Converting Map Image To Graph Representation For
A Map Helper App

Manal Hamdi

Supervised by Dr. Naeem Nisar Sheikh

Spring 2019

SCHOOL OF SCIENCE & ENGINEERING – AL AKHAWAYN UNIVERSITY
Capstone Report

Student Statement
I have applied ethics to the design process and in the selection of the final proposed design. And that, the designer has held the safety of the public to be paramount and has addressed this in the presented design wherever may be applicable

MANAL HAMDI

_________________________
Manal Hamdi

Approved by the supervisor

_________________________
Dr. Naeem Nisar Sheikh
ACKNOWLEDGEMENTS

I thank my supervisor for helping me throughout this project. I also thank my family and friends who were a great support system during this semester.
Contents

List of Figures 4

List of Tables 5

1 Introduction 7

2 Methodology 9

3 Implementation 10

3.1 Detecting Roads .................................................. 10
3.2 Conversion to single line format .............................. 12
    3.2.1 Detecting Road Format and Calculating Road Width .......................... 13
    3.2.2 Dilation ..................................................... 19
3.3 Making roads 1 to 2 pixels wide – Erosion operator .......... 22

4 Going towards the Graph Representation 23

4.1 Graphs and their Representation .............................. 23
    4.1.1 Graph Representation ...................................... 23
    4.1.2 Graph Traversals .......................................... 24
4.2 Constructing the Graph ........................................... 25
4.3 Identifying Vertices .............................................. 25
4.4 Detecting Edges .................................................... 27

5 STEEPLE Analysis 29

6 Final Remarks 31

6.1 Challenges and Limitations .................................... 31
6.2 Future Work ....................................................... 31
List of Figures

1. Main steps of our implementation ........................................ 10
2. Example of RGB values for two different pixels ....................... 11
3. Example of single line road map ...................................... 12
4. Example of double line road map ...................................... 13
5. Double line road map; road width = 7 ............................... 15
6. Plot of figure 5, peak at 7 ............................................. 15
7. Double line road map - Road width = 7 ............................. 16
8. Plot of figure 7 - Peak at 8 ............................................. 17
9. Double road format .................................................... 17
10. Example of a tilted road ................................................. 19
12. After 1 iteration of dilation ........................................... 22
13. Example of a graph ..................................................... 23
14. Frame pixels ............................................................ 26
15. The detected intersections .......................................... 27
16. Intersections grouped .................................................. 28
List of Tables

1  Vertical Orientation .................................................. 20
2  Horizontal Orientation ............................................... 20
3  Example of Adjacency List Representation ....................... 24
Abstract

This project takes as input an image of a map and converts it to a graph data structure representation. For this purpose we first extract road pixels based on their color, we then make the roads 1 pixel wide using a technique that we developed as well as dilation and erosion operators, then we detect the intersections and make them the vertices of the graph and then finally detect the edges between the vertices and build the graph. This project has been implemented using Python and OpenCV. The above steps are mostly following the methodology outlined by Chiang and Knoblock in their two papers on this subject from 2008 and 2009, with one step having a variation that is our contribution.
1 Introduction

Image processing is an important field within computer science as images are an essential source of information. Many tasks seem trivial for humans like reading handwritten text in an image or recognizing that a shape corresponds to a human or not, while those tasks end up being complicated for computers. Thanks to image processing with the combination of other fields these tasks become possible.

A subfield of image processing is map processing. Maps give rich and valuable information about the world we live in. This project is part of this field and contributes to it.

This project was a suggestion from my supervisor Dr N. N. Sheikh. He came up with the idea of a map helper application which consists of taking as input a map for a specific location and then the map would assist the users in planning their route beforehand. The app would also be helpful for people with vision impairment (including blindness) as the app could guide them given a map of the facility they are in. This app could take a photo of a guide board that has image of the map of a multi-building facility, such as a college/university campus, a zoo, etc, and then converts it from an image of pixels to a graph representation (vertices and edges), and then in response to voice queries from the user, it would do planning/navigation over this map, and respond with voice instructions/assistance that guides them from location A to location B. Of course such a project requires a time frame that exceeds a one-semester period. Thus, we thought about which portion of this project would be intellectually most challenging, given the current knowledge. Of course, voice recognition and production are challenging tasks, but in our literature survey we didn’t find enough attention to the problem of conversion of the map image to graph representation. So, we decided to focus on this task for the semester.

The software will essentially take an image (jpg, jpeg or png) as input, detect the roads and intersections, and then uses this information to build a graph. Once the graph is ready, various operations will be possible to perform like the shortest path from a place to another. This is an important feature to implement because it will allow users to navigate in places unfamiliar to them and plan their routes beforehand.
This project also served as a learning process for me to understand the basics of the field of image processing. Through the course of our work, we have built upon techniques identified in literature, as well as contributed our own techniques, which we hope can be of use to the broader community of researchers working in image processing, esp. in map processing.
2 Methodology

Before implementing our project, we decided to first check the existing literature in order to review the available techniques that can be used for our solution. Most of the work found was done in the 1990s and it relates to the process of digitizing satellite maps and not raster maps which is the focus of our project. We based our work on both [2], and [3] mainly as these were the only papers we found that deal with raster map input images. This paper suggest steps to vectorize raster maps. It points to other papers that it used in some of its steps. I chose this paper as a starting point that will guide my project. I used some techniques from the paper with some variation, modified others and came up with new ones. So to sum up the methodology followed in this capstone is starting from literature, designing techniques either as variation of what is in the literature or new ones, then proceed to implementation and finally testing. I tested the code on a couple of maps and tried to tune the algorithms according to the results.

For the implementation we used Python and OpenCv [1] as the latter presents powerful tools in image processing. Briefly, OpenCv represents images as $width \times height \times 3$ arrays where the 3 values in the third dimensions are the red, green and blue values of each pixel. More details are provided in the next section.
3 Implementation

This project is structured as follows. The input is an image of a map. The first step is to detect the road pixels and differentiate them from the other objects. We use color as our identifier and convert then the input image to a binary image where foreground pixels represent roads and the rest is background. Then we want to reduce the width of the roads as much as possible, i.e. to a width of 1 or 2 pixels only. We go through a 3 step process in order to achieve it. Once we have the 1-2 pixels width roads we detect the vertices and the edges linking them and use them to build our graph. We used the 2 papers mentioned above as well as made our own contribution to the techniques for reasons that are mentioned in the relevant parts.

![Diagram showing the main steps of the implementation](image)

Figure 1: Main steps of our implementation

3.1 Detecting Roads

The input of this stage is a map image given by the user and the output is a binary image where 0s represent background and 1s represent ideally only road pixels.

An image is composed of a number of pixels. Each pixel has a vector of values that encodes its color. In my project we use the RGB system which refers to red, green and blue. In other words a pixel has 3 values, each one referring to the “amount” of red, green and
blue (see figure 2).

In a map image there are many objects such as roads, text, legends, and symbols. We are interested only in the roads. Thus an intuitive solution to detect the roads is to know the color of the roads (i.e. the value of the pixels that represent roads) and then use it in order to select those pixels. One way to know the color of the roads is to ask the user to point to a road and we get the value of it. But as pointed by [2], an image contains more colors than we think it does. Although for a naked eye we might see that an object is red, if we zoom in we can find that each pixel is a different shade of red for example. Thus even a small image with seemingly few colors could have hundreds of colors. For this reason Yao-Yi Chiang and Craig A. Knoblock suggest to reduce the number of colors in the map using K mean and Mean shift methods. More details can be found in [2]. For my capstone we have opted for the solution where the user will be asked to point to any pixel within a road, so that we can detect the color of the roads in this picture. Since our current project is not implementing any user interface, and is focusing on the mathematical and algorithmic details
of these operations, we assume that we have received this value and then we proceed.

Then we mark the pixels whose color value is within a range of this color (i.e. this color plus or minus a threshold). The pixels that satisfy this condition are made white, and the rest are made black. This method detects all the roads, but it also picks up other elements of the picture, such as building names, etc. (as long as their color value is in that range).

### 3.2 Conversion to single line format

According to [3], maps can be classified into 2 categories according to the way roads are illustrated: maps in which the roads are drawn as single lines and maps in which roads are drawn as double lines (see figures 3, and 4).

![Figure 3: Example of single line road map](image)

In order to be able to interpret the images and build a graph from them we need to unify them into the same format so that we can design a generic solution that would work regardless of the format. In the paper [3], it is suggested to convert all the maps to single line
roads format and then reduce the roads width to 1 pixel. This unification has 2 subtasks: converting to single line road format and making the road width 1 pixel.

3.2.1 Detecting Road Format and Calculating Road Width

In this step the input is a binary image where road pixels are represented as foreground pixels and the rest as background pixels and the roads are either single or double line format. The aimed output is a single line format road map.

First we should detect the format of the map. In the paper [2] they use a technique that they call “parallel pattern matching”. Our initial intention was to use their algorithm but because of some limitations we decided to come up with our own. Below I present their method, the limitations we encountered, and the process we went through.
a. Chiang-Knoblock approach: Parallel Pattern Matching

The goal of this algorithm is to find the road format and road width. Although road width is an identifiable parameter for both single line roads and double line roads, we will need the road width only in the case of double line roads. This information is critical as we will need it in next stages.

They say that a pixel is considered to be on a double road if and only if we can find at least 1 foreground pixel on its left or right at a distance of road width and at least one foreground pixel up or down at a distance of road width. For sake of brevity, if a pixel satisfies this property, we say it has the “double property.”

To do that we need to know the road width. As this is not an information that we have at this stage we try road widths from 1 to M (parameter to be chosen by the programmer). To know which value within the range is the correct one they count the number of pixels that have double property and divide it over the initial number of foreground pixels for each value. The width value (greater than 1) that gives the highest ratio is the right road width. The time complexity of the algorithm will be O(M * l * w) where l is the length of the image and w is the width. Once we run the algorithm we know that it is single line road map if the trend in the resulting ratios is decreasing overall. If there is a peak within the trend of ratios then it means that the map is double line and the peak is the road width. This is based on the idea that if the map has double line roads when we get to the correct road width most of foreground pixels should be detected as being part of double line roads and therefore the ratio should be very high.

After implementing this technique we found that in all the tests the road format is detected correctly but not always the road width. There are some limitations to this technique. First the choice of M is a critical step as M can vary widely from low to high resolution maps. High resolution maps can have road widths of 40 pixels (or even more) for example while low resolution maps can have roads widths of 7 pixels. In the paper they suggest to use M = 10 but this will miss the right road width for many high resolution maps. One might be tempted to think that choosing a high value for M is the solution. Although this might
work but the time complexity of algorithm will increase dramatically which would make this solution a bottle neck for the overall software.

Another limitation of the method is that although they claim that we should take the
road width that gives the peak ratio, in our testing we found that following this does not always give an accurate result.

![Figure 7: Double line road map - Road width = 7](image)

For the figure 5 we see that the technique works well as it detects the road format. In double format roads the peak does not happen exactly at the correct road width. There is an explanation to this part; in the 2 examples we have the width of a line is 2. 11’ 000 1”1”. The road width detected by the algorithm (that gave the highest ratio) is the one that starts from the extremities. In the above example the road width is supposed to be 4 but 5 would give a higher ratio. So the reason behind not detecting the road width is that the line width is higher than 1.

As we see in those example the road width is not represented by the ultimate peak but by the peak in a small region. A solution that we saw plausible is given a map we estimate a potential road width and run the algorithm on the range [estimated road width - small delta, estimated road width + small delta]. This will solve the issue of time complexity and also we will be able to detect the peak within this range. To estimate the road width we suggest...
that the user selects a window from the image that will contain a segment of a road only.

b. Our approach

This technique takes as input a portion from the map that selects a segment from the road only and outputs the road width, line width, and road format. An intuitive way to detect the road format is get a row or a column from the binary version of the window selected by the user, and classify the pattern we detect from the changes of colors. If we have a sequence of [black, white, black] then we know it is a single line road otherwise if it is [black, white, black, white, black] then it is double line road.

For example in figure 9, we can see that if we take the row on the right, we have the following pattern [black, white, black, white, black], which represents a double road. In general, we need to know how we can get a sample that will give us the appropriate pattern that represents the format of our road. First we need to know the orientation: hor-

![Figure 8: Plot of figure 7 - Peak at 8](image)

![Figure 9: Double road format](image)
izontal or vertical. In other words do we take a row or a column? To do this, for each orientation I define 5 or 6 strokes (rows if we are working horizontally or columns otherwise) in the image, and for each stroke I count the number of alterations \(^1\). The strokes are equally distanced from each other.

After we finish all the strokes we get the max number of alterations in each orientation. we keep the orientation that has the max number of alterations. At this point we know the orientation, and the maximum number of alterations. If the number of alterations is bigger than 2, it means that the road is double. We go through the index of the stroke that gave us the max number of alterations, and detect the road width and line width according using the following regular expression patterns:

\[
\begin{align*}
0^*1^*0^*, & \quad \text{For single roads} \\
0^*1^*0^*1^*0^*, & \quad \text{For double roads}
\end{align*}
\]

The road width is the number of 1s (white pixels) in the first case, and the number of the middle 0s (black pixels) in the second one. When the maximum is equal in both orientations we chose by default horizontal orientation (arbitrary choice, no specific reason behind it). Because we made the assumption that the user will select a window where the road is correct (if it is double they will select the way it is, etc.) and also centered we agreed that we can just use the window to extract the information we need and not do the Parallel Pattern matching algorithm on the whole image.

Now that we have decided to use only the user-selected window and not the PPM on the whole picture we can do more work in order to have a more accurate road width. One of the modifications suggested by Dr Sheikh is instead of choosing the maximum number of alterations as a criteria for choosing the orientation it would be better to use the average of number of alterations that we get from each orientation and chose the orientation that gives the highest average. This is specially useful for slightly tilted horizontal and vertical lines (see figure 10).

\(^1\) An alteration is the change in the color of pixels either 0 to 1 or 1 to 0
Tables 1 and 2 show a step by step of how the algorithm work for figure 10. For vertical orientation, we get an average of 1.75 and horizontally we get 3 so we choose the horizontal orientation for our analysis. The stroke that gave us the max number of alterations are all the rows so we chose by default the first one. The number of alterations is so the format of the road is double line. We detect the pattern 0*1*0*1*0*. We count the number of middle 0s and we find 3, and we find one 1. So the road width is 3 and the line width 1. If we have chosen vertical orientation we would have chosen Col 3 and we would have detected a road width of 6 pixels and line width of 2 pixels.

3.2.2 Dilation

Dilation is a known morphological operation used on binary images. The goal of dilation is to enlarge foreground regions which leads to enlarging the boundaries and filling the holes. This
<table>
<thead>
<tr>
<th>Columns</th>
<th>Number of Alterations</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col 1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Col 2</td>
<td>2</td>
<td>1.75</td>
</tr>
<tr>
<td>Col 3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Col 4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Vertical Orientation

<table>
<thead>
<tr>
<th>Rows</th>
<th>Number of Alterations</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Row 2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Row 3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Row 4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Horizontal Orientation

A method is also suggested in the paper [2] to fix the holes caused by the noise, and convert double line roads to single line roads. Dilation algorithm shrinks the gaps in iterations. In a single iteration we carry the following algorithm: if “a background pixel has any foreground pixel in its eight adjacent pixels, it is filled up as a foreground pixel” [2]. How many iterations do we need in order to convert a double line road to a single line road? The answer is \(\lceil \frac{\text{road width}}{2} \rceil\) iterations of dilation. If the road is single the paper suggests doing 3 iterations of dilation which fixes gaps of maximum 3 pixels. At this stage, line width means the new width of the road so we should update line width with \(\text{left line width} + \text{road width} + \text{right line width} + 2 \times \text{num dilation iter}\) if the road was initially double otherwise we update it with \(L + 2 \times \text{num dilation iter}\).

To illustrate better this operation here is an example. The orange pixels are the ones that are background pixels and have foreground pixels as neighbours so they should become foreground, too. From an implementation perspective we know that in most cases we will have more background pixels than foreground therefore instead of looking at the background pixels and checking it neighbours we can look at the foreground pixels and make its background neighbours foreground.
(a) Before dilation starts
(b) Orange pixels will change
(c) After 1 iteration of dilation
(d) After 2 iterations of dilation
3.3 Making roads 1 to 2 pixels wide – Erosion operator

Erosion is also a basic morphological operation, we will use it in order to make the roads 1-2 pixel width. For every foreground pixel if it has a background in its neighbours it becomes background too. As we already know the line width then we can know the number of iterations to be done. If the line width, \(L\), is odd the number of iterations of erosion is \(L/2\). We cannot use this formula if the line width is even because it will delete the road completely, so instead we use \(L/2 - 1\). This leaves us in the first case with 1 pixel roads and in the latter case with 2 pixels width roads.

![Figure 12: After 1 iteration of dilation](image-url)
4 Going towards the Graph Representation

We chose to represent our map images as a graph data structure as it is a powerful data structure that will allow us to perform multiple operations on the roads and is the most suitable one for this type of data.

4.1 Graphs and their Representation

A graph, in computer science, is a data structure for representing relationship between object. Graphs are composed of vertices (the objects), and the relationship between those vertices is represented using edges. Figure 13 gives an example of a graph. Graphs are good for representing network of roads. This can be done by choosing important places of a map as vertices (e.g, cities), and the roads between them as edges.

![Figure 13: Example of a graph](image)

4.1.1 Graph Representation

There are different ways for representing a graph inside memory. The important ones are:

- **Adjacency matrix**: The graph is represented as a two dimensional matrix. Rows in the matrix are source vertexes, and columns are destination vertexes. A direct edge between \(u\), and \(v\) is represented as 1 in \(u\)’s row and \(v\)’s column.
<table>
<thead>
<tr>
<th>Vertices</th>
<th>Adjacency list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[2, 5]</td>
</tr>
<tr>
<td>2</td>
<td>[3, 1, 5]</td>
</tr>
<tr>
<td>3</td>
<td>[2, 4]</td>
</tr>
<tr>
<td>4</td>
<td>[5, 3, 6]</td>
</tr>
<tr>
<td>5</td>
<td>[1, 2, 4]</td>
</tr>
<tr>
<td>6</td>
<td>[4]</td>
</tr>
</tbody>
</table>

Table 3: Example of Adjacency List Representation

- **Adjacency list**: The graph is represented as a list of source vertices. If $v$ is $u$’s list, that means that there is a direct edge between $u$ and $v$.

- **Incidence matrix**: A two dimensional matrix, where rows are vertices, and columns are edges.

In our implementation we used an adjacency list, because it is memory efficient (linear function of number of edges), and is best for graph traversals. Table 3 represents the graph in Figure 13 as an adjacency list.

4.1.2 **Graph Traversals**

A useful operation on graphs is graph traversal. There are two main traversal algorithms for graphs:

- **Depth First Search (DFS)**

- **Breadth First Search (BFS)**

There are distinctions between the two algorithms in how the traversal is done. For example, BFS visits vertexes using a first in first out (FIFO) strategy, while DFS uses a last in first out strategy (LIFO). BFS results in computing the shortest path in unweighted graphs, but this distinctions are not relevant in our project as our aim is to test for connectivity only. We used a DFS implementation because it is the easiest to code. Algorithm 2 is what we used for traversal.
4.2 Constructing the Graph

All the previous steps were a preparation to make the graph construction possible and easier. After the Erosion step, we have a binary image of 1 pixel roads. To construct a graph we have to choose our vertices and edges, we have settled down with what [3] did, we have chosen road intersections as the vertices, and the direct path between two intersections as edges. We use the following algorithm to create the graph:

Algorithm 1 Construct Graph
1: procedure CONSTRUCTGRAPH(imageMap)
2: intersections ← GetIntersections(imageMap)
3: let $G$ be an empty graph
4: $G$.vertices ← intersections
5: for $u$ in intersections do
6:   for $v$ in intersections do
7:     if $u \neq v$ & AreConnected($u,v$) then
8:       add edge $(u,v)$ to $G$
9:     end if
10:   end for
11: end for
12: return $G$
13: end procedure

The sections below give more details how we calculate the intersections, and how we find direct edges between them.

4.3 Identifying Vertices

We chose intersections as our vertices. For every foreground pixel we draw a frame of 1 pixel width around it, and see how many intersections there are with the frame (see figure 14). If we have exactly 2 intersection points, then we carry out the following check: If we denote the pixel under test as $O$, and the intersection points as $A$, and $B$ and if the slope $OA$ is different that the slope $OB$, then $O$ is an intersection, otherwise it is not.
If we let $\alpha$ be the number of intersections, then we have the following cases:

\[
\text{IsIntersection(pixel)} = \begin{cases} 
\text{true}, & \alpha = 1 \\
\text{true}, & \alpha = 2 \ & \text{slope}(OA) \neq \text{slope}(OB) \\
\text{true}, & \alpha > 2 \\
\text{false}, & \text{otherwise}
\end{cases}
\]

When one pixel is flagged as an intersection, usually adjacent ones will be considered as intersections, too (see figure 15). We cluster all adjacent intersections pixels as one component. We do this by doing a union find. Every component is given a different id, which we reuse as node ids.
4.4 Detecting Edges

An intersection is connected to its closest intersections by either vertical, horizontal, or diagonal line roads. The line roads maybe noisy, and not be perfectly straight, but the noise is negligible, and our algorithm account for that.

To get the graph edges, we start a depth first search from one of the intersection components, and follow pixel roads until we find another intersection. Once an intersection is found, we mark the intersection we started DFS from, and the one we reached as connected. We do the same for all other intersections.
Algorithm 2 DFS

1: procedure DFS(intersection)
2:     seen(intersection) ← true
3:     for ∀pixel ∈ Adjacent(intersection) do
4:         if seen(pixel) then
5:             continue
6:         end if
7:         if SameComponent(intersection, pixel) then
8:             DFS(pixel)
9:         else
10:             Follow road pixels until a new intersection β is found or end of road is reached.
11:             if β exists then
12:                 AreConnected(intersection, β) ← true
13:                 DFS(β)
14:             end if
15:         end if
16:     end for
17: end procedure
5 STEEPLE Analysis

The work presented here has primarily an ethical, societal, and technological implications. The tool proposed under the hood uses feature extraction from images in order to capture a map from an image. This certainly raises several ethical issues.

Ethical

Mining information from images can be used in multiple ways, certainly it gives the opportunity to solve many real life problems, but unfortunately can also be used the wrong way. The main issue is that it threatens users’ privacy. People are constantly sharing their pictures and videos so a similar technique can be used to detect their location for example, which leads to the problem of massive surveillance. All sort of information can be extracted so having the right data and techniques limits highly other people’s freedom and right of privacy. The extracted features can also be used for business purposes like targeted ads.

Technical

From a technical viewpoint, my work can be considered as a research project in computer vision. My solution tries to reproduce some algorithms already available in literature, which is important in science because it gives credibility to proposed solutions. Moreover, we will try to propose new improvements to existing techniques that will hopefully help others, and hence contribute to public knowledge.

Social

Last, the project specification focuses mainly on being useful to our society. This consist in giving users the possibility to navigate in places unfamiliar to them. The software is also very valuable for indoor maps, and can turn to be useful for finding escape routes in emergencies. For example in case of a fire the software can allow you to find the shortest path to the nearest exit in a small amount of time. It will also allow users with special needs to plan their route in advance. It is also a valuable asset for zoos, national parks, and similar places. Instead of designing their own solution for navigation, this software can be used by users for scanning the map of the place, which can be put at the entry. This of course will reduce the costs of those entities. Google maps and similar applications are limited to certain
options like giving the shortest path from a location to another. This software will offer the possibility of running different algorithms like planning tours when visiting cities.

**Economic**
This project does not raise or solve any economical issue in a direct way.

**Environmental**
The deployment of this capstone does not require the use of energy in a large scale and does not lead to the misuse of natural resources in any way.

**Political and Legal**
This project does not deal with politics and does not interfere with legal policies and regulations.
6 Final Remarks

6.1 Challenges and Limitations

We faced several challenges during our implementation of the project. We first had to fa-
miliarize ourselves with new topics, and find relevant research literature. Our project was
inspired from both [3] and [2], we used some of the techniques cited in the papers, but some-
times they didn’t always work as intended. So, it was a trial and error until we found what
best works for us. We had to modify some of the ideas sometimes, too. This required lot of
testing, and that usually takes a lot of time. We also were short of test data, we tested our
algorithms on existing maps, but we also had to generate our inputs for an exhaustive test.

6.2 Future Work

Extracting a graph from an image is a huge project. A semester is barely enough to do all the
research, and write a working prototype. Being ready for production will need more time and
more research. We proved that’s it’s possible, but lot of things should still be investigated,
and made better. More work has to be done to make the process less user dependent. We
also need more research to tune some of the internal parameters like the frame size used for
finding intersections, and the number of iterations during dilation and erosion. A mobile
application should also be created so that users can use this feature on the go. The mobile
app would ideally support voice assistance.
References

