delta R^*}{K_R} + \frac{\Delta G^{ab}}{K_G} + \frac{\Delta B^{ab}}{K_B}\right)^2} \]

In their research [9] concluded that the CIE76 was the fastest formula for measuring color difference, while CIED2000 was the most advanced

Algorithm metrics

K. Borna et al. [2] compared the difference between two models by using three measures: correctness, completeness and quality. These were calculated using three classifications: true positive, false negative, false positive. Definition of the classifications was stated by Borna to be

“[...] Positive (TP), correctly detected pixels, False Positive (FP), falsely detected pixels, and False Negative (FN), incorrectly identified pixels.” [2]

Correctness = \( \frac{True \ Positive}{True \ Positive + False \ Positive} \)

Completeness = \( \frac{True \ Positive}{True \ Positive + False \ Negative} \)

Quality = \( \frac{True \ Positive}{True \ Positive + False \ Negative + False \ Positive} \)

OCR

Measuring the performance of different OCR scan engines was conducted by S. Rice, F. Jenkins and T. Nartker [8]. They evaluated at the time five leading OCR engines which were tested on over 2000 pages containing a sum of 5 million characters. In order to measure how well an engine performed they measured values such as character accuracy and throughput. Where n is amount of characters in each text and P is the penalty value assigned for each error:

characterAccuracy = \( \frac{n - \#errors}{n} \)

Throughput = \( \frac{n - P\#errors}{\#seconds} \)

Furthermore they examined the effects such as image resolution along with page quality on the results of the scans and the performance of the engines.
RELATED WORKS

OCR
The are multiple engines in the area of Optical Character Recognition such as Tesseract, Ocrad and Asprise OCR to name a few. These are engines used to detect text on images, which can in turn be further processed or simply displayed. G. Zhao et al. [10] used Tesseract to detect barcode defections. Images of barcode was segmented using a horizontal projection algorithm, upon which the authors used Tesseract-OCR to identify characters on segments. To further improve upon Tesseract-OCR results Zhao also preprocessed the image with Otsus method, an image threshold algorithm which reduces grayscale image to a binary image. Furthermore they manually train Tesseract to improve the detection on certain fonts related to the area. As a result the system achieved a 93.4% accuracy with limited sample size.

Image processing algorithm
A study was conducted with the goal of extrapolating only the map elements from an image of an indoor map. In the study Honto et al. [6], applied different algorithms and machine learning to cut out the parts of image which did not represent nor contribute to the map. For this purpose [6] employed the snakes and the grabcut algorithm. Both of these required the user to specify the general area which included the map. While the grabcut algorithm also required the foreground and background color codes to properly differentiate between the elements. Honto determined that for this scenario the grabcut algorithm produced results was deemed superior to that of the snakes, despite the increased manual input [6].

Floor plan partitioning
An algorithm for partitioning floor plans was developed by S. Dodge et al [5]. Using a users manual sections dividing as a gold standard to which they then compared the implemented algorithm with. This was built with the assumption that each label on the image had a section associated with it. The metrics used to evaluate the algorithm where:

- Jaccard index - how similar their created sections were to the ground truth.
- Labels - if each label had a section surrounding it.
- Overlap - if the sections were overlapping each other.

S. Ahmed et al [1], M. Sébastien et al [7] and L de las Heras et al [4] all used room boundaries to partition a floor plan. What differed in their work were how they extracted the floor plan structure. Ahmed [1] used OCR as a part of extracting room labels in order to determine their function. Processing the walls by utilizing morphological binary erosion which strips away inner and outer bounds pixels from an object, after which the opposite morphological binary dilation was applied which adds pixel layers to both inner and outer bounds of the object. This process resulted in an image where only the thick walls on the the floor plan remained. Sébastien [7] detected walls in a floor plan by utilizing classical hough transform which detects long lines, alongside image vectorization used to detect smaller lines. By coupling the two methods Sébastien was able to create a prominent line detection algorithm. De las Heras [4] implemented a pixel based approach to extract objects in a floor plan. They categorized pixels into three categories walls, doors and windows.

METHOD

The method is divided in two different parts. First we developed an algorithm to automatically section the image. We then needed to measure the viability of said algorithm on different maps using different metrics including correctness, quality, completeness, name accuracy, jaccard index and time taken.

The algorithm
Ahmed [1], Sébastien [7] and de las Heras [4] all had different methods for extracting boundaries within a floor plan. We decided to implement our own method which drew inspiration from their research. The most prominent idea was to retrieve the walls around a section. We also followed [4] example of using pixels but instead of categorizing pixels we traverse and compare their color as a method of finding the stop points.

The algorithm’s purpose is to split an image of an indoor map into different sections, see Figure 1 and 2. It consists of different steps which are executed for every found section in the image. The algorithm we developed and implemented drew inspiration from the work conducted by Honto et al. [6], in the sense of sharing the goal of extrapolating key areas of the image. With the conclusion that grabcut emerged as the superior algorithm in their specified scenario, we determined that some sort of color comparison would prove to be a key aspect in our own algorithm. Unlike Honto’s implementation however our algorithm does not rely on user input for determining needed color values. Instead the algorithm itself calculates the stop colors in the initial steps for each section to properly determine the borders of said section.

S. Ahmed et al [1], M. Sébastien et al [7] and L de las Heras et al [4] all used room boundaries to partition a floor plan. What differed in their work were how they extracted the floor plan structure. Ahmed [1] used OCR as a part of extracting room labels in order to determine their function. Processing the walls by utilizing morphological binary erosion which strips away inner and outer bounds pixels from an object, after which the opposite morphological binary dilation was applied which adds pixel layers to both inner and outer bounds of the object. This process resulted in an image where only the thick walls on the the floor plan remained. Sébastien [7] detected walls in a floor plan by utilizing classical hough transform which detects long lines, alongside image vectorization used to detect smaller lines. By coupling the two methods Sébastien was able to create a prominent line detection algorithm. De las Heras [4] implemented a pixel based approach to extract objects in a floor plan. They categorized pixels into three categories walls, doors and windows.

METHOD

The method is divided in two different parts. First we developed an algorithm to automatically section the image. We then needed to measure the viability of said algorithm on different maps using different metrics including correctness, quality, completeness, name accuracy, jaccard index and time taken.

The algorithm
Ahmed [1], Sébastien [7] and de las Heras [4] all had different methods for extracting boundaries within a floor plan. We decided to implement our own method which drew inspiration from their research. The most prominent idea was to retrieve the walls around a section. We also followed [4] example of using pixels but instead of categorizing pixels we traverse and compare their color as a method of finding the stop points.

The algorithm’s purpose is to split an image of an indoor map into different sections, see Figure 1 and 2. It consists of different steps which are executed for every found section in the image. The algorithm we developed and implemented drew inspiration from the work conducted by Honto et al. [6], in the sense of sharing the goal of extrapolating key areas of the image. With the conclusion that grabcut emerged as the superior algorithm in their specified scenario, we determined that some sort of color comparison would prove to be a key aspect in our own algorithm. Unlike Honto’s implementation however our algorithm does not rely on user input for determining needed color values. Instead the algorithm itself calculates the stop colors in the initial steps for each section to properly determine the borders of said section.

Figures 1. Illustration of example map

6https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm#erosion
The primary step of the algorithm is to locate potential sections on the image. We will be using OCR to extract text on the map which in turn will be seen as a potential section. This is built on the assumption that any text found on the image should be classified as a section name. Although the results of an OCR scan can vary depending on the engine used [8], we decided to implement a wrapper of Tesseract OCR in this project. Tesseract OCR appeared to be one of the very few which was directly compatible with our Angular system, which would allow direct integration with our system and enable the user to toggle between OCR and manual mode.

The results after the OCR scan was concluded, consisted of bounding boxes where text was discovered, along with the text itself. Bounding boxes coordinates allowed us to further calculate where a section should be based on the relation between bounding box and section borders, which will be further explained in the next section.

In order to alleviate the issue of finding text or symbols where in reality there is none we added basic safeguards to ignore a section deemed incomplete. The first safeguard was added due to the realization that many of the nonsense text consisted of symbols rather than letters, thus we added a blacklist to TesseractJS which it should ignore during scanning.

blacklist: : ! / ; _ ? = — , . | ' ´¨

Furthermore we decided to remove sections which name only consisted of one letter or symbol in total, along with removal of names only containing one letter or number and rest symbols.

Scanning was included to find the initial location of sections on the image, which then can be used to extrapolate section coordinates and size of the section.

**Pixel traversing**

The next step of the algorithm is to use the found text coordinates in the previous OCR scan as starting point and traverse different directions of the image to find the edges of the section. In other words this step aims to discover the attributes of a section such as the x and y coordinates along with the width and height. For the algorithm to be able to traverse the pixels of the image we used a HTML canvas[7] which allowed us to traverse pixel by pixel and extract RGB colors of these individual pixels.

Below the different traverse steps will be explained, each in their own direction and purpose. As seen in Figure 3 we traverse from x0, y0 in the left direction (x - 1) until we encounter stop mark. Similarly we traverse from x1, y1 in the right direction (x + 1) until another stop mark. We define the stop mark as a set of pixel coordinates where a significant color change occurs compared to the starting color. After these steps the two colors which resulted in a stop mark for each direction are saved for later traversals as stop colors. Using two stop colors instead of a single one was in order to limit the impact of encountering a small group of pixels with a rare color. This is an event which dramatically could impact correctness of the section in later traversals. If the stop color was defined as a one occurrence color, the algorithm would most likely fail to find the proper dimensions for the section due to a stop color never being reached.

The remaining traversals are fairly similar with the exception of the direction which they traverse in. In their specific direction, they traverse until they encounter one of the stop colors which was found in the initial two traversals in Figure 3. The coordinates where each traverse ended are then also stored in order to extrapolate respective values. To see examples of the directions see Figures 4 - 8.

\begin{align*}
x &= \text{Top}_\text{stop}.x \\
y &= \text{Top}_\text{stop}.y \\
\text{Height} &= \text{Math.abs(Right}_\text{stop}.y - \text{Bottom}_\text{stop}.y) \\
\text{valHeight} &= \text{Math.abs(Top}_\text{stop}.y - \text{Val}_\text{height}.y) \\
\text{Width} &= \text{Math.abs(Top}_\text{stop}.x - \text{Right}_\text{stop}.x) \\
\text{valWidth} &= \text{Math.abs(Val}_\text{height}.x - \text{Val}_\text{width}.x)
\end{align*}

In cases where a significant discrepancy is found between height and valHeight, the smaller one of the two is chosen. This is in order to avoid sections which are abnormally large on one side, but might cause the section to be smaller than it should be, see "WC" section in figure 2. The same principle is applied to width as well.

---

Measuring color difference

When traversing the pixel we are looking for a color change to detect a stop mark. This color change is measured using an algorithm called CIE76.

\[ \Delta E_{ab}^* = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2} \]

The CIE76 albeit not as precise indication of the human eyes color perception compared to its successor CIE2000, it is however faster in its determination [9]. This formula is executed for every pixel in every pixel traverse step, see Figure 3 - 8. Since the calculation will be run for every pixel until a condition has been met in 6 directions performance is of the essence, thus CIE76 was in our case optimal. Result from CIE76 calculation is in range 0-100, larger value indicating larger difference. For the initial steps the threshold for detecting a stop color is a difference above 60, while in the later steps, Figure 4 - 8 the formula continues unless color similarity is below 15 compared with any of the stop colors.

Handling section overlap

In order to minimize the incidents where a sections border is not properly discovered in previous steps, which in severe cases could result in section mistakenly covering the entire map. We implemented a backup step in the algorithm, which improved results in cases where some rooms lacked border separation between labels, as a result of map design. This step is based on the premise that sections should never overlap one another similar to the research conducted by Dodge [5]. In order to prevent such cases we implemented a function which after sectioning the image scans every section for overlapping. If an overlap is detected, the algorithm changes the border locations of both sections to be half of the length between them in the respective direction.

As seen in Figure 9 used during testing the stop line between the two sections is drawn in the middle between these two. The darker areas of the images indicate the existence of a section in our system, and our test images themselves contain no borders between the text, only a black outline around the area as a whole. The direction in which the separation between the sections is made depends on the size of the overlapping area. If the overlap is greater in X respective Y direction, the restriction also occurs in that direction.
Similar in the X direction, Figure 10, the algorithm checks each sections compared to the other sections and determines if and where they should be restricted to.

Algorithm metrics

When comparing the performance of the algorithm we chose to compare it with the gold standard result of manual sectioning.

Correctness, Quality, Completeness

We used three calculations to compare the results of the algorithm with the manual sectioning which in these metrics we define as the gold standard. These three metrics are calculations based on three classifications of the results:

- **True Positive (TP)** - Found a section it should have found
- **False Positive (FP)** - Found a section that it should not have found
- **False Negative (FN)** - Did not find when it should have found

Based on these three classifications will use three calculations to measure the results of the algorithm. These are identical to those used by K. Borna et al. in their research when comparing different models. [2]

\[
\text{Correctness} = \frac{\text{True positive}}{\text{True Positive} + \text{False Positive}}
\]

Out of all the sections we found, how many in total were meant to be found.

\[
\text{Completeness} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}
\]

Of all the existing sections in the gold standard how many did we find.

\[
\text{Quality} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative} + \text{False Positive}}
\]

How well our system performed, by considering all the flaws in the amount of discovered sections.

**Jaccard Index and name accuracy**

In order to properly determine the performance of our algorithm in terms section attribute quality, we implemented an identical metric to that of S. Rice et al. [8] used to measure the performance of OCR engines. Specifically we aim to employ the name accuracy and measure how accurate our algorithm is in terms of section name.

\[
\text{Name accuracy} = \frac{n - \text{#errors}}{n}
\]

Showed how accurate the OCR scan was in terms of text retrieved for a section, where \(n\) = amount of characters in name. Higher percentage indicates closer similarity to the actual name

\[
\text{Jaccard index} = \frac{\text{Overlap area}}{\text{Union area}}
\]

Showed how large of an overlap the automated algorithm resulted in for each section compared to the gold standard determined by us previous to the test. A larger percentage indicates a closer size and placement to the gold standard. The metric is inspired by [5] but formal definition of the metric was retrieved from [8].

The metrics presented above will be calculated for every section which the algorithm discovered, after which an average number then be presented for the discovered sections to compute the respective metrics for each map.

**Time taken**

We measured time taken as the time it took for the algorithm to discover and then create the respective sections. The measure was conducted using a timer inside the angular web page, specifically the function performance.now().

8http://ase.tufts.edu/chemistry/walt/sepa/Activities/jaccardPractice.pdf
9https://developers.google.com/web/updates/2012/08/When-milliseconds-are-not-enough-performance-now
started when the image finished uploading and the OCR started scanning and concluded when the algorithm had created the discovered sections.

RESULT

Correctness, Quality and Completeness

Our algorithm when tested on three different maps, produced the summed up classifications seen in Figure 11. From a total of 34 sections in the gold standard the algorithm correctly found 26 sections (True positive). From these 34 it missed 8 (False negative). It also created 2 sections that were not on the map (False positive).

Figure 11. Classifications produced by algorithm and map resolution

<table>
<thead>
<tr>
<th>Gold</th>
<th>True Pos</th>
<th>False Neg</th>
<th>False Pos</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map 1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2400 x 1592</td>
</tr>
<tr>
<td>Map 2</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1600 x 1230</td>
</tr>
<tr>
<td>Map 3</td>
<td>17</td>
<td>5</td>
<td>1</td>
<td>1840 x 1760</td>
</tr>
</tbody>
</table>

As seen in Figure 12 the correctness, quality and completeness is consistent across multiple maps with different sections. The algorithm when run on map 1 got no error and achieved a perfect score. On map 2 and 3 the correctness continued to be high while the quality and completeness were significantly lower compared to the perfect score of map 1.

Figure 12. Correctness Quality and Completeness results from the different maps.

Time taken

In terms of time taken the algorithm contains only a slight increase for each section discovered on the image, noted that map 1 contains 10 section, map 2 contains 7 sections, map 3 contains 17. The time taken does not show a correlation between amount of sections and time taken. However a correlation is shown between resolution and time taken, where higher resolution in this case yields slightly lower time taken.

Jaccard index and name accuracy

In Figure 14, 15, 16 all the sections for each individual map is shown along with the performance of our algorithm for that particular section. Sections not found by the algorithm are marked with a "." in the tables to indicate missing values.

As seen in Figure 14 the algorithm’s results on map 1 it achieved a wide range of varying values on both jaccard index and name accuracy. Some sections were correctly named such as the “laundry” section, while other were far from correct, such as "bed 3" correlating to "3505". The jaccard index also had a varying range on the sections in map 1 with a ceiling of 86.1% and a low point of 9.7%.

Figure 14. Jaccard index and name accuracy for map 1

<table>
<thead>
<tr>
<th>Map 1</th>
<th>JI</th>
<th>Found Name</th>
<th>Name Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BED 2</td>
<td>0.402</td>
<td>BED2</td>
<td>0.8</td>
</tr>
<tr>
<td>BED 3</td>
<td>0.519</td>
<td>3503</td>
<td>0</td>
</tr>
<tr>
<td>KITCHEN</td>
<td>0.182</td>
<td>MTCHEN</td>
<td>0.714</td>
</tr>
<tr>
<td>LIVING</td>
<td>0.097</td>
<td>LnnNG</td>
<td>0.5</td>
</tr>
<tr>
<td>LAUNDRY</td>
<td>0.861</td>
<td>LAUNDRY</td>
<td>1</td>
</tr>
<tr>
<td>WC</td>
<td>0.827</td>
<td>ch</td>
<td>0</td>
</tr>
<tr>
<td>BATHROOM</td>
<td>0.270</td>
<td>BATHROOM</td>
<td>1</td>
</tr>
<tr>
<td>LIN</td>
<td>0.702</td>
<td>UN</td>
<td>0.333</td>
</tr>
<tr>
<td>BED 1</td>
<td>0.460</td>
<td>BED1</td>
<td>0.8</td>
</tr>
<tr>
<td>VERANDAH</td>
<td>0.554</td>
<td>VERANDAH</td>
<td>1</td>
</tr>
</tbody>
</table>

Map 2 resulted in a range between 38.3% and 93.8%, in terms of jaccard index. A result which is superior compared to the results of map 1 but still worse than the range of map 3. The same can be seen in the average JJ for the two maps where map 2 had a 15% higher average in terms of JJ compared to map 1 in Figure 15.

In terms of name accuracy map 2 received an average score of 86.5%, a value which is 25% higher than that of map 1, with the results in the range 46% and 100%. There is however also a notable section amount difference, map 1 consisted of 10 created sections whereas map 2 only found 4.

Figure 13. Time taken in seconds for the different maps

![Figure 13. Time taken in seconds for the different maps](image-url)
In terms of time taken, it might be possible for a manual sectioning to outperform the algorithm in terms of time taken, specifically in cases where the map only contains a single section. This possibility is also the reason why the algorithm is rather consistent with time taken, since no matter the section amount it still requires to perform OCR initialization along with complete image scan for text. How much text the OCR discovers is rather irrelevant in this metric, since OCR is the most time consuming part of the algorithm and less text will not decrease time taken since the scan still has to complete.

Worth noting however in the case of this metric, the algorithm’s time taken is not guaranteed to be correct in terms of complete sectioning of the map. In real world cases where the algorithm is used, the user would still spend time either confirming the findings of the algorithm and maybe even correcting several mistakes. This means that the time taken for the algorithm in a real scenario would not be the final time for the complete sectioning. Instead it gives a perspective to the potential of such an algorithm in terms of time saved, and why such a solution should be considered.

**Algorithm metrics**

True positive, false positive, false negative were all metrics for how well the OCR managed to find sections on our maps. As such discussing them means that indirectly we are discussing the performance of the OCR engine, in terms of text found.

When configuring our OCR scan we came up the blacklist as a mean to limit the amount of wrongly named sections appearing. An example of this is that we found a section named ",,," which in reality was a wall. This blacklist might have played a big part in why the algorithm only found two sections it should not have found(false positive), since many other nonsense sections where not included.

Due to the nature of our calculations the quality metric will always be lower compared to correctness or completeness. This relation exists due to which values are included in the calculation. Quality takes into account false negative and false
positive when calculated, while correctness and completeness uses false positive and false negative respectively.

Name accuracy is the metric we had least control over in terms of what the algorithm actually named the sections. We did not have full access to all the configurations to Tesseract-OCR and could not fine tune the OCR to give us results we were fully contempt with. Even so our average name accuracy shows promise since in most of the cases the algorithm’s names were fairly close to the gold standard.

In the aspect of measuring the jaccard index, the results may be skewed in favour of faulty size. Due to the nature of the section even by human standards the size may vary up to a size of a few pixels, but visually still perfectly align with the image sections. The images we are working with are basic images, they contain no extra data on pixel values where sections are located, and we have to set the gold standard ourselves. Such inaccuracy may prove to favour a less than full score since even a slight variation will render full score impossible.

**Method**

In the following sections we will discuss possible weaknesses and improvements of our methods.

**OCR**

The algorithm uses TesseractJS which is a wrapper for Tesseract-OCR as a starting point, which means that in cases where OCR fails to identify text labels the algorithm will assume that the location contains no section and not proceed further calculations. Something which is much more present on some maps, while not as prominent on others. Our algorithm seems inconsistent in terms of measuring its results since it heavily depends on the design, quality and font choices of the image. Although Tesseract-OCR engines provide many optional parameters to tailor the image scanning to fit specific needs, we could not achieve complete satisfaction even in what we deemed good circumstances. This might be related to Tesseract-OCR itself, or be a side effect of using the wrapper TesseractJS, but might also be the case of our lack of understanding of the Tesseract engine.

In order to limit the effect of imperfect OCR scan our algorithm might have benefited by using some sort of fallback method for finding sections, even in cases where text was not found. Such a method might use a background color to find large chunks of the image which were in open spaces, and thus assume the presence of a section in that location. This would however require the algorithm to also be able to know the background color in preferable RGB, either by manual input or automated assumptions.

Another possible improvement could consist of splitting the image into segments, a method which has shown to improve the results of OCR in different studies [10] [1] 10. Such an implementation could separate the image into segments, scan each segment and then calculate the label bounding boxes in relation to the segments location in the original image.

Otherwise alleviating the issue of lacking OCR performance in terms of discovered text could be achieved by implementing our algorithm with different OCR engines. In our research there are cases where different engines produced improved results compared to that of our implementation of Tesseract-OCR. OCR in our case deemed to be the primary bottleneck in terms of results. A different implementation could greatly improve the final results of the algorithm.

**Color difference**

Another potential weakness of the algorithm is the reliance on substantial color difference to identify section borders. Although the threshold number on what our system considers a significant color change can easily be tweaked, finding a general and perfect threshold for all the maps is near impossible. Due to different color schemes and section borders, the required value could vary. This might be mitigated by allowing the advanced user to input a threshold before scanning which they believe would be optimal for a certain image. This would however cause the algorithm move away from full automation and move towards semi automated due to user input. This is something which we did not want to include in the scope of the project, since we focused on creating an as automatic as possible proof of concept.

**Irregular section shapes**

Since we chose to only focus on maps containing rectangular shapes, the algorithm is not trained to detect anything else. This is both a limitation in our system as well as in our algorithm. We do not believe that adding more functionality to the system such as multiple different shapes would add anything to the algorithm as a proof of concept, and thus chose to focus on other priorities. This however should be considered in case a real world implementation is desired.

Furthermore there are cases where the section is not contained by borders on all sides, this might cause the algorithm to wander off into other sections until it finds their stop color. We tried alleviating this issue by adding another pixel traverse which validates the height and width. If we detect a discrepancy we decided to chose the smaller of these two. This in order to avoid creating sections which accidentally cover large portions of the map. However this creates the side effect that in some cases the sections are smaller than they should be in the eyes of a human.

**Optimizing the image using preprocessing**

Large part of optimizing OCR is the art of preprocessing the image to improve the OCR scan. In our cases this consisted of scaling the image to a much larger size, which showed a clear improvement in OCR scan results. This action was however performed manually by us before uploading, optimally this up scaling should, along with many other image optimizations, occur after uploading and automatically by the system. There are many other forms of preprocessing which can be beneficial in these cases, such as threshold the image into a binary image or clean up the noise 11. These are methods which we did not focus on applying to our system, mostly due to our priorities being elsewhere.

10https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality

11https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality
CONCLUSION
In terms of the metrics for measuring the performance of the algorithm the results varied. In a total of 34 sections in the gold standard the algorithm found 26, along with 2 sections which were not actual sections. The metrics employed for these classifications received perfect score on map 1, while varying on map 2 and map 3. The largest difference in terms of average jaccard index between the highest map 3 and the lowest map 1 with 85.6% and 48.8%, while name accuracy difference between the same maps were 91.7% and 61.5% respectively. As these two metrics are based around the section attributes, they show how good the algorithm is at finding correct names, size and location for maps with different designs.

TesseractJS with our configuration was during our testing the largest bottleneck in terms of achieving the best sectioned results. Mostly because the other steps of algorithm expand on what the OCR scan initially revealed, and imperfect scan results in missing sections. In cases where the label is not discovered by OCR, the algorithm does not attempt to find borders for this non existing label, which results in a not found section (false negative).

The pixel traverse part of the algorithm accompanied by the CIE76 color difference formula appeared to perform adequately on most maps. However in some cases the traversing would bleed out of the desired section due to lacking sufficient border differentiation and resulting in worse jaccard index results.

The time which a user would save in the sectioning step would be significant, in similar systems but can vary greatly depending on aspects of the map such as text readability, color difference and image resolution. All of which are values which impact the performance and results of the algorithm.

Future work
Our algorithm is not perfect in terms of our performance metrics, and would benefit from further tweaks and development. Future research could focus on improving certain aspects of the algorithm such as:

• Different OCR-engines and compare the result between the different engines. To see how much our algorithm is being limited by OCR.

• Backup method for discovering sections on a map, as a fail safe to OCR. To see if there is another or supplemental method which improve results in terms of sections correctly found.

• Image preprocessing is something which did not fit in the scope of our project. In future work image preprocessing such as segmentation, images threshold or other techniques could prove to be an essential part in achieving a greater result in terms of OCR scan.

REFERENCES