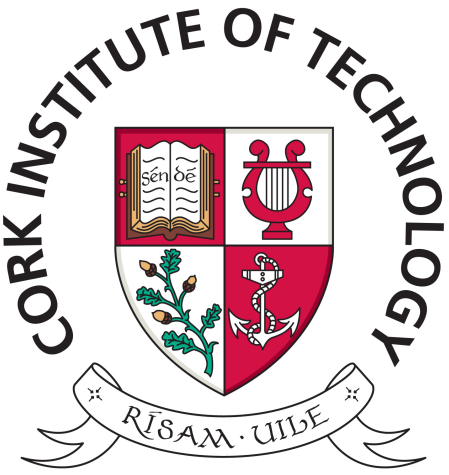


Automated three-dimensional translation of floor plans

by

Vilius Valiusis

This thesis has been submitted in partial fulfillment for the degree of Bachelor of Science in Software Development

in the
Faculty of Engineering and Science
Department of Computer Science

June 2018

Declaration of Authorship

I, Vilius Valiusis , declare that this thesis titled, Automated three-dimensional translation of floor plans and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for an undergraduate degree at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

CORK INSTITUTE OF TECHNOLOGY

Abstract

Faculty of Engineering and Science
Department of Computer Science

Bachelor of Science

by Vilius Valiusis

The thesis presents an automated system for analyzing and reconstructing floor maps as three dimensional model. To achieve this, floor plans will be deconstructed and reconstructed in a sequential approach. The system is broken down into five parts: information segmentation, structural analysis, semantic analysis, architectural decomposition and reconstruction. This process allow for a seamless transition, going from an image of a floor plan; to an interactable three dimensional representation of it.

...

Acknowledgements

To Dr. John Creagh and Mr. Arthur Tobin for acting as the project supervisors

Contents

List of Figures

List of Tables

Abbreviations

OCR	**O**ptical **C**haracter **R**ecognision
RGB	**R**ed **G**reen **B**lue
API	**A**pplication **P**rogramming **I**nterface

For/Dedicated to/To my...

Chapter 1

Introduction

1.1 Motivation

Today, we are busiest we've ever been. This has led us to many new innovations across various fields, just for sake of convenience so we could save few minutes. However, some fields have remained rather stagnant despite many attempts. One of such fields is house/apartment marketing.

When a house or an apartment is advertised, it is normally done through a few photographs and a description. This is often lacking, as everything is left to the customers imagination. Of course, many new approaches have been used to attempt to alleviate this problem, such as 2D video, 3D video, laser scanned models and hand made models. These solution often are expensive, time consuming or simply do not provide enough value.

The aim of this project is to try to introduce a possible alternative solution to this problem, one that would require no external data or user input and could be easily automated.

1.2 Executive Summary

The project aims to introduce a two layer system for generating for a 3D models. The end goal is to have an automated system, where an image of an architectural floor plan can be broken down into its constituent parts to be represented inside a web application. There the model can be viewed and interacted by the user through its interface. The aim of the project is not build a product, but build a skeleton of a system that can be further extended to create a product.

The first part of the system is the systematic break down of the architectural floor plan. There, the extracted information will be used to build a structured file that can be further used in the web application to compile the 3D model.

This will be achieved through a manually run python application that will take floor map(s) as its input. Through this it will:

- Generate the layout of the accommodation.
- Classify features denoted on the floor map.
- Map features onto the layout.
- Classify and label rooms.
- Generate output file to be used for the JavaScript application.

The HTML embedded JavaScript application will take the generate output of the python application and produce an interactable model. Through the JavaScript application the user will be able to:

- Switch between perspectives of top down and full 3D.
- Move freely around the model in full 3D perspective.
- Move in four directions from a top down perspective.
- Zoom in/out in top down perspective.
- Create, edit and delete added feature areas in the model from the top down perspective.

1.3 Structure of This Document

- **Chapter 1**: A gentle introduction to the project subject.
- **Chapter 2**: A short review of the areas/topics research areas within the project.
- **Chapter 3**: The break down of the problem. The chapter described the objectives, requirements, assumptions and constrains pertaining to the problem.
- **Chapter 4**: This chapter covers the implementation approach to the problem, further reasoning why and how certain step are taken.

- **Chapter 5**: Describes the implementation and why features are implemented the way they are.
- **Chapter 6**: Reflects on the work done and problems faced throughout the project, while also discussing the possible feature work.

Chapter 2

Background

2.1 Thematic Area within Computer Science

The project is an automatic system that analyzes floor plans using a layered approach, producing structured and label data that can be used to build a interactable 3D model through a web application for accommodation advertisement.

Primarily the problem described in the theses is a marketing problem that is solved by the use of machine learning. What we are taking is an image of a floor map and breaking it down into its components through the use of machine learning techniques. Through this process data is gathered and reconstructed into a form-factor that can be used in the JavaScript front-end application for advertisement purposes.

The main areas that the project falls under are computer vision and software development. These areas are broad and span over multiple of core subject areas.

- **Computer Vision:**
  - **Optical Character Recognition:** Better know as OCR. Used to analyze and extract text from an image.
  - **Morphological Image Processing:** A set of image processing techniques that work on shapes rather than colors to extract meaningful information.
  - **Object Detection:** Finding and labeling know objects within an image.
- **Software Development:**
  - **Python Desktop Application:** A Python based desktop application to hold the computer vision solution; used for administrative purpose.

- **JavaScript Web Application:** A JavaScript web application to hold the output of the python application; used for customer interaction.

It must be noted that the problem that is being solved is done so in a limited capacity. The reason behind is the exclusion of a back-end service to handle transactions between essentially two front-end applications. Cloud computing could potential be used in order to mitigate the need for the desktop application.

2.2 Project Scope

The problem discussed within the these is a multifaceted one, however it can be approached sequentially. To begin with the heart of the problem is one of computer vision.

2.2.1 Computer Vision

Computer vision in its essence is divided three parts, consisting of analysis, extraction and comprehension of an image. To further understand what that entails it is paramount to understand what is an image.

For a computer to understand an image it must, it must be represented in a numeric format. The formatting an image uses is called a pixel. An image consists of many of these pixels, all represented in a grid usually called a resolution. A typical image with the the dimension of 1920x1080 has around two million pixel. Each pixel represents a color using binary number(s) that range from 2^1 to 2^3 For example 2^1 is considered a binary image with a value of either 0 or 1 representing the colors of black and white. Now to represent more color in an image than just black and white a bigger value of 2^3 or 8 binary numbers is used, allow for 255 shades of a color. However, due to many limitations we do not use any colors outside of the three primary colors of red, blue and green. Despite only having access to three colors, we can make mix them to achieve any color in between.

When you look at an image full of colors, what you see is an RGB pixel. A pixel comprised of an three values, each representing one of the primary colors. This means that each pixel can represent 255 * 255 * 255 = 16,581,375 colors.

In the field of computer vision we try to take all of this data from an image, look for patterns in all these pixels, then extract those patterns and label them automatically.

2.2.2 (OCR)Optical Character Recognition

OCR is one of the sub-fields in computer vision. The purpose of this field is to find and extract text from an image. For this process to occur it first need to understand what text looks like. Pattern matching is one of the ways it does this, it works on a macro scale looking for recognizable patterns within a letter. A pattern usually consists of certain strokes that are unique to a given letter. This type of a process is fast and works really well for computer generated text that has no variation.

2.2.3 Morphological Image Processing

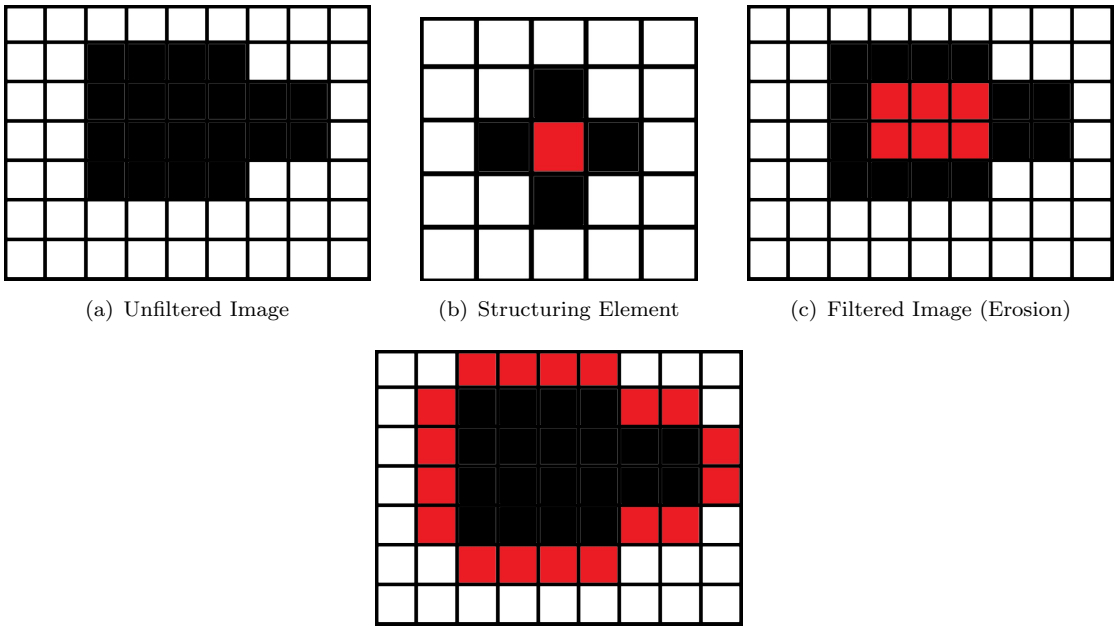
"Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image"[2]. The study of morphology only recently came into computer science, with it's roots being found in both in biology and linguistics.

Morphology in image processing is often used to manipulate an image in such a way, that it makes it easier to extract certain information from it. Because morphological algorithms only work on shapes rather than colors, we have to convert an image into a binary image. A binary image is composed of pixels that only store 0 or 1 to represent the colors of black and white. By converting an image to a binary one it allows us strictly focus on the shapes we are looking for.

As an example lets take binary erosion operation. The operation is one of the fundamentals in morphological image processing. It takes an image as an input 2.1(a) and then applies structuring element to it 2.1(a) e.g. circle of 1 pixel diameter. What the operation does, it takes every black point inside the image and tries to fit in the structuring element. If the element 2.1(b) doesn't fit into the unfiltered image 2.1(a), we subtract that points until we have a resulting image as as seen in 2.1(c). In image 2.1(c) red pixels all back pixels are removed, leaving only the red pixels.

Now if we look at a process of binary dilation which along side erosion will also be used within this project. Dilation just like erosion uses a structuring element 2.1(b), but it does the opposite. So, for any location where the structuring element fits, we add on the points of the structuring element. This effect can be seen in 2.1(d), where red block are added on and the black block are the originals.

To see this process on a larger scale we should look at the image of a floor plan2.2(a). For this image the task is to remove everything else except the walls. In order to accomplish this we first convert the image from RGB to binary. To do this we change every pixel above a certain threshold to white and everything below it to black. The final step is to apply an erosion operation with a large enough structuring element to produce 2.2(b)



(a) Unfiltered Image (b) Structuring Element (c) Filtered Image (Erosion)

(d) Filtered Image (Dilation)

FIGURE 2.1: Morphological Erosion and Dilation

(a) Unfiltered Image (b) Filtered Image

FIGURE 2.2: Morphological Erosion

2.2.4 Object Detection

Object detection is one of the computer vision fields, where the objective is to find an object of a know class, such as person, car, chair etc.. An image may contain any number of these classes in any given position inside an image. These objects may have variations within a class, such as scale, orientation, color and many more. The challenge is to recognize the objects class despite all of these differences.

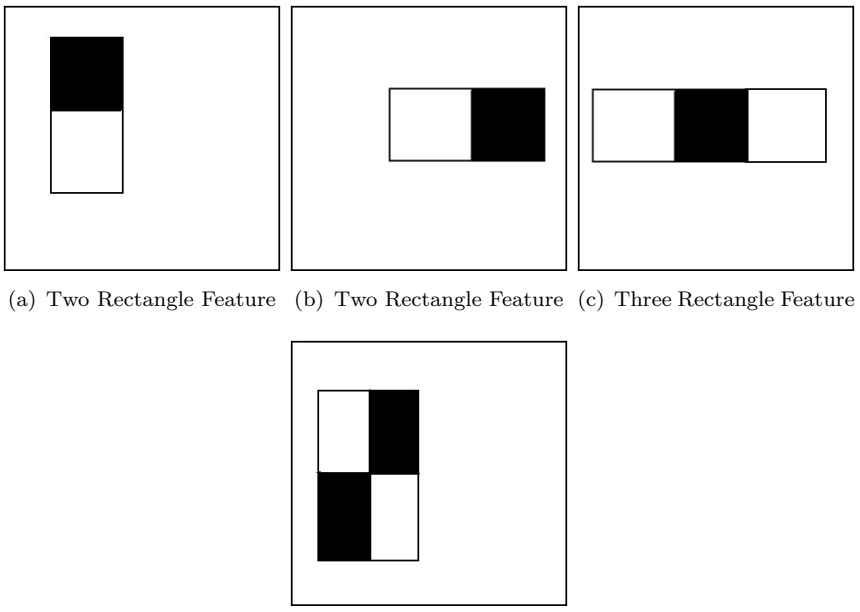
However, object detection is not just limited to classification of object but also localization. What that means is essentially finding the location of an object inside the image. The output of this process is usually four coordinates on the image.

There are many approaches for this problem, but the approach chosen for this project is the first introduced by Paul Viola in his paper named Rapid Object Detection using a Boosted Cascade of Simple Features [3]. The algorithm proposed in the paper claims to be able to achieve rapid image processing with high detection rates, up to 15 frames per second. The proposed method can be divided into three main aspects. First is the *integral image*, which allows for rapid feature detection. Second is the learning algorithm based on *Adaboost*, it allows to pick out the best feature out of a large set. Third is the *cascade of classifiers*, this method allows to focus on image regions that have most potential while discarding others.

The object detection is done through the use of features over pixels. There are many reasons behind this, with the most common one being is that features provide domain knowledge that pixel cannot with a limited set of training data. For the feature detection we will use three types of features. First is the rectangle feature seen in 2.3(a) and 2.3(b), to calculate the value of these feature we take the difference of the sums of the two rectangular regions (black and white) that have the same size and share and are horizontally or vertically adjacent. For the three rectangle feature 2.3(c) we take the sum of the two of the middle region and subtract from the sum of the regions adjacent to it. For the four rectangle feature2.3(d) we take the sums of the opposite diagonal rectangles and subtract them to get the difference value.

Taking a detection window of the size 24x24 would lead us to have 180,000 features. This is too many feature for such a small detection window and leads to overcompletion.

The paper introduces the use of *integral images*. Integral image allows for a rapid summation of pixels from, allowing for quick calculation of feature values. This means rather than calculating large number of pixels, we only require four pixels. So an integral



(a) Two Rectangle Feature (b) Two Rectangle Feature (c) Three Rectangle Feature

(d) Four Rectangle Feature

FIGURE 2.3: Haar features relatively showed to the enclosed detection window

image is a sum of pixel from top to left including the origin. And can be represented as:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.1}$$

Where the original image is *i(x,y)* and the integral image as *ii(x,y)*, represented as:

$$s(x, y) = s(x, y - 1) + i(x, y)(1)$$
$$ii(x, y) = ii(x - 1, y) + s(x, y)(2)$$

So, feature 2.3(a) on requires 6 calculations, feature 2.3(c) requires 8 and feature 2.3(d) only 9. This vastly reduces the computation time.

Now we still have more features than we have pixels. At this point we use Adaboost method to reduce this number. What we do is take every feature and apply it to each training image, finding the features that allows for best feature positive or negative classification. We select the features that have the minimal error rate. This process reduces the amount of feature significantly while also improving the accuracy.

For the next step we use cascade classifier. It is used to significantly reduces the time needed for classification, by grouping features into different stages of classification. So for example, if we find a feature that is matches the classifier, but the feature after it does not, we discard that detection window.

2.2.5 Web and Desktop Application Development

There are two major building blocks for the solution proposed in this thesis. First is the producer. The producer or the desktop application will generate the input for the consumer or the web application.

The desktop application is required to process the data (floor plan) and produce an output (structured data). This is a multistage scientific process that requires a well supported set of libraries. Libraries such as OpenCV (Open Source Computer Vision Library) [4] that support languages, such as C++, C, Python and Java.

There are few languages that meet the requirements for this task. Languages, such as C++,C, Java or scientific language such as R or MATLAB have their advantages and disadvantages in this particular field. However, the general consensus in the scientific community that Python is one of the most flexible languages that is easy to develop and test with. The language has seen a general rise in its popularity as its both on the global Tiobe Index [5] where it is places at 5^{th} place, and the open source community on GitHub [6] where its placed 2^{nd}. One of its other added benefits is that it is extremely easy to deploy in majority of environments. For example it would be simple to extends the functionality of this application through a web interface that is hosted on server, essentially making a cloud service.

After the python application generates the output that is presented in such a structured way that it now can be transferred into our web application. There is a variety of different languages that could be used this task, but none of them are quite as versatile or robust as JavaScript. The language is one of the core three languages that are used for web development, namely Cascade Style sheets(CSS), Hypertext Markup Language(HTML) and JavaScript(JS). The versatility of it allows for it to be integrated into web, desktop, phone and hybrid applications.

The goal of the web application is to have an interactable 3D models. In order to render an object in 2D or 3D inside a web browser without any plugin or extensions the web graphical library (WebGL) is used. WebGL will allow us to transfer the structured data output from the Python application to the JavaScript web application.

2.3 A Review of the Thematic Area

Through the research conducted throughout various thematic areas pertaining to the theses, many The focus of this section is at the heart of the project research phase. You must identify the main sources of information you should be aware of within your chosen

area and pay regular attention to so as to strengthen your knowledge in the core topic you are working at. So here you should develop an knowledge of not only your core topic but also about the area of computer science the topic falls under.

More specifically you should research the following:

- International Conferences and Journals
  - A paper published in the 2011 International Conference on Document Analysis and Recognition [2] is one of the primary sources for this thesis. It introduces a three stage floor plan analysis work-flow to help breakdown the process into sequential steps in order to help with automation of the whole process.
  - A paper published in the 2012 10th IAPR International Workshop on Document Analysis Systems [7] proposes a method for automated room detection and room labeling, by building upon the method initially proposed in [8]. The method used separates the walls into thick, medium and thin lines– unlike [8], where lines are just into thick and thin. The method also uses SURF (Speeded-Up Robust Features) for detecting features, such as doors in order to further separate rooms, allowing for up 89% accuracy for room detection and up to 80% for room labeling a major from 50% [8].
  - One of the most recent attempts at creating 3D models from 2D architectural floor plans was written in the paper [9]. It proposed a system where the pre-processed binarised image is treated as a vector image, allowing for extracted lines to be represented as vector coordinates.
  - A paper published in the 2011 International Conference on Document Analysis and Recognition [10] that introduces a method of text extraction from architectural floor plans that produces results up to 99% with a precision of up to 97%.
  - One of the most recently published papers just in November 2017 by Tang R, Wang Y, Cosker D, Li W in a paper named Automatic Structural Scene Digitalization[11] explores a variety of strategies that require to analyze more ambiguous and varied floor plans that are formatted in Computer-aided Design (CAD).
- The top 3 most recent books/texts related to topic.
  - Book written by Richard Szeliski called Computer vision: algorithms and applications [12] has a large variety of techniques used in computer vision field of machine learning.

  - Book called Programming Computer Vision with Python: Tools and algorithms[13] for analyzing images is a library of techniques used for computer vision outside of the OpenCV library in the python language.
  - An book consisting of all OpenCV tutorial called OpenCV-python tutorials documentation [14] is an amalgamation of techniques used in OpenCV using the python language.

- This is a very niche product aimed at producing fast results over the overall quality. However, there is one major market for that in the advertisement sector. The idea when this project was started, was to use 3D models on accommodation rent/sale websites for the goal of alternative way of advertisement. These websites would often have large quantities of properties at any given time. The goal was to be able to generate these models in seconds, with no extra work or cost needed.It was intended to allow the potential customer to view and get a general feel for the property, without actually going to the site. This would potentially lower the costs operation for the advertisers and the customers. The potential market place is huge as language is not a barrier allowing for it to work in any country.

- Other sources of information.

  - YouTube channel [15]. The channel covers few topics pertaining to this project in the python language.
  - The mathematics community network on stack exchange [16]. Here we can find few examples of people working on similar projects.
  - The OpenCV Q&A forums [17]. Has a ton of answers for most of computer vision problems.

2.4 Current State of the Art

Unfortunately, there is nothing alike on the market. The most recent work that is closely related this this project was done in 2017[11]. There it proposes a system for detecting and labeling of computer-aided design(CAD) format floor plans. It proposes a very similar of gradually breaking down floors plans using computer vision techniques to extract appropriate information. The system was initially proposed in the paper [2] and later improved upon in the paper [7]. The same automated system will be used for this project as well and can be seen in detail in the following chapters. There have also been other attempts that tried to build 3D models using architectural floor plans[9], with the most recent in 2008 paper [18]. However, the system used will not work well with the system introduce in [2]. Therefore, we will build upon the system [2], extending

its functionality so it is able to structure extracted information is such a way that it can be represented as a 3D model.

Chapter 3

Problem - Automated three-dimensional translation of floor plans

3.1 Problem Definition

Architectural floor plan analysis has been attempted number of times over the last two decades [19], [20], some have even attempting to take a step further and attempted convert these results to their three dimensional representation [9],[18]. It must be noted that this conversion process is more often than not experimental and aimed to explore the possible solutions rather than producing concrete results.

The goal of this project is to have an automated conversion system of floor maps into 3D interactable models, that would ideally require no external user intervention throughout the analysis stage. However, that would be a perfect scenario, as most floor maps contains a lot variation between each other. The reason behind the variance in the representation of some or all aspects between floor maps are largely dependent on the the person who designed it, the geographic location of the designer and other elements, such as software used.

3.2 Objectives

The goal is to achieve three main aspects of information extraction process from a floor plan and use the extracted information to build a 3D model.

- **Analysis Stage**: Here is a number of objectives required to perform successful analysis of architectural floor plans.
  - Perform successful wall segmentation and analysis, extracting location data.
  - Perform successful symbolic analysis, extracting semantic, location and volumetric(sink, shower etc.) data.
  - Perform successful room detection, extracting both volumetric and location data.
    - Perform successful room labeling based on extracted semantic data.
- **Reconstruction Stage**: Here is a number of objectives required for a successful reconstruction from the data gathered in the analysis stage.
  - Represent the extracted data in such a way that it accurately represents the information presented in the architectural floor plan.
  - Allows for a certain level of interaction for the user with the 3D object.

3.3 Functional Requirements

3.3.1 Analysis stage

- **Preprocessing**: This is the cleanup step.
- **Information Segmentation**: The goal of this step is to extract as much distinct information from a floor plan as possible.
  - Extract text from the image.
  - Extract large thick(External walls) lines.
  - Extract medium(Internal walls) lines.
  - Extract thin(Symbolic objects, such as doors, windows etc.).
- **Structural Analysis**: This stage is primarily aimed at the extraction of structural information using the segments gathered from the previous section. The process is a sequence of steps, rather than a list of independent processes.
  1. Build a complete image of the walls (thick + medium).
  2. Perform contour extraction on the image.
  3. Perform polygon approximation for the contoured image.
  4. Perform gap closing using the extracted walls.
     - Close outside walls, such as large gaps created by windows and doors.

   - Close inner walls, such as small gaps created by doors.
5. Perform boundary extraction.

- **Semantic Analysis**: This step is where all the semantic information is extracted from the floor plan.
  - Use the symbols extracted from the information segmentation step and apply machine learning techniques for identification.
  - Perform room detection.
    * Find relative room size
  - Label rooms.

3.3.2 Reconstruction Stage

- **Software**: The functional requirements for the application can be divided into two main categories.
  - **Application**: These are the basic functional requirements the application
    * The application must provide a way to recognize already assembled models or build new ones from the data collected in the information segmentation stage.
    * The application must provide a view for the model.
    * The application must allows for the user for minimal spacial interaction with the model.
    * The application must provide some interactive elements with the model itself.
  - **Structural Reconstruction Process**: A sequential process that can be viewed apart from the application itself.
    1. Render floor
    2. Render outer outer and inner walls.
    3. Render generic structurally integral symbolic objects, such as doors and windows.
    4. Render any other recognizable symbolic objects in any suitable format.

3.4 Non-Functional Requirements

- The analysis stage must produce results that are at least 80% accurate in order for it to be commercially viable.

- The conversion process (data to 3D) must produce at least 95-100% accuracy to the architectural floor plan.

3.5 Assumptions and Constraints

3.5.1 Assumptions

- Floor maps are using a generic style to denote symbols.
- Floor maps do not use a legend.
- Floor maps are clean or have as little noise as possible.

3.5.2 Constraints

- Floor maps provided must be within a certain resolution range for the algorithm to be viable.
- Floor maps must be at least gray scale or completely binary for the algorithm to work.
- No information must be encoded in color.

Chapter 4

Implementation Approach

4.1 Architecture

The high level overview provided in 4.1 shows a relatively simple representation of the linear system that consists of four steps:

1. Input an architectural floor plan into analysis algorithm.
2. The algorithm breaks down the floor map into its constituent parts.
3. The constituent parts are used to generate a structured data file.
4. The viewing application takes in this structured data file and uses it to compile a model for viewing.

The analysis algorithm can be further broken down into its sub parts as seen in the 4.2.

4.1.1 Technologies involved

- **Analysis Algorithm**: Python, OpenCV.
- **Viewing Application**: JavaScript, WebGL framework three.js[21].

4.1.1.1 Languages

The reason for choosing the two languages of Python and JavaScript were selected can be found in the section 2.2.5.

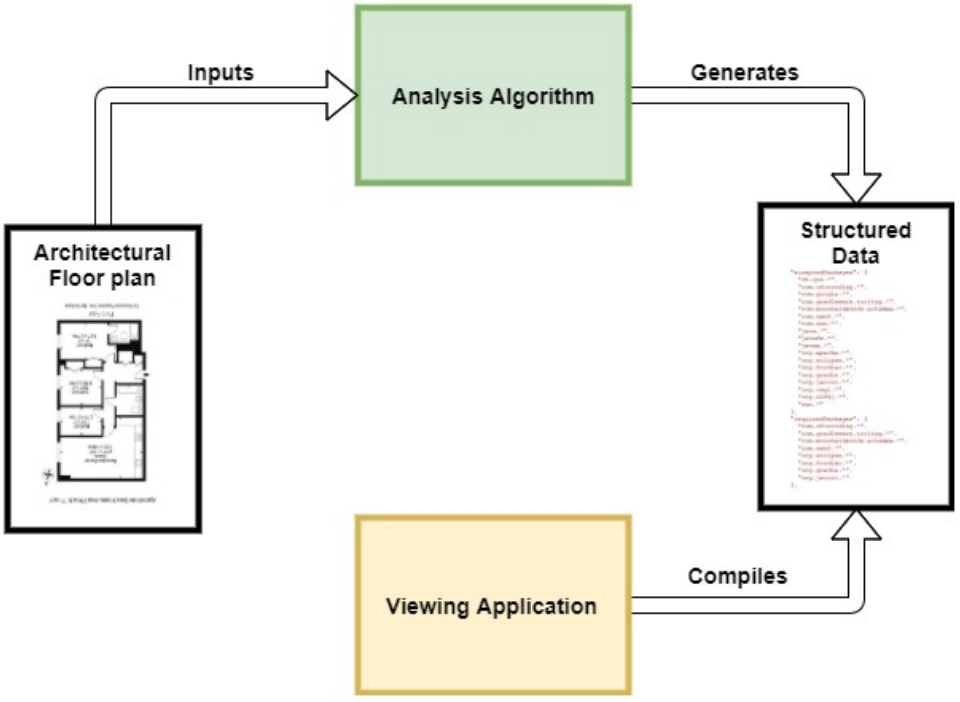


FIGURE 4.1: High Level System Overview

4.1.1.2 Libraries

- **OpenCV(Open Source Computer Vision Library)** : The OpenCV library is arguably one of the most popular ones used for computer vision related problems. It consists of over 2500 different algorithms, making it highly versatile and adaptable to any problem. Majority of the algorithms used in this project have some implementation of it in the library.
- **three.js**: Three.js is a library built upon the WebGL API that is used to render 2D and 3D objects within the browser and by extension working almost on any device. The reason for using a library over working with WebGL API itself is that it makes the building process much simpler for a beginner. The choice for the library to be used in this project is still not permanent, however, it is highly likely that it will be three.js.

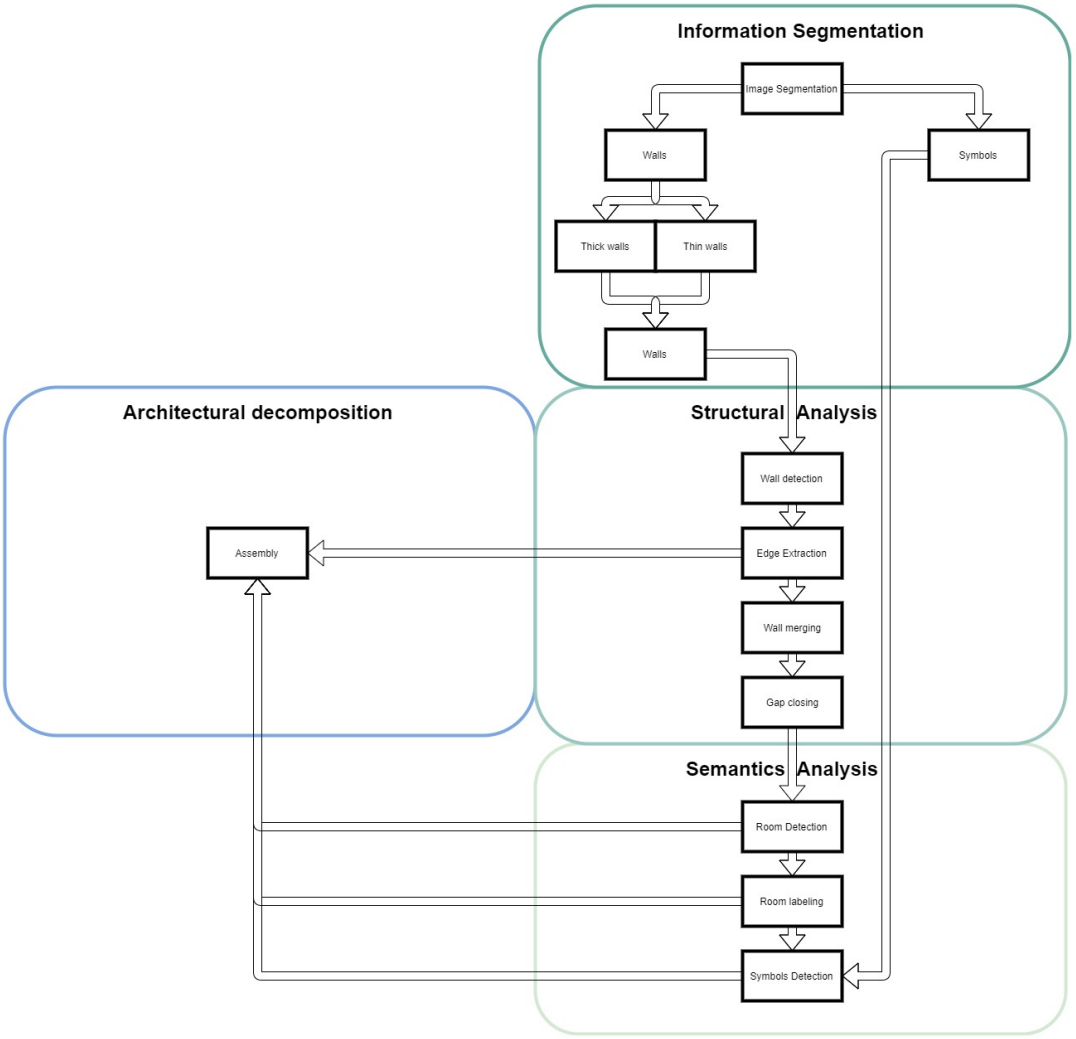


FIGURE 4.2: High Level System Overview

4.1.2 Information Segmentation

4.1.2.1 Morphological Erosion and Dilation

The preprocessing section goal is to remove any noise from the image that may interfere with the structural analysis of the floor plan. Noise within an image is considered anything that may cause unwanted or unpredictable results after running through the computer vision algorithms. Objects such as text, symbols (sinks, doors, windows etc.) and other miscellaneous items such arrows if not properly removed may cause unwanted results within the structural and semantic analysis stages. Noise removal within this project is is done through the use of morphological image processing techniques of erosion and dilation. These processes are described in 2.2.3 as part of the thematic areas.

For preprocessing of floor maps we use a 3x3 size kernel/structuring element as was suggested in [2]. The process is begun by the use of morphological erosion that is done n number of iterations to remove as much noise as possible without removing any essential information such as inner walls. The **n** of iterations is a dynamic number as it is effected by the kernel size and the resolution of the floor plan e.g. 240x360 as used in the example 4.3. In 4.3 we can see that all information except the structural was removed. Allowing for the next step witch is dilation.

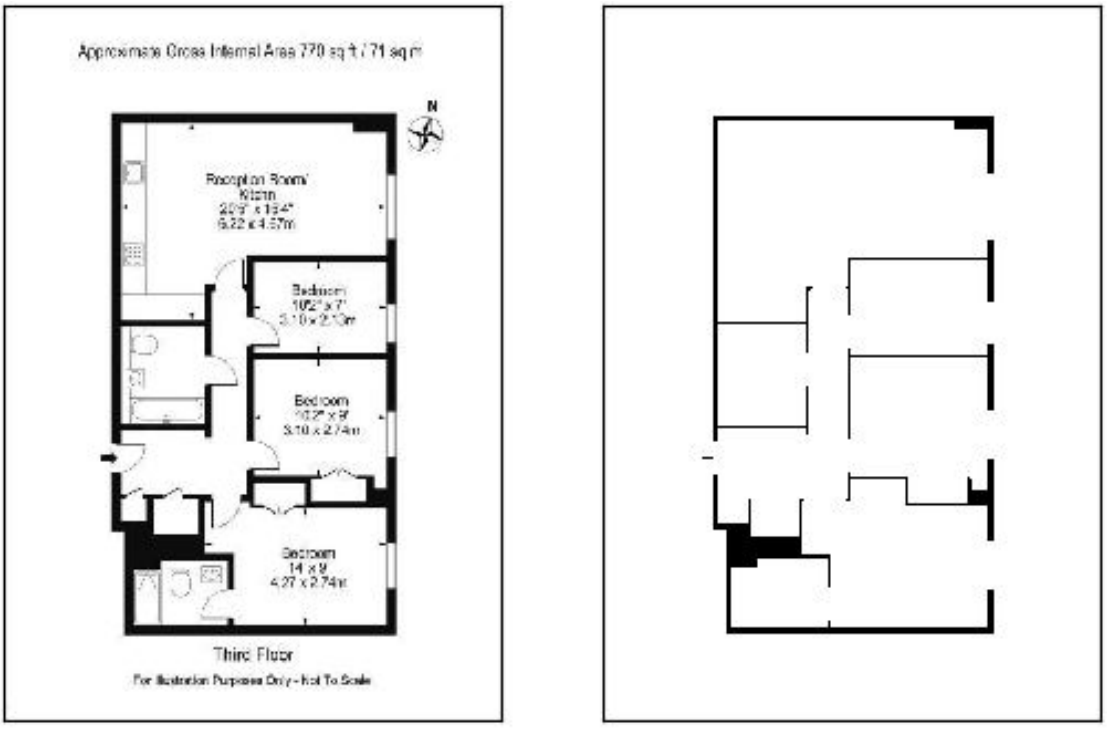


FIGURE 4.3: Process of erosion comparison

Dilation is the process that is used to allow the elements within the floor plan to go back to the original size. However the same kernel and iteration number must be used as done in the erosion process. Here we apply dilation process to 4.3 and this results to 4.4. As we can see the walls have both thick and thin are back to the original size with minimal loss. The next step is to separate the walls from the symbols from the floor plan operation. This is a rather simple operation that can be done in OpenCV. If we take the image 4.4 and do a simple subtraction from the two images allowing us to separate out the symbols leaving with results seen in 4.5.

4.1.3 Structural Analysis

The structural analysis stage is where the extracted segments from the segmentation stage is run trough a series of algorithms to extract structural data. The final goals of

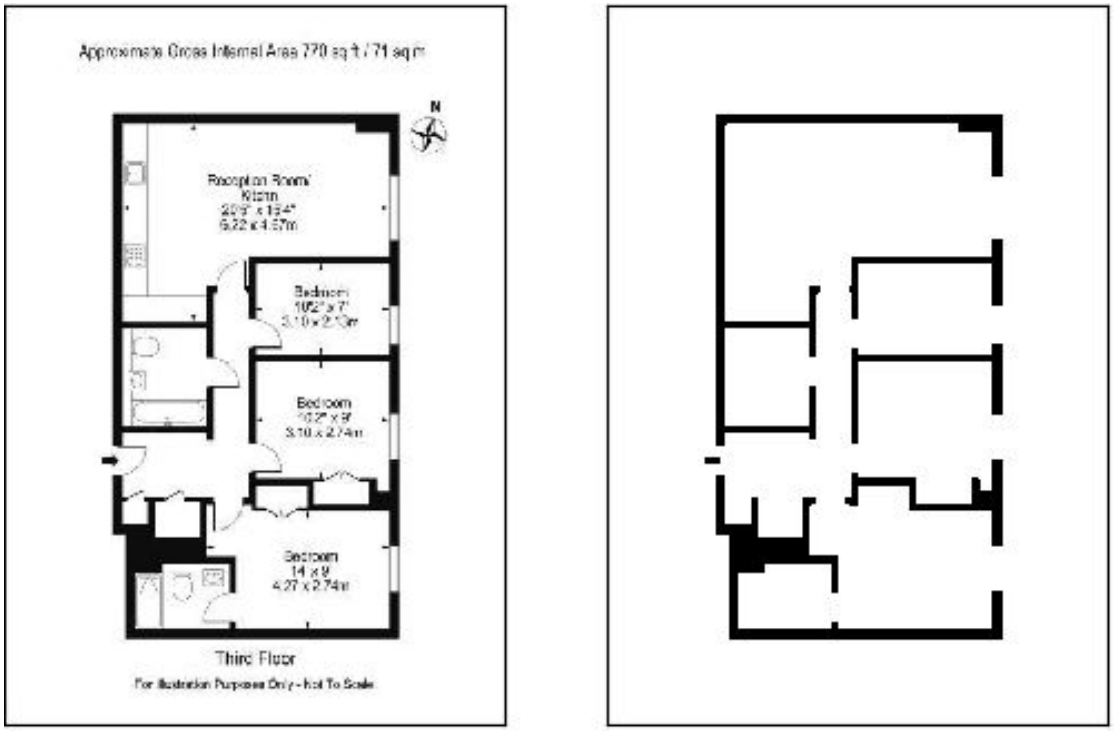


FIGURE 4.4: Process of dilation after erosion comparison

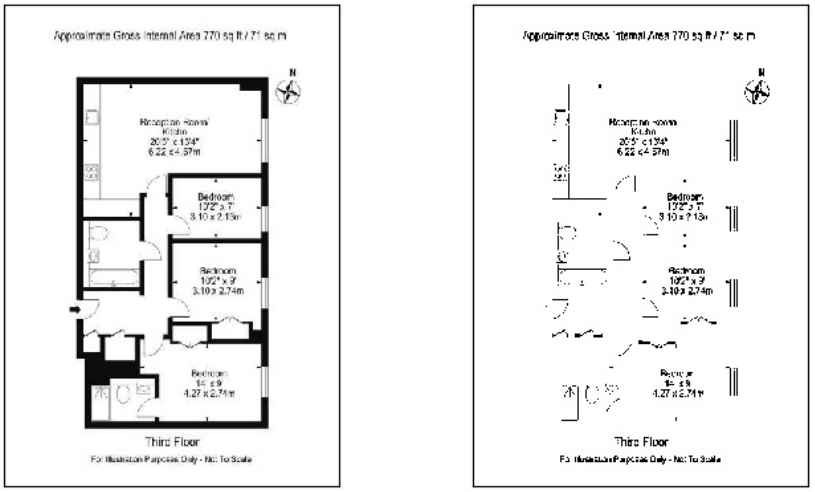


FIGURE 4.5: Subtracting walls from original to separate symbolic data

this section is have an image that can be used to extract rooms in the semantic analysis and to find the vectors for the reconstruction stage. Unlike the previous section this section has a lot of possible ways of approaching, therefore, the information written in this section is susceptible to change.

The process of structural analysis is a sequence of processes that entails the following steps:

1. Detect walls.
2. Detect edges of the walls.
3. Merge gaps where short edges occur.
4. Extract floor boundary.

In order to detects walls first we need to find the walls. One good way of this is by applying the contour extraction algorithm

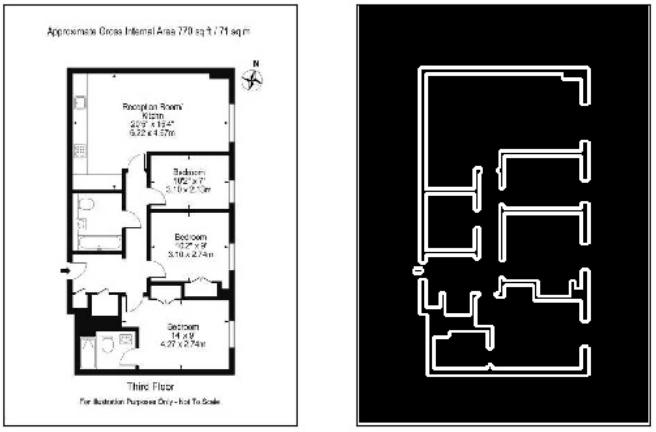
4.1.3.1 Contour extraction

Contour extraction is the first step of vectorizing walls of the floor plan. There are few variation of the algorithm, but the one that is going to be used in this project is an implementation of [22]. The algorithm works on binary images where it essentially look for abrupt change or a border. This border region is found in location where there is abrupt change going from 0 to 1. The locations between these regions are called holes that are followed by a border. The result is a border representation of a binary image as seen in 4.6 which contains 7 distinct border regions.

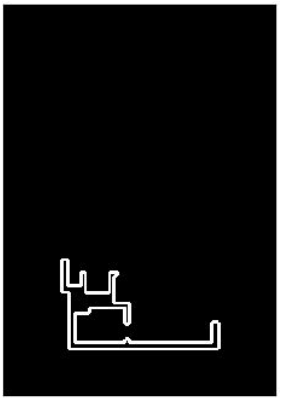
Each of the regions is a parent that contains multiple distinct child points within this region. For example the border seen in 4.6(b) contains over 1139 points. Unfortunately, this is too many points for the step described in 4.1.3.3 as too many points are clustered around the corners. There what it is required to reduce this number by the use of polygonal approximation.

4.1.3.2 Polygonal Approximation

The point of this step is make the image vectorizable as in every point represent a corner rather than a random point on the border. So for example in case of a rectangle of any size would be reduced just to 4 points. This method requires us to condense any horizontal,vertical and diagonal border points to just a single corner point. Luckily OpenCV has such a method of compression, however, it is unclear of what algorithm it is using for this. Using this method it allows us to reduce the size of a border the contour of size 11394.6(b) to just 53 points, with the minimum being 32.



(a) Contour extraction



(b) Single Contour

FIGURE 4.6: Contour extraction: Border representation of a binary image

4.1.3.3 Harris Corner Detection

One of the better know corner detection algorithms was comprised by Chris Harris and Mike Stephens in the paper[23]. The algorithm known as Harris Corner Detection works by finding the difference in the change of intensity of the shift *(u,v)* defined by the equation 4.1.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \tag{4.1}$$

The point of this algorithm is for us to find the edge points of each wall in order to identify the wall edges where windows and doors are located. The algorithm has an

implementation in the OpenCV library, making easy to use. The result would look something like this 4.7.

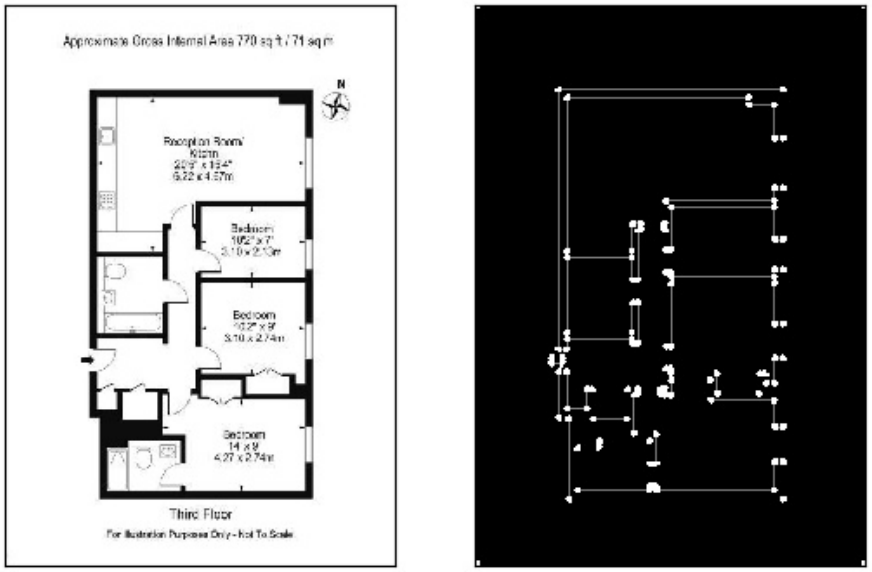


FIGURE 4.7: Extracting corners using Harris corner detection algorithm

4.1.3.4 Gap closing

Now that the edge points have been found; we shall use these edge point to find the wall edges. A wall edge consist of a short line where both angles are either convex($<90°$) or concave($>90°$).

The goal of finding these wall edges, is to fill in the gaps between them. These gaps are located where doors or windows should be located at. The assumption is that, if the distance between two edge walls does not exceed the threshold T, it is safe to close the gap between them. Unfortunately, this only works for doors and some windows, but does not work in all cases. If for example, the area of a large window exceed the threshold T, it shall not be able to close the gap. The problematic areas are usually found the outer wall.

4.1.3.5 Convex Hull

This algorithm will allow us to find the absolute outline of the the floor plan. In case of this project it is an optimal way to find the outline of the the floor plan for which we will use to close up the final gaps in the wall. Convex hull can be defined as a smallest polygon built to surround a field of n number of points in a euclidean plane. The OpenCV library has a nice implementation that uses the worst-case complexity of *O(n log n)*, first introduced in paper *Finding the convex hull of a simple polygon* [24]. We will use the outer wall extracted in the information segmentation stage 4.1.2 to find the convex hull of the floor plan.

Using the polygon that was extracted we will apply vertical and horizontal smearing along its bounds. This step is done to close any large gaps that may be located along the walls.

4.1.4 Semantic Analysis

The goal of semantic analysis is to extract any or all semantic information pertaining to the floor map. Semantic in this context means the identification and interpretation of any building elements within within the floor map. These elements may consist of:

1. Detecting and identifying rooms.
2. Detecting and identifying symbolic objects.

Now with the fully enclosed walls image that was built in the previous stage we apply variate of algorithms to define the rooms. One of the to do this is by the distance transform plus connected components algorithm.Just like the previous section, the techniques used here are susceptible to change.

4.1.4.1 Distance Transform

Distance transform serve the function of finding the centers of each room. It takes a binary image as input where 0 represents empty and 1 represent boundary. The goal is to find the distance measurement from the nearest boundary. The distance from the boundary is denoted by color. The closer it is to the center, the lighter the gradient color is. An example of this can be seen in 4.8

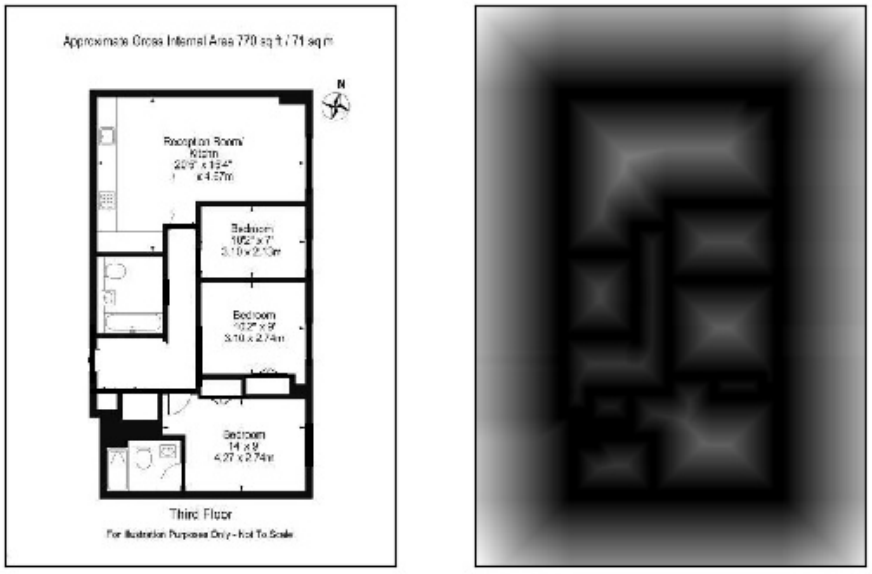


FIGURE 4.8: Finding the centers of rooms using distance transform algorithm

4.1.4.2 Connected Components

Connected components is an algorithm used to label different objects within a binary image. This is done by traversing the image pixel by pixel, checking each of the surrounding pixels(top, bottom, left and right) and marking them accordingly. Not all regions are marked, boundary region that are marked by 0 are skipped. Each region is marked with a label allow us to access information such as the area of the region. In 4.9 can see this in effect, where each region is colored with a unique color.

4.1.4.3 Haar Cascade

Haar Cascade was described in detail in the object detection section 2.2.4. Initially the algorithm will be trained on select few symbols, such as doors, sinks and toilets. Allowing to find the position and orientation of each of the objects.

FIGURE 4.9: Finding the centers of rooms using distance transform algorithm

4.1.5 Architectural decomposition

This goal of this architectural decomposition is to take peaces of information gathered through the previous stages to assimilate a structured layout of the floor plan. To do this we must separate the information into chunks, where each chuck represent a unique structural feature. The structure will be broken down into:

- **Wall polygon**: An array of corner points/ vectors. There can be multiple wall polygons within a floor plan.
- **Room**: Each room is a set of coordinates defining its position/dimensions and its name.
- **Object**: Each object, such as door will be assigned a type e.g. door and position.

Properties, like the height of walls or the position of objects that depend on vertical space for placement, such as windows will be generic values. The proposed file format for this structure is JavaScript Object Notation(JSON).

4.1.6 Reconstruction

The goal of the reconstruction stage is an independent stage where the constructed file in 4.1.5 is used to build the model of the floor plan using one of the 3rd party JavaScript WebGL libraries, such as three.js.

4.2 Risk Assessment

This section is for identifying potential risks that the project may face and according step to reduce them. The table 4.1 is used to help with the classification of these risks.

TABLE 4.1: Initial risk matrix

Frequency/ Consequence	**1-Rare**	**2-Remote**	**3-Occasional**	**4-Probable**	**5-Frequent**
4-Fatal					
3-Critical					
2-Major					
1-Minor					

- **Insufficient Skills** (*Probable and Critical*): This project requires a substantial amount of knowledge outside the scope of the college course. Time for suck cases must be allocated for each deadline.
- **Time management** (*Occasional and Major*): Some areas of the project will require more time investment than others due to the unknown knowledge factor. It is important include a buffer window for each step for such cases.
- **Backup** (*Rare and Minor*): Loss of source code. This is mitigated by constant backup to services such as GitHub.
- **Productivity** (*Occasional and Major*): Loss of productivity at any point may have severe consequences at the end of the project. It is important to include time-off in the time line of the project.
- **Insufficient Testing** (*Probable and Minor*): The algorithms must be run through multiple test cases to insure that each step poses works as intended and pose no risks for the next ones.
- **Technical Risks** (*Remote and Critical*): Risk such as not being able to meet deadlines due to task complexity. In such cases the scope of the project will be reassessed and according reduction to the functionality will be made.

4.3 Methodology

Each step within this project is standalone set of problems pertaining to that step. Some steps bleed into others, but for the most part they are separate. Therefore, each step requires independent research to accomplish. There is a variate of resources pertaining to each step, mainly focusing on the specific algorithm required to perform a certain task in computer vision. However, there are no streamline explanation of the process outside of some research papers, so a lot of the information simply have to be gathered from variety of resources, such as books, papers, YouTube, stack-overflow and other sporadic sources.

The favoured approach to this problem is to use agile process to break it down to its constituent parts, rather than working on the whole. Each task shall be divided into separate sprints consisting of development, research and testing. This will allow me to approach each step as an individual problem.

4.4 Implementation Plan Schedule

The plan is to approach this problem with the agile process. The flow of the project works really well with this. Each sprint is a standalone objective and its tasks can be that directly mapped to the requirements in the 3.3 subsection. Breaking down the monumental task that does not have much visual feed throughout the process into its constituent parts. This allows to focus on the detail for each individual task. The schedule can be seen in 4.10.

4.5 Evaluation

For the evaluation the dataset provided by [21] consisting of 215 apartment floor plans. Each plan averaging around 400x600 pixel resolution. It is difficult problem in itself to accurately evaluate computer vision problems. Here we will select object detection as it is easiest to classify. Other measurements may be added later.

4.5.1 Object detection

Because our classifier works in binary, we will use a system established in this paper[1] to measure the success of the system. It uses 4.2 classes for measuring.

Using these will will measure the:

Sprint	Task	Time Allocated
Sprint 1	Object Detection	3 weeks
Sprint 2	Image Segmentation	1 week
Sprint 3	Structurual Analysis	1 week
Sprint 4	Semantic Analysis	1 week
Sprint 5	Architectual Decomposition	1 week
Sprint 6	Testing	1 week
Sprint 7	Reconstruction	2 weeks
Sprint 8	Testing	2 week
	Total Time	12 weeks

FIGURE 4.10: Planned schedule for the project

Data class	Classified as *pos*	Classified as *neg*
pos	true positive (*tp*)	true negative *fn*
neg	false positive (*fp*)	true negative *tn*

TABLE 4.2: Performance measuring classifications[1]

- Effectiveness of the classifier

$$Accuracy = \frac{tp + tn}{tp + fn + fp + tn}$$

- Precision of the classifier

$$Precision = \frac{tp}{tp + fp}$$

4.6 Reconstruction Application Prototype

The project requires an application for compilation and display of the model. The application will implemented in its most basic form due to time limitations and complexity of the project. The application will comprise of the following:

- **3D model space**: Area in which the model of the floor plan will be displayed.
- **Free mode**: This will allow for the user to switch to the three dimensional viewing of the model.

- **Top down mode**: This will allow for the user to reset their view to a limited top down view.
- **Import model**: A button that shows a prompt for the user to upload the structured JSON file that was composed in the 4.1.5.

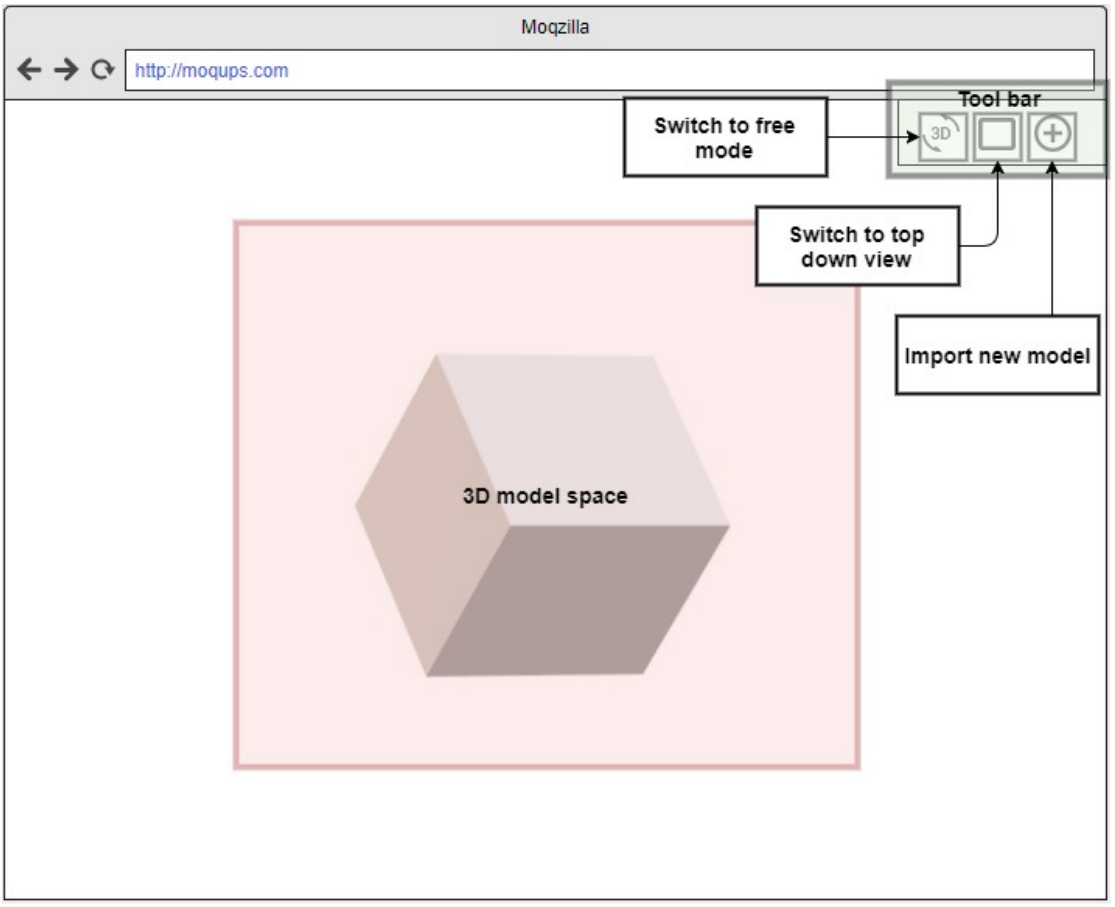


FIGURE 4.11: Prototype of the Web application

Chapter 5

Implementation

5.1 Introduction

The main objective of the project was to develop a system that can automatically translate architectural floor plans to their 3D representation. Due to this there was not a heavy emphasis on the type of languages that would be used for the development as they could be easily interchanges.

5.1.1 Languages

The main language used for the analysis stage of the project was python. This was easiest choice, as the language is quick and easy to use. That allows for rapid implementation and testing of each feature.

The language for the development of the 3D web application is JavaScript. This was the most straightforward choice, due to its utter dominance and no true alternative in the web market.

5.1.2 Tools

OpenCV or Open Source Computer Vision Library is arguably the best library for computer vision problems. This is massive library that contains over 2500 different algorithms, proving a solution to almost any computer vision problem. Every issue through the analysis stage of the project was solved through the assistance of OpenCV algorithms.

The web application was developed using a WebGL library called three.js. It is the most popular and robust library for JavaScript 3D application development. Many of the complexities from WebGL API are abstracted in three.js making the generation of various structural elements, such as walls, very straightforward.

5.2 Development: Analysis Stage

The fundamental variance in architectural plans brings a plethora of challenges. Every variance has placed certain limitations upon every step of the analysis and reconstruction stages. Therefore, each step has been simplified to most fundamental logical functionality and can only serve as a proof of concept.

To understand why steps were taken in a certain way it is important to underline the limitation that were placed upon this project during the analysis stage.

5.2.1 Information Segmentation

The information segmentation is the most fundamental of steps, yet it is one of the most difficult to do correctly. The fundamental goal of this step is to segment structural data into three categories of text, walls and symbols. The later two are where the difficulties and possible confusions may present themselves.

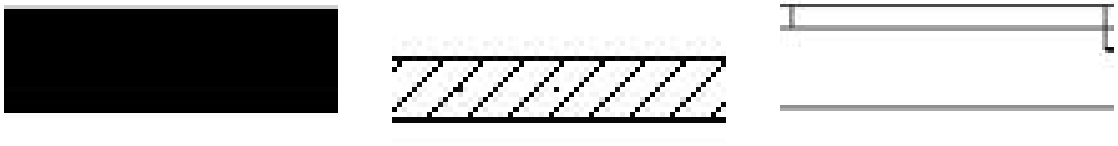
Of the two mentioned, the segmentation of walls is the most critical to get right. Each of the subsequent steps are highly dependent on the walls being correctly segmented. There are two fundamental problems that may pose problems, these are wall of structure and symbols.

The wall structure in an architectural floor plan can be presented in numerous ways, however, there are three fundamental ones.

First and foremost is the filled wall 5.1(a). This is the most common type and is the type that this project will focus on.

Second and third types are the semi-filled wall 5.1(b) and the empty wall 5.1(c). These types of walls are the kind that the segmentation algorithm used for this project cannot handle. To understand why this is the case we have to first delve into what the algorithm does in this stage.

The first step taken is the extraction of the outer walls of the building. This process depends on the morphological operations of, erosion and dilation 2.2.3.



(a) Filled wall (b) Semi-filled wall (c) Empty wall

FIGURE 5.1: Wall structures types

In the process of erosion any thin line is removed leaving only the thickest that the algorithm allows. This process removes any possibility of leaving semi-filled 5.1(b) or empty 5.1(c) wall types in the image. For this not to be the case a different method of segmentation must be done.

However, for filled walls we can depend on erosion and dilation to segment structural data. Each of the operation depends on two parameters, kernel structure with size and the iteration count. In basic terms the larger kernel size the more pixel it will remove/add and the iteration number denotes how many times an operation is performed. These two parameters must be identical for both operations.

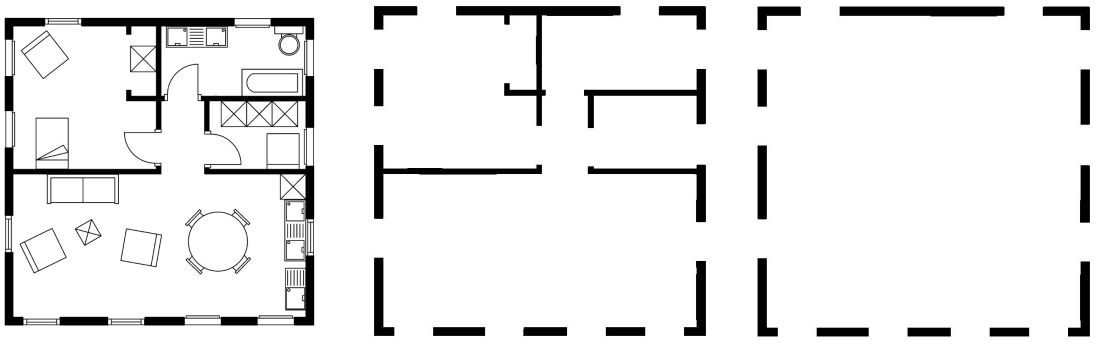
These two parameters may vary between architectural plans due to one reason, resolution. The resolution of the image provided plays a key role defining the two parameters. A lower resolution may require a lower a smaller kernel and less number of iterations opposite would be for a larger resolution.

The general approach is to keep the kernel size small such as a square of 3x3 and simply increase the iteration number to produce the wanted results. However, this approach has one major problem. For larger images it may cause deterioration of corners walls which in the structural analysis stage may produce erroneous results.

So, for the architectural floor plan 5.2(a) of resolution 1629x1604 the optimal kernel size is a square of 5x5. It requires three iteration of erosion and dilation to produce an image of all walls 5.2(b) and eight to reduce it to just outer walls 5.2(c).

The final peace of structural data is generate by subtracting outer walls image 5.2(c) from the inner walls with outer walls image 5.2(b) resulting in image of just inner walls 5.3.

Unfortunately, the segmentation task can produce does not distinguish between what is a wall and what is symbol. Some symbols may be formed from thick lines that can be mistaken for walls. An example of this can be seen here 5.4 where a symbol will pass the erosion process and will be considered as part of the outer wall. Fortunately, this is an edge case, as most floor maps do not have thick symbols such as this.



(a) Floor Plan 1629x1604 (b) Inner and Outer walls (c) Outer walls

FIGURE 5.2: How resolution affects morphological operation parameters

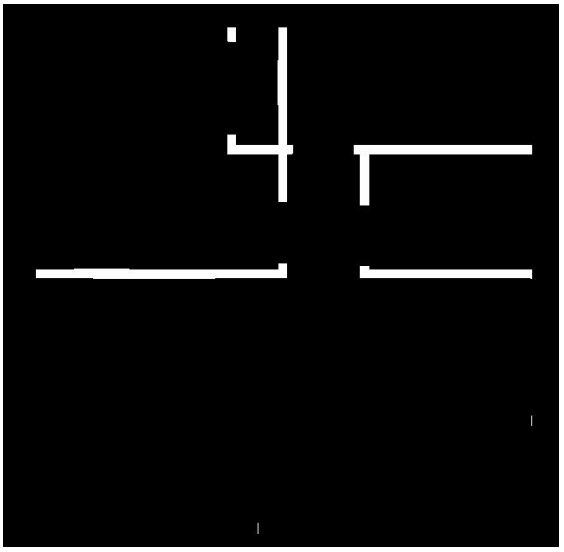


FIGURE 5.3: Results of subtracting 5.2(c) from 5.2(b)

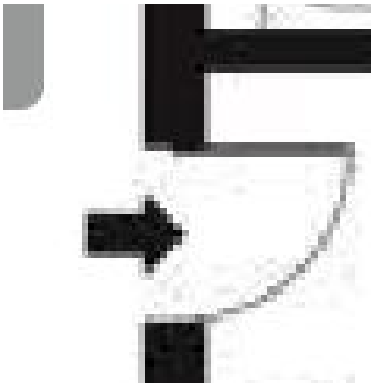


FIGURE 5.4: Symbol left after the process of erosion

5.2.2 Structural Analysis

In this section the following structural data will be extracted:

1. Contours from inner and outer walls.

2. Corners from each wall pertaining to found contours.

3. Closing areas of outer walls.

5.2.2.1 Contour Extraction

Now with external and internal walls being extracted, it is now possible to find individual walls. This done through finding contours, because of the way the algorithm works 4.1.3.1 it allows to find individual walls. In conjunction with the contours extraction algorithm polygonal approximation 4.1.3.2 is also used. Through polygonal approximate the wall structure is simplified, this vastly reducing the complexity of the wall.

The extracted wall contours can now be drawn on a blank canvas, as seen 5.5. OpenCV provides a simple way of doing this through the function drawContours. The objective of drawing each of the contours individually rather than drawing all of them on a single canvas, is that it allows for a logical separation for the following steps.

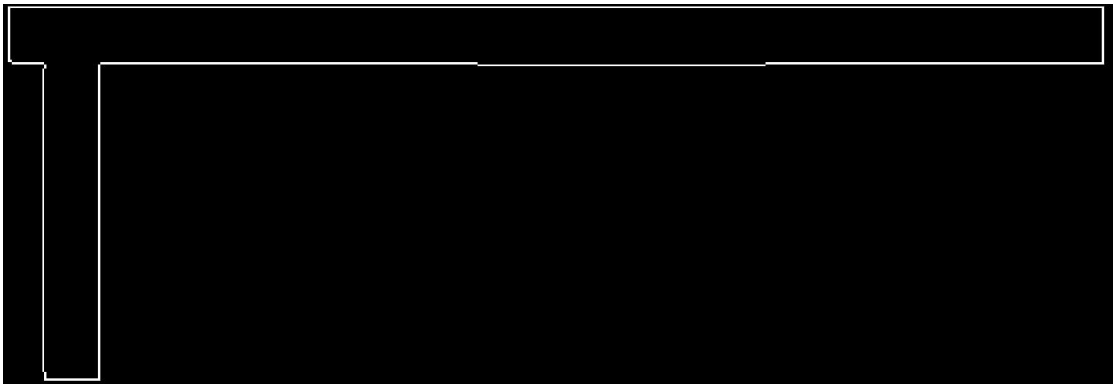


FIGURE 5.5: Single contour

5.2.2.2 Corner Extraction

The next step is to find corners in each of the contour images. This task is accomplished through the use of Harris Corner Detection algorithm 4.1.3.3. This is arguably the most important step through the analysis stage. If done incorrectly or corner points are not detected it will affect the remaining steps.

The main reason why this step may fail to produce desired results fundamentally boils down to the quality of the architectural floor image. Even with high resolution images anomalies can occur. Some of these can be accounted for such corners appearing on a straight line as seen 5.6. The reason for these can often be credited to to the blur caused by image compression algorithms such as JPG or PNG.

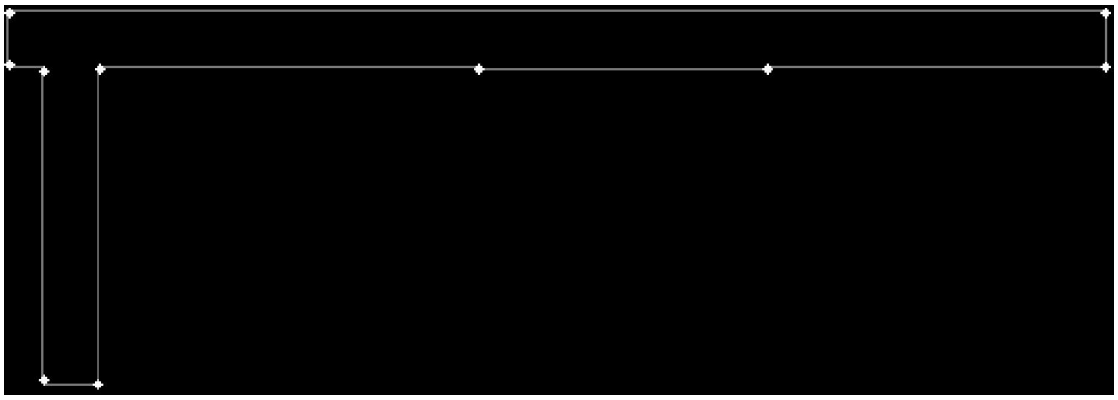


FIGURE 5.6: Harris corner detection anomalies

5.2.2.3 Corner Translation

The extracted corners from each the contours is the first peace of translatable data extracted from the image. However, there is one major issue with it. If the data is used as is, in the reconstruction stage it will form jumbled up wall models. The reason behind this is the way that corner detection works. It looks for corners, row by row going in a downward direction as show here 5.7.

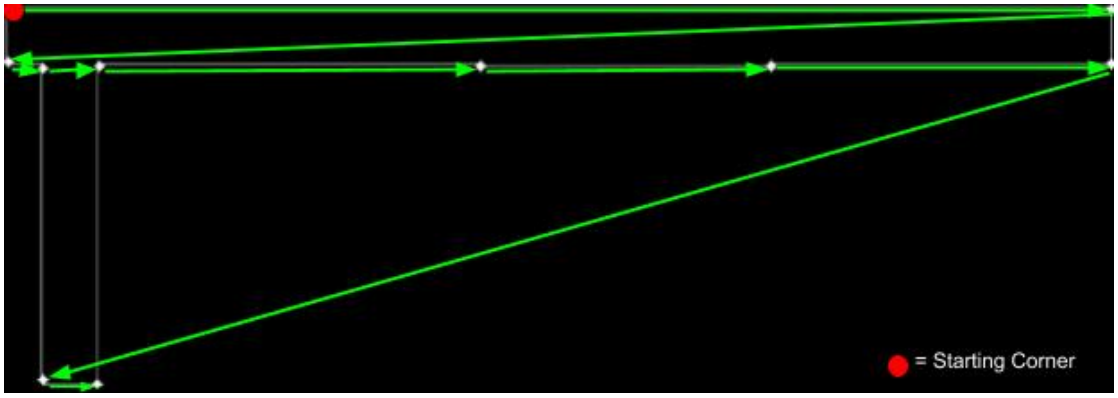


FIGURE 5.7: How Harris corner detection stores corners

There is a lack of structure that must be accounted for. The solution proposed is to use the extracted corner points in conjecture with the image of the wall contour, to walk along the wall. To further explain this we must first understand the structure of the wall.

A typical wall consists of right angles. There are four possible orientations for these angles, seen here 5.8.

To find out the type of a corner, we first take the coordinates of it. We use the coordinate of the corner on the contour image of the wall. Then check all four directions from that point for non zero values. Now, there is a problem not all corners are aligned, mainly

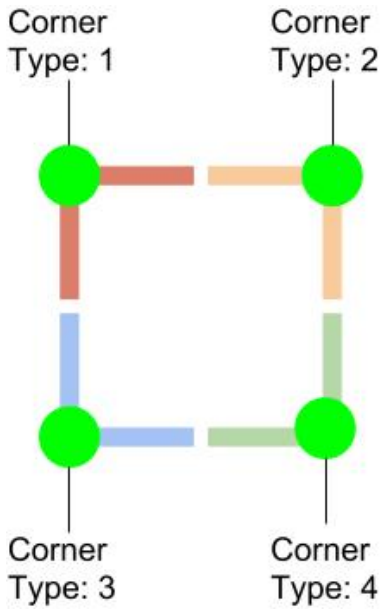


FIGURE 5.8: Wall corner corner types

due to the quality of the image. Sometime a corner can be few pixels off the line, making it difficult to detect the type of a corner it is. The solution is simple, rather than doing a single pixel line we increase the thickness. For higher resolution image around five pixel is a good number. By understating which two of the four direction the corner contains non-zero values, from this it's possible to derive the type of a corner it is.

The next step after finding out the corner type is to see what corners exist in the two directions. However, now we hit the same corner misalignment problem. The solution is the same, simply by increase the thickness of the search it is possible to hit points that otherwise would be missed. The illustration of this process can be seen here 5.9.

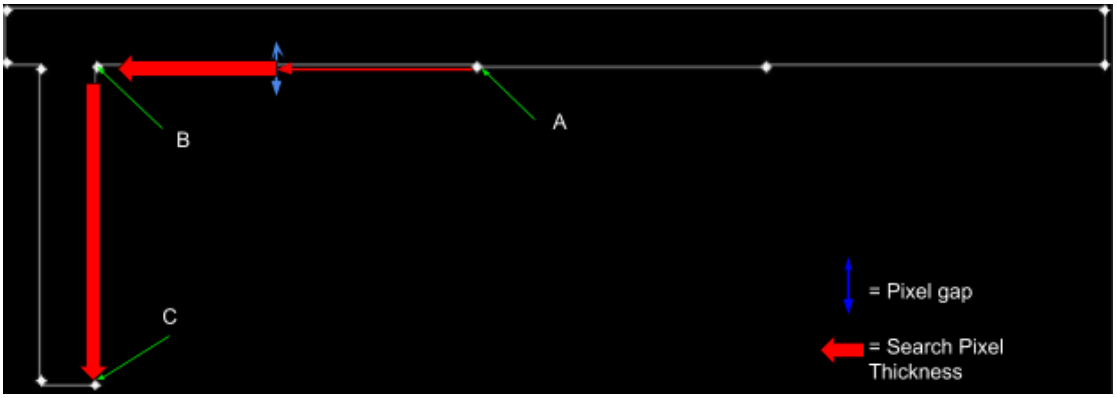


FIGURE 5.9: Searching for the next corner

At the end the process should look like this 5.10. Now corners follow a logical structure, allowing us to use it in the reconstruction stage, except for one minor detail. Because

of the misalignment of corners, some lines have a slight tilt that is visible in the reconstructed model. The solution here is to simply shift the next corners x or y values to match the currently selected corner. This will make all corners follow a straight line, with only a slight alteration.

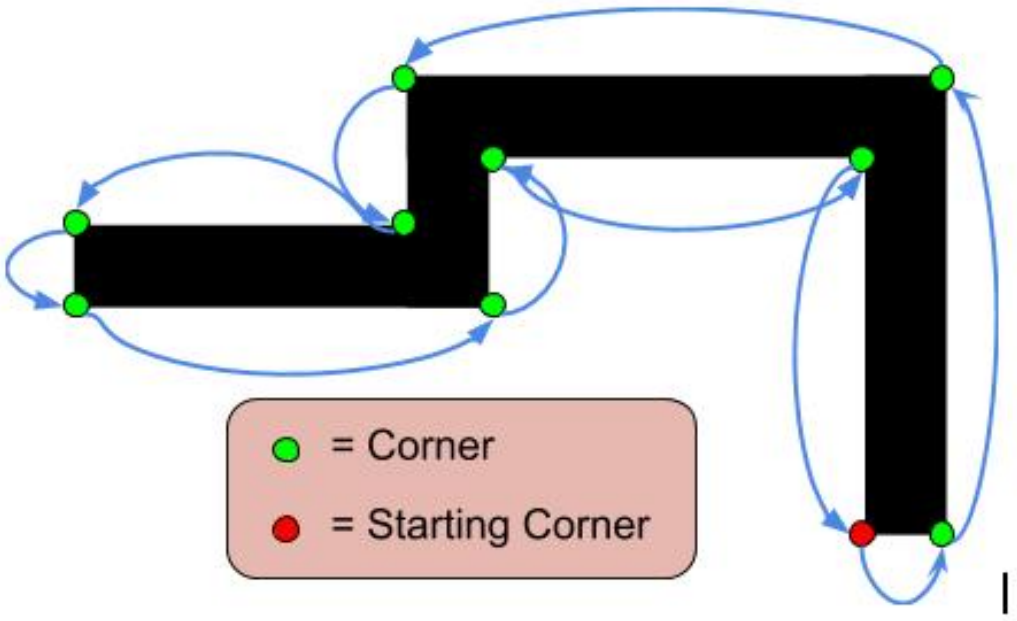


FIGURE 5.10: Walking contour using corners

5.2.2.4 Finding Outer Wall Closure Areas

The next step is to find the closing areas in the outer wall. The simplest approach to this is to assume that each wall segment is ever connected by only two sides. A side of wall can be defined as any two connecting corners of a rectangle, as illustrated here 5.11. There are only four possible unique pairs.

This process follow a similar algorithm as seen in 5.10. The only difference is that we store corner pairs that are match one the four template as seen in 5.11. Here is the illustration of results of the process 5.12

After collecting all viable corners for each of the walls it time to filter the down based on the distance between the two corners. The reason behind this, as mentioned before, is to find the two sides of wall that are most likely to be connected to another outer wall. Therefore, we pick the top two results with the shortest distance between points.

Once filtered down to just two pairs for each wall, it's time to find the connecting walls. In the previous step when the pairs were made, another peace of information was stored

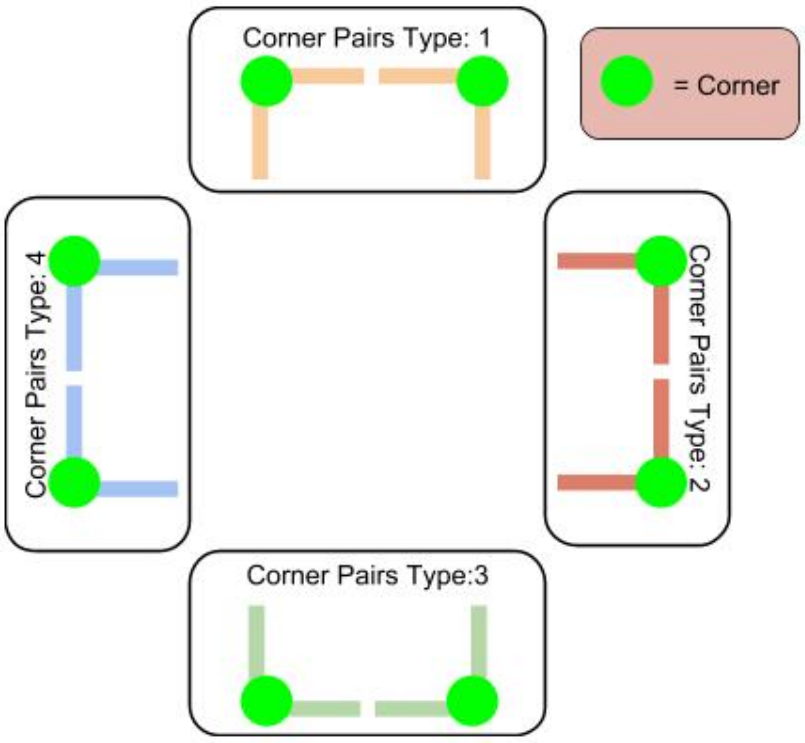


FIGURE 5.11: Viable corner pairs

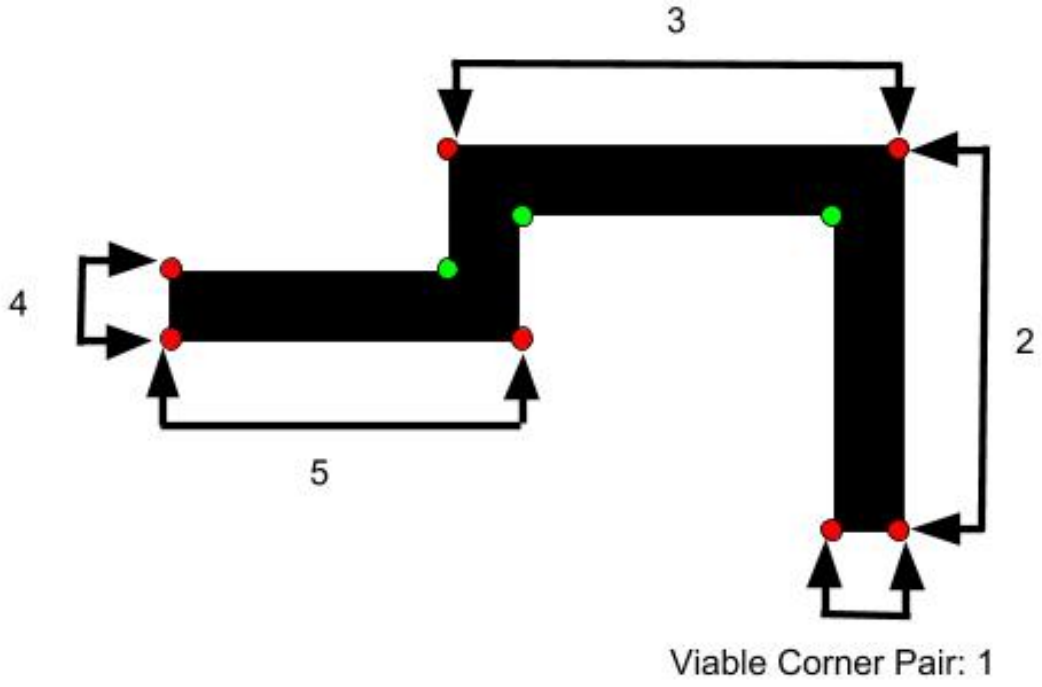
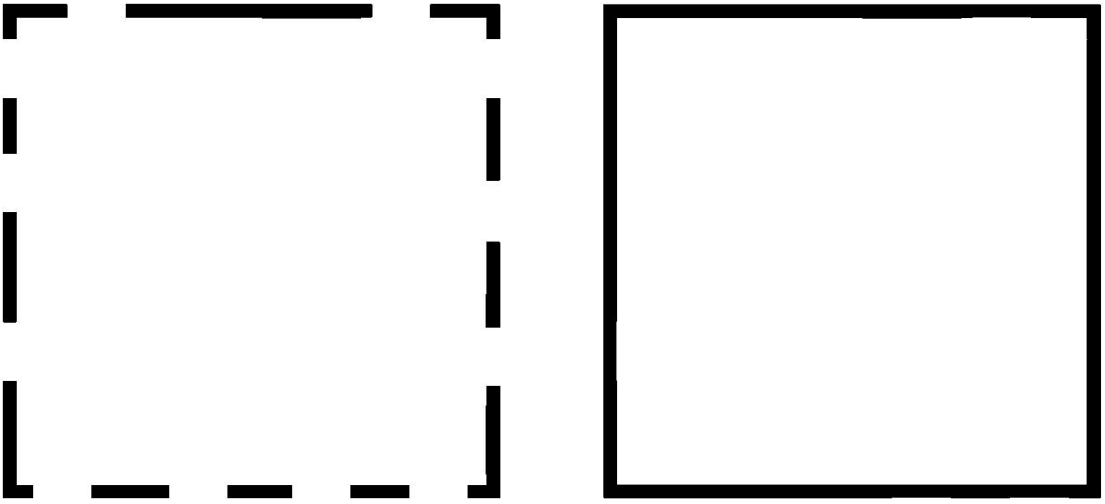


FIGURE 5.12: Finding viable corner pairs

alongside the distance between two points. That peace of data is the axis the side of the wall is facing. This reduces the calculations by half.

The next step is to simply find the matching sides that are closest and have the same distance between their two points. Each matching pairs form a closing area that is a rectangle and can be drawn to fill out the gaps in the outer wall 5.13(b). Finding the closing areas for the outer wall has two main objectives. First it allows us to find out the positions of all windows and possible doors on the outer wall. The second aspects is that it allow us to figure out the whole layout of the building, which is not implemented in this version of the application.



(a) Outer walls (b) Outer walls closed

FIGURE 5.13: Drawing in the closing area onto the outer wall image

5.2.3 Semantic Analysis Analysis

The purpose of this stage has two main purposes:

- Detecting and identifying symbolic objects.
- Detecting and identifying rooms.

5.2.3.1 Object Detection: Doors

Due to the time it takes to prepare dataset and train a machine learning model such as the Haar Cascade classifier 2.2.4, object detection was limited to just doors.

For training a set of 50+ positive door images were used 5.14. These images and 500 negate images from image-net were used to generate 500 positive images. For training

the ratio used was 500:500 positive to negative. The training was performed for 20 cascade stages. The images used were at the scale of 40x40 pixels allow for a more accurate but time consuming training.

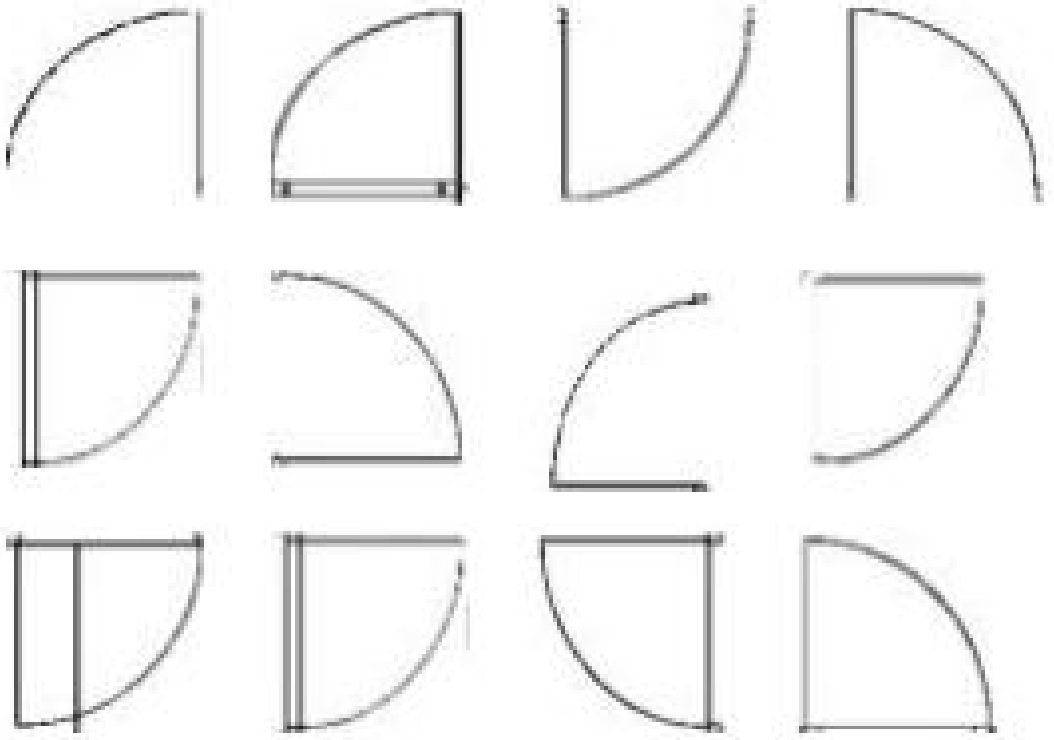


FIGURE 5.14: Sample doors used for training

The results of the training, however, are less then satisfactory, as seen here 5.15. This can be attributed to multitude of factors such as limited image data set for both positive and negative images. It can also be the resolution of the training images or the number of stages used.

Even with poor results 5.1 as long as doors are detected, which in this case they are, the results are usable. To understand why, we first need to delve deeper into what the object detection results represent. To be more precise, the green box that marks the results. This box is used a search area in our algorithm to detect walls that need to be closed, best illustrated here 5.16.

TP	FP	TN	FN	Precision	Accuracy
3	16	3	0	0.15	0.3

TABLE 5.1: Evaluation of Classifier

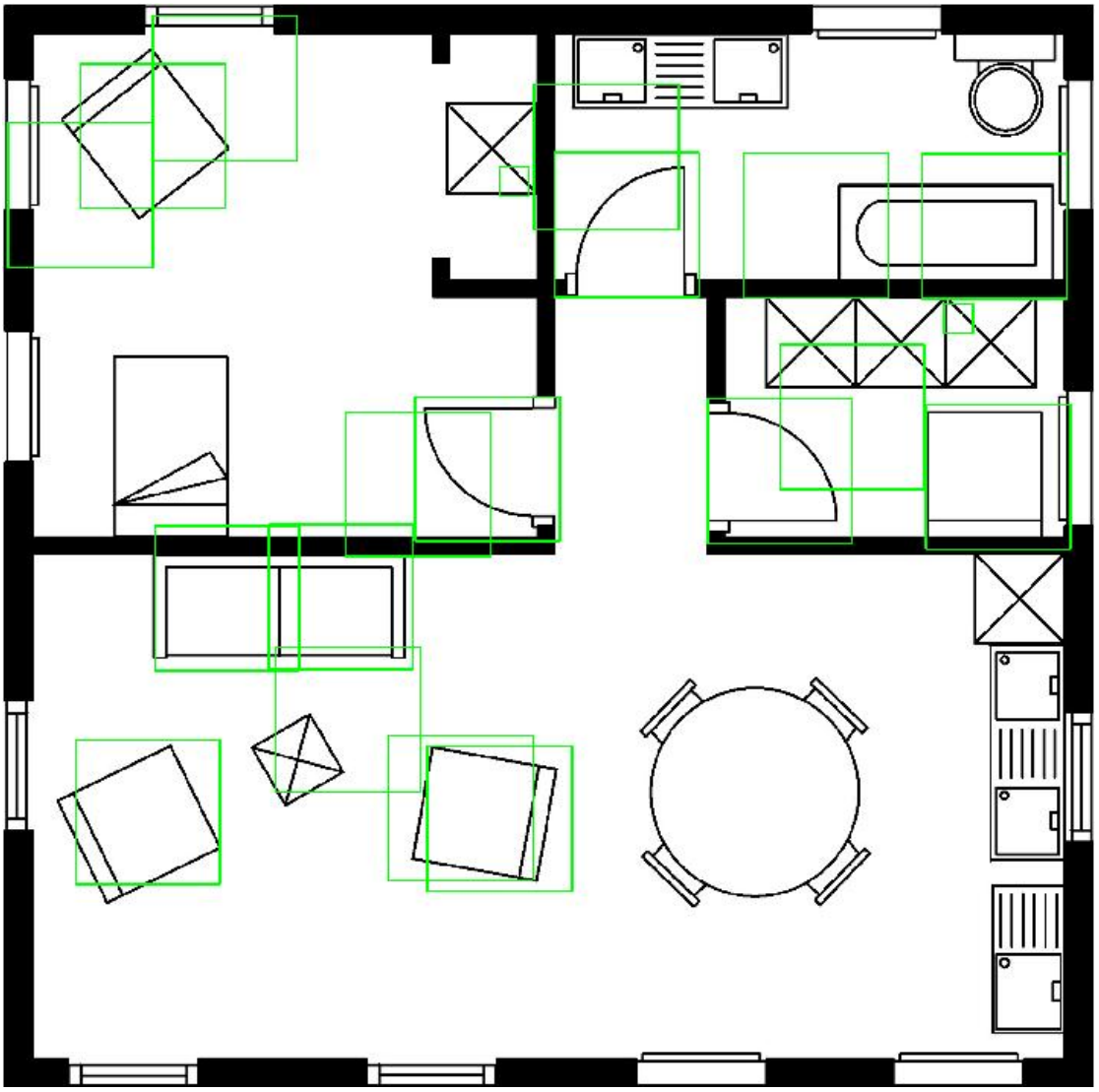
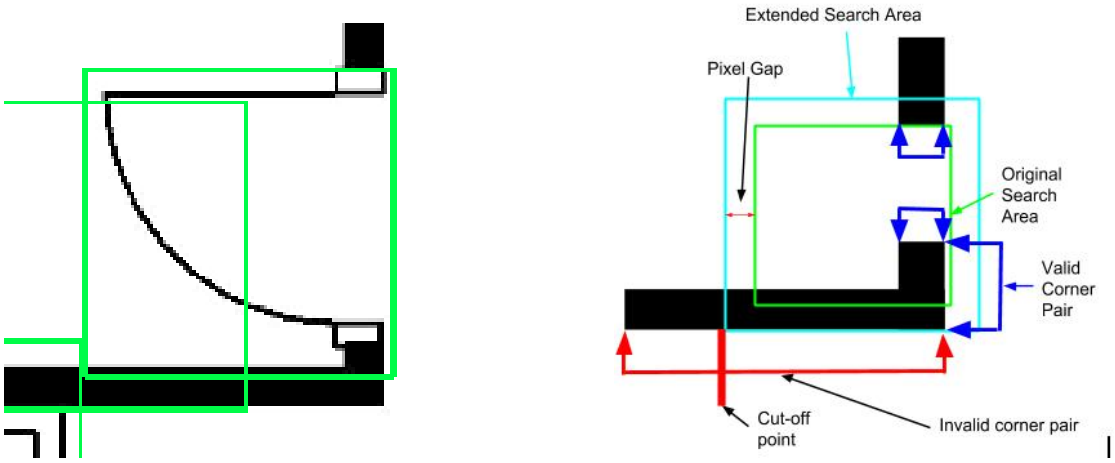


FIGURE 5.15: Haar Cascade object detection for doors

The first thing we do with the green area that denotes the found object is extend its area. Main reason for this is to increase the likely-hood of overlapping the connecting walls.

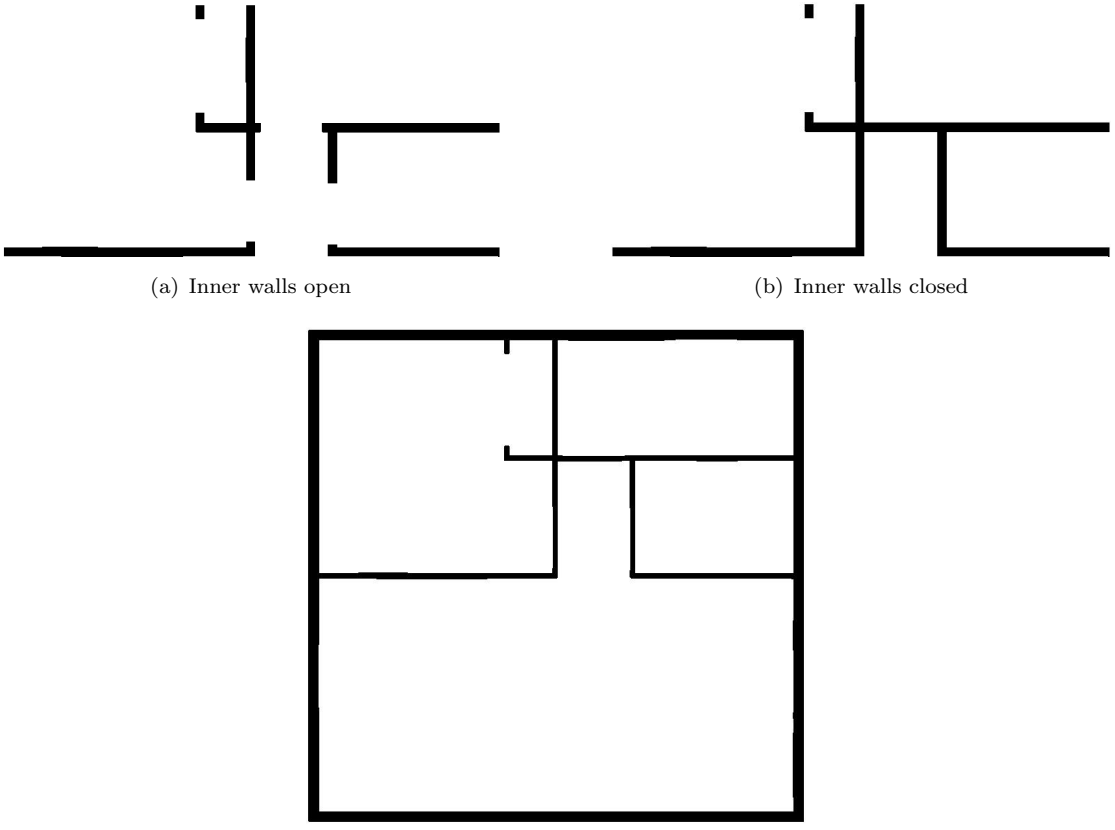
The next step is to find valid corner pairs (or single corner of a pair) that overlap with the extended search area. Creating the closure area rectangle works as previously described process for outer wall closure areas.

The resulting closing areas 5.17(a),5.17(b) just like previously can both used to locate doors and be drawn onto the floor map completing the prerequisite for room detection 5.17(c).



(a) Door area found (b) Analysis of door area

FIGURE 5.16: Analysis of a single object detection result



(a) Inner walls open (b) Inner walls closed

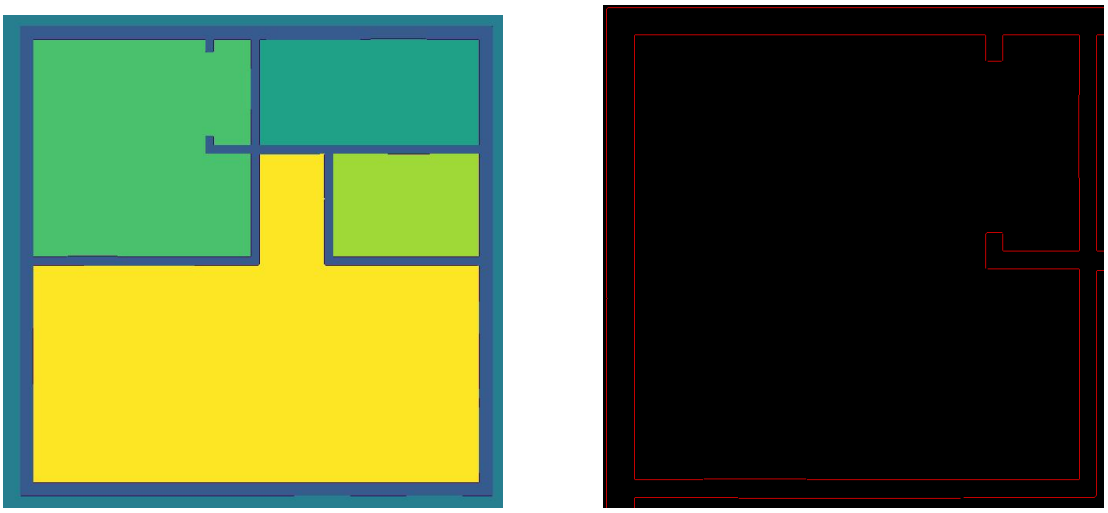
(c) All walls closed

FIGURE 5.17: Final wall closures

5.2.3.2 Room Detection

Room detection is accomplished through the use of distance transform 4.1.4.1 and watershed algorithm. The watershed algorithm basically allows each of the colored regions

5.18(a) to be 'watered' down to their border 5.18(b). This process essentially allows us to segment each room. Unfortunate this data is yet to be translated to usable data for reconstruction



(a) Distance transform

(b) Watershed applied to distance transform image

FIGURE 5.18: Room detection algorithm

5.3 Development: Reconstruction Stage

In order to render in the browser the library three.js was used. Three.js is a free 3D library for JavaScript 4.1.1.2.

Now that data is structured and exported, reconstruction becomes relatively easy. As of this point in the project, the web renderer requires three JSON files: walls, windows and doors. This design choice was made to remove the need to build complex JSON files and allow for rapid testing.

Each individual wall segment is built simply by reading the structured array of corner points from the JSON file. For example a corner at pixel 123, 41 in the image is directly mapped to a corner at that point in the grid. Because of this feature in three.js, the scale and position of all elements in the architectural floor map image remains the same in the reconstructed model. Currently all object from all three JSON files are rendered as 'walls' rather than custom models, this includes doors as seen here 5.19.

Source code can be found on GitHub repository [25]

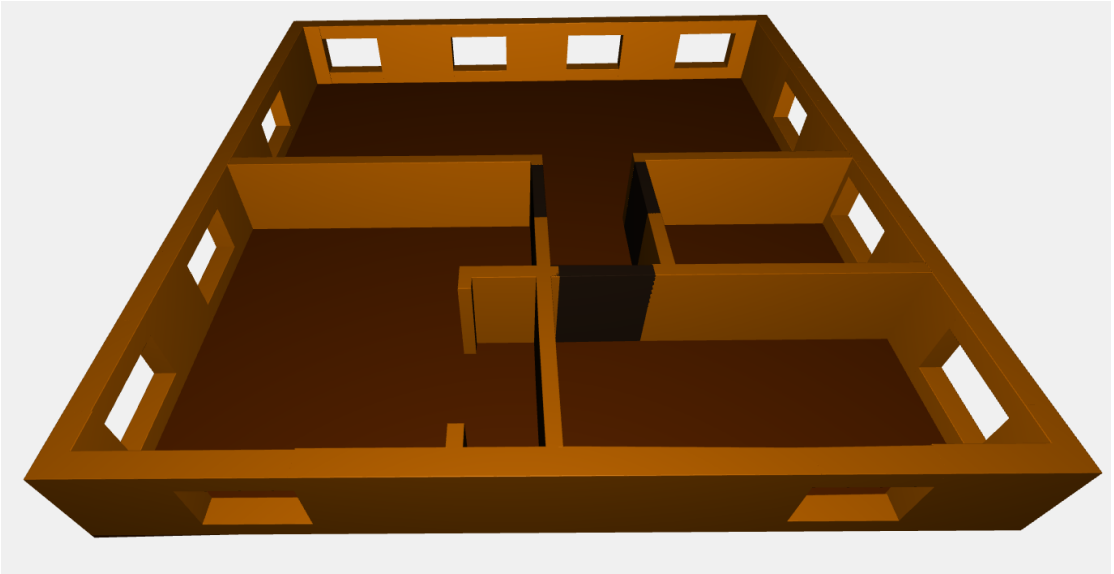


FIGURE 5.19: Rendering 3D model from JSON

5.4 Verification

Despite the analysis stage being divided into a number of easily defined steps. With each step of the process having a clearly defined input and output. The success or failure of each task is very abstract, making testing an extremely daunting task. A lot of information is hidden and must be inferred by the tester in order to rate how successful a step is. Even for a human, defining the success of a step very difficult. Most data is composed of large and complex multidimensional arrays that encode data found in each pixel. Only through rendering and comparing results by human is it possible to define the success or failure of a step. This makes any attempt at automated testing futile, as any test that are written are made irrelevant if any changes occur in an image.

Chapter 6

Conclusions and Future Work

6.1 Solution Review

System overall is divided into a number of logically independent steps, making each step susceptible to being changed as long as output remains the same or similar. The current state of the project works in rather limited capacity, due to many limiting factors, it can only, at best serve as a proof of concept.

Much of the functionality can only work in very specific environments due to certain design choices that were made as the system was developed. However, the design choices that were made, are meant to be as inclusive as possible. Meaning that the ground work for most steps were conceptualized to work for multiple visual iterations of an architectural plan.

6.2 Project Review

Initially the approach to the project was simple. Implement a single feature and test until I could visually confirm that the feature was working as intended. The goal was to use a single simplified floor plan throughout the whole development cycle of the project until the success was achieved.

The approach worked quite well with the project, due to the system having a clear beginning and end. It allowed each step to be developed in the limited time frame that was allocated to it. Usually architectural floor plans are complex and may have unique features, that would require a lot of time to engineer. This approach allowed to skim over many of these feature, but in contrast it severely limited the reliability of each step.

For example, one major assumption that was made, was to consider that an outer wall could only have two connecting sides. This limit was placed due to the complexity it would introduced otherwise. Therefore, if any third connecting wall of the same thickness was introduced it would either remain open or other unintended connection would be made. This would break all the following steps. At this point in time there is no solution to this problem.

Many of these limitations have been placed on the project, mainly due to me being unfamiliar with computer vision field overall. Another reason is that, developing an algorithm that solves a specific problem requires a lot of time. Some of the algorithms in this project took dozens grueling hours of testing to develop. Robustness was key, even if the problem solved has minimal scope.

Unfortunately, unlike papers that mention that after analysis it would not require much effort to build a 3D model. It is not the case. Many small details and nuances that these papers do not mention can become a critical factor that determines how successful a translation is. If a corner or object is not detected, or it is detected but a few pixel off, it can cause glaring visual issues when data is translated to 3D. This was often the case for me.

An, example of this was the initial translation of walls to 3D. What was not considered in analysis, was that an array of corner coordinates has no meaning when translated. If done directly it forms a jumbled up object that is completely unusable. Therefore a complex process of data analysis must occur to restructure the array so it makes sense. This complexity was not calculated into the schedule.

Human factors also payed a key role in getting the project to its current state. This underestimation in workload severely impeded progress. Often having progress blocked either due to lack of knowledge in certain field or inability to solve a certain problem can be a cause for lack of motivation. This can lead to procrastination and a general lack of work ethic. This led to certain requirements not being implemented or implemented poorly.

To summarize, the project contained too many unknowns and too broad in scope to be a viable final year project. If done again, the first step would be to reduce the scope of the project. The most optimal choice would be to stick to just analysis stage and avoid reconstruction. This should be a two stage project, allowing for deeper analysis pertaining each step to produce better quality results.

6.3 Key Skills

- Deeper understanding of computer vision algorithms, their uses in certain use cases. This provides me some confidence in developing tools that depend on visual analysis of a scene.
- Better understanding and ability to estimate tasks, pertaining to computer vision.
- Improved skills in languages such as python and JavaScript.
- Working with frameworks such as OpenCV and Three.js

6.4 Future Work

- Of the changes to be made first and foremost would be to redesign the segmentation process. This is the first and most important task. If done incorrectly, it can form critical problems that produce incorrect results. One of the suggested ways of improving this, is done through neural networks.
- Currently many algorithms work through brute force. Redesigning these should be another task, especially so if the solution is to be hosted on the cloud as a service.
- Improve algorithms to include as many edge cases as possible.
- Improve the 3D visual rendering of floor plan.
- Possibly change the object detection algorithm.
- Build a large library of symbolic objects for training object detection algorithm.
- Move the service to work in the cloud.
- Provide an easy system to integrate the rendering software on a web-page.
- Explore the possibility of performing the analysis stage in JavaScript, allowing for a server free solution.
- Provide a 3rd party way of permanently changing the results of the floor plan.
- Implement a larger array of functionality to the web application.

6.5 Conclusion

Throughout the research phase I found that currently there is no publicly available process to accomplish the task set for this project. There is ether not enough demand for the process or no reliable way of doing it, and I believe its a mix of both. There is certainly demand for 3D dimensional floor plans, as there is a plethora of different services online providing this service. However, there are none that have automated this process, as it is all done by hand of an artist. There have been a number of attempts in the last two decades; most of them succeeding to some extent, promising to work on improving the process. However, so far none have done so, at least not to my knowledge.

Throughout the development cycle for this project I have found some success in streamlining the process of conversion. However, the unpredicted growth in complexity of requirements caused a variate of issues. This lead to requirements not being met or done so in poor quality. I have found deeper understanding as to why this task has remained unaccomplished by many research teams over the last two decades. Overall, this has been learning experience that has showed me ups and downs of taking a project of this scale.

Bibliography

[1] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[2] S. Ahmed, M. Liwicki, M. Weber, and A. Dengel, "Improved automatic analysis of architectural floor plans," *2011 International Conference on Document Analysis and Recognition*, 2011.

[3] (2017) morphological image processing. [Online]. Available: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm#basic

[4] (2017) Opencv library. [Online]. Available: https://opencv.org/

[5] (2017) Tiobe index. [Online]. Available: https://www.tiobe.com/tiobe-index/

[6] (2017). [Online]. Available: https://madnight.github.io/githut/

[7] S. Ahmed, M. Liwicki, M. Weber, and A. Dengel, "Automatic room detection and room labeling from architectural floor plans," in *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on.* IEEE, 2012, pp. 339–343.

[8] S. Mac, H. Locteau, E. Valveny, and S. Tabbone, "A system to detect rooms in architectural floor plan images," *Proceedings of the 8th IAPR International Workshop on Document Analysis Systems - DAS '10*, 2010.

[9] S.-h. Or, K.-h. Wong, Y.-k. Yu, and M. M.-y. Chang, "Abstract highly automatic approach to architectural floorplan image understanding & model generation," *Pattern Recognition*, 2008.

[10] S. Ahmed, M. Weber, M. Liwicki, and A. Dengel, "Text/graphics segmentation in architectural floor plans," in *Document Analysis and Recognition (ICDAR), 2011 International Conference on.* IEEE, 2011, pp. 734–738.

[11] R. Tang, Y. Wang, D. Cosker, and W. Li, "Automatic structural scene digitalization," *PLOS ONE*, vol. 12, no. 11, p. e0187513, 2017.

[12] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[13] J. E. Solem, *Programming Computer Vision with Python: Tools and algorithms for analyzing images*. " O'Reilly Media, Inc.", 2012.

[14] A. Mordvintsev and K. Abid, "Opencv-python tutorials documentation," *Obtenido de https://media. readthedocs. org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals. pdf*, 2014.

[15] sentdex, YouTube, 2018. [Online]. Available: https://www.youtube.com/user/sentdex

[16] 2018. [Online]. Available: https://mathematica.stackexchange.com

[17] 2018. [Online]. Available: http://answers.opencv.org/questions/

[18] P. Dosch and G. Masini, "Reconstruction of the 3d structure of a building from the 2d drawings of its floors," in *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*. IEEE, 1999, pp. 487–490.

[19] Y. Aoki, A. Shio, H. Arai, and K. Odaka, "A prototype system for interpreting hand-sketched floor plans," in *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, vol. 3. IEEE, 1996, pp. 747–751.

[20] T. Lu, H. Yang, R. Yang, and S. Cai, "Automatic analysis and integration of architectural drawings," *International Journal on Document Analysis and Recognition*, vol. 9, no. 1, pp. 31–47, 2007.

[21] (2017) Javascript 3d library. [Online]. Available: https://threejs.org/

[22] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[23] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Manchester, UK, 1988, pp. 10–5244.

[24] R. L. Graham and F. F. Yao, "Finding the convex hull of a simple polygon," *Journal of Algorithms*, vol. 4, no. 4, pp. 324–331, 1983.

[25] V. Valiusis, "Automated three-dimensional translation of floor plans," https://github.com/vilius-valiusis/floor-map-to-3D, 2018.

Appendix A

Code Snippets

```
import matplotlib.pyplot as plt
from algorithms import wall_analysis, morphological as morph,
    room_detection as rd
import cv2
import json
import numpy as np
import haar_cascade.classifier as cls

image_outer_walls = []
image_outer_walls_closed = []
image_outer_walls_contours = []

image_inner_walls = []
image_inner_walls_closed = []
image_inner_walls_contours = []

image_inner_outer_walls = []

outer_wall_contours = []
outer_wall_corners = []
outer_wall_closure_areas = []

inner_wall_contours = []
inner_wall_corners = []
inner_wall_closure_areas = []


class MyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.integer):
            return int(obj)
        elif isinstance(obj, np.floating):
```

```
32              return float(obj)
33          elif isinstance(obj, np.ndarray):
34              return obj.tolist()
35          else:
36              return super(MyEncoder, self).default(obj)
37
38
39  def export_json_list(walls, file_name):
40      file = open(file_name+'.json', 'w')
41      file.write(json.dumps(walls, cls=MyEncoder))
42      file.close()
43
44
45  def show_images(img1, img2):
46      plt.subplot(121), plt.imshow(img1)
47      plt.subplot(122), plt.imshow(img2)
48      plt.show()
49
50
51  def show_image(img1):
52      plt.imshow(img1, cmap='gray')
53      plt.show()
54
55
56  def extract_outer_wall_image(thresh_img):
57      return morph.morph_operation(thresh_img, 8, 8, 8)
58
59
60  def extract_inner_outer_walls(thresh_img):
61      return morph.morph_operation(thresh_img, 3, 3, 3)
62
63
64  def extract_inner_wall_image(thresh_img):
65      outer = extract_outer_wall_image(thresh_img)
66      inner = extract_inner_outer_walls(thresh_img)
67      inner_only = cv2.subtract(outer, inner)
68      cv2.bitwise_not(inner_only, inner_only)
69      return inner_only
70
71
72  def extract_outer_wall_closure_areas():
73      return wall_analysis.extract_closure_areas(image_outer_walls_contours,
        outer_wall_contours, 'outer')
74
75
76  def extract_inner_wall_closure_areas(door_objects):
77      return wall_analysis.extract_closure_areas(image_inner_walls_contours,
        inner_wall_contours, door_objects, 'inner')
```

```
78
79
80 def extract_wall_contours(image):
81     return wall_analysis.extract_wall_contours(image)
82
83
84 def extract_corners(contours_image, contours):
85     return wall_analysis.extract_corners_from_contours(contours_image,
       contours)
86
87
88 if __name__ == "__main__":
89     # Read in the image as a gray scale image
90     img = cv2.imread("floor_maps//test1.jpg", 0)
91
92     # Binerise the image
93     ret, thresh = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.
       THRESH_OTSU)
94
95     # Segment inner and outer walls
96     image_outer_walls = extract_outer_wall_image(thresh)
97     image_inner_walls = extract_inner_wall_image(thresh)
98     image_inner_outer_walls = extract_inner_outer_walls(thresh)
99
100    # Find inner and outer wall contours
101    outer_wall_contours, image_outer_walls_contours = extract_wall_contours
       (image_outer_walls)
102    inner_wall_contours, image_inner_walls_contours = extract_wall_contours
       (image_inner_walls)
103
104    # Find inner and outer wall corners
105    outer_wall_corners = extract_corners(image_outer_walls_contours,
       outer_wall_contours)
106    inner_wall_corners = extract_corners(image_inner_walls_contours,
       inner_wall_contours)
107
108    # Find outer wall closure areas
109    outer_wall_closure_areas = extract_outer_wall_closure_areas()
110
111    # Find inner wall closure areas
112    door_cascade = cv2.CascadeClassifier('haar_cascade//
       door_classifier_40x40.xml')
113    door_objects = cls.find_objects(img, door_cascade, 5, 1)
114    inner_wall_closure_areas = extract_inner_wall_closure_areas(
       door_objects)
115
116    # Close inner and outer areas
```

```
117     wall_analysis.draw_closure_areas(image_inner_outer_walls,
        outer_wall_closure_areas)
118     wall_analysis.draw_closure_areas(image_inner_outer_walls,
        inner_wall_closure_areas)
119
120     # Extract room contours
121     extracted_rooms_contours, markers = rd.extract_rooms(
        image_inner_outer_walls)
122
123     # Append inner and outer walls to one array
124     for val in inner_wall_corners:
125         outer_wall_corners.append(val)
126
127     # Export values as JSON
128     export_json_list(inner_wall_closure_areas, "output/doors")
129     export_json_list(outer_wall_closure_areas, "output/windows")
130     export_json_list(outer_wall_corners, "output/walls")
```

Listing A.1: Analysis Stage: Main

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # Distance between corners that is define weather the corners
6  # are to be considered to be on the same line.
7  PIX_RANGE = 5
8  INNER_WALL_PIX_RANGE = 10
9  wall_contours = []
10 wall_contour_images = []
11 wall_closure_areas = []
12
13
14 ##################################################
15 # Show image for test purposes
16 ##################################################
17 def show_images(img):
18     plt.imshow(img, cmap='gray')
19     plt.show()
20
21
22 ##################################################
23 # Draws a contour on a blank canvas
24 ##################################################
25 def draw_wall_contour(img, contour):
26     canvas = np.zeros(img.shape, np.uint8)
27     cv2.drawContours(canvas, contour, -1, (123, 125, 245), 1)
28     return canvas
```

```
29
30
31 ################################################
32 # Extract wall contours from the provided image
33 ################################################
34 def extract_wall_contours(img):
35     canvas = np.zeros(img.shape, np.uint8)
36     im2, contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.
       CHAIN_APPROX_SIMPLE)
37     cv2.drawContours(canvas, contours, -1, (123, 125, 245), 1)
38     return contours, canvas
39
40
41 #
       ##############################################################################################

42 # Uses Harris algorithm to detect corners on the provided image.
43 # Each corner is then detected and roundedc
44 #
       ##############################################################################################

45 def extract_wall_corners_sub(img):
46     gray = np.float32(img)
47     dst = cv2.cornerHarris(gray, 2, 3, 0.04)
48     dst = cv2.dilate(dst, None)
49     ret, dst = cv2.threshold(dst, 0.001 * dst.max(), 255, 0)
50     dst = np.uint8(dst)
51
52     # find centroids
53     ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)
54
55     # for corner in np.int0(centroids):
56     #     x, y = corner
57     #     cv2.circle(img, (x, y), 2, 255, -1)
58     # show_images(img)
59     return img, np.int0(centroids)
60
61
62 #
       ##############################################################################################

63 # Helper function to check if a pixel is within a range of another pixel,
       using the maximum
64 # pixel variance e.g. PIX_RANGE
65 #
       ##############################################################################################

66 def is_in_range(a, b):
```

```
67      return a + PIX_RANGE > b > a - PIX_RANGE
68
69
70  #
        ##########################################################################################
71  # Checks for the direction of the corner
72  # So if a corner is 'L' shaped it will return top = True and right = True
73  #
        ##########################################################################################
74  def check_direction(corner, contour_img):
75      top, right, bottom, left = False, False, False, False
76      y, x = corner
77
78      for i, v in enumerate(reversed(contour_img[x][:y])):
79          if 3 < i < 8:
80              for r in range(-5, 5):
81                  if contour_img[x + r][y - i] == 123:
82                      left = True
83                  if contour_img[x + r][y + i] == 123:
84                      right = True
85                  if contour_img[x + i][y + r] == 123:
86                      bottom = True
87                  if contour_img[x - i][y + r] == 123:
88                      top = True
89
90      return top, right, bottom, left
91
92
93  #
        ##########################################################################################
94  # Finds the next direction/axis the corner should be on depending on the
        previous corners.
95  # Requires minimum of two sequential corners.
96  # 0 = top, 1 = right, 2 = bottom, 3 = left
97  #
        ##########################################################################################
98  def get_next_direction(corners, contour_img, last_direction):
99      current_corner = corners[-1]
100     last_corner = corners[-2]
101     t1, r1, b1, l1 = check_direction(last_corner, contour_img)
102     t2, r2, b2, l2 = check_direction(current_corner, contour_img)
103
104     if (t1 and b2) and (t2 and b1) and is_in_range(current_corner[0],
        last_corner[0]):
```

```
105         return last_direction
106
107     if (t1 and b2) and is_in_range(current_corner[0], last_corner[0]):
108         if t2:
109             return 0
110         if r2:
111             return 1
112         if l2:
113             return 3
114
115     if (t2 and b1) and is_in_range(current_corner[0], last_corner[0]):
116         if r2:
117             return 1
118         if b2:
119             return 2
120         if l2:
121             return 3
122
123     if (r1 and l2) and (r2 and l1) and is_in_range(current_corner[1],
    last_corner[1]):
124         return last_direction
125
126     if (r1 and l2) and is_in_range(current_corner[1], last_corner[1]):
127         if t2:
128             return 0
129         if r2:
130             return 1
131         if b2:
132             return 2
133
134     if (r2 and l1) and is_in_range(current_corner[1], last_corner[1]):
135         if t2:
136             return 0
137         if b2:
138             return 2
139         if l2:
140             return 3
141
142
143 ######################################################################
144 # Walk the shortest distance from x->x or y->y on the same line
145 ######################################################################
146 def walk_shortest_corners(corners, contour_img):
147     rearranged_corners = []
148     last_direction = 0
149     for i1, corner1 in enumerate(corners):
150         y1, x1 = corner1
151         if i1 == 0:
```

```
152             continue
153         elif i1 == 1 or i1 == 2:
154             rearranged_corners.append((y1, x1))
155         else:
156             # x-current, x-last, y-current, y-last
157             xc, xl, yc, yl = (0, 0), (0, 0), (0, 0), (0, 0)
158             last_added = rearranged_corners[-1]
159             next_direction = get_next_direction(rearranged_corners,
    contour_img, last_direction)
160 
161             # Loop through all the corners to find the next one
162             for i2, corner2 in enumerate(corners):
163                 y2, x2 = corner2
164                 # If a corner was previously added
165                 if i2 == 0 or (y2, x2) in rearranged_corners:
166                     continue
167                 # Filters any values that do no follow the direction of the
     contour
168                 # from the previous corner.
169                 if next_direction == 0:
170                     if last_added[1] < x2:
171                         continue
172                 if next_direction == 1:
173                     if last_added[0] > y2:
174                         continue
175                 if next_direction == 2:
176                     if last_added[1] > x2:
177                         continue
178                 if next_direction == 3:
179                     if last_added[0] < y2:
180                         continue
181 
182                 # Checks if y coordinate is in range of the last added
    corner
183                 # If it is it will try to find the smallest/largest
    possible value
184                 # depending on the direction 2 = down, 0 = up
185                 if is_in_range(y2, last_added[0]):
186                     if next_direction == 2:
187                         if yc[1] <= yl[1]:
188                             yl = yc
189                             yc = (y2, x2)
190                     if next_direction == 0:
191                         if yc[1] >= yl[1]:
192                             yl = yc
193                             yc = (y2, x2)
194                 # Checks if x coordinate is in range of the last added
    corner
```

```
195                 # If it is it will try to find the smallest/largest
    possible value
196                 # depending on the direction 1 = right, 3 = left
197                 if is_in_range(x2, last_added[1]):
198                     if next_direction == 1:
199                         if xc[0] <= xl[0]:
200                             xl = xc
201                             xc = (y2, x2)
202                     if next_direction == 3:
203                         if xc[0] >= xl[0]:
204                             xl = xc
205                             xc = (y2, x2)
206
207             # Add value to corner array
208             if yc == (0, 0) and xc != (0, 0):
209                 rearranged_corners.append(xc)
210                 last_direction = next_direction
211             elif xc == (0, 0) and yc != (0, 0):
212                 rearranged_corners.append(yc)
213                 last_direction = next_direction
214
215     return readjust_corner_coordinates(rearranged_corners)
216
217
218 #
    ##############################################################################

219 # Due some corner pixel on the same line may vary
220 # This step is done to offset this variance by following the previous line
221 #
    ##############################################################################

222 def readjust_corner_coordinates(corners):
223     new_corners = []
224     for i, corner in enumerate(corners):
225         x, y = corner
226         if i == 0:
227             new_corners.append(corner)
228             continue
229         elif is_in_range(x, corners[i - 1][0]):
230             new_corners.append((corners[i - 1][0], y))
231         elif is_in_range(y, corners[i - 1][1]):
232             new_corners.append((x, corners[i - 1][1]))
233     return new_corners
234
235
```

```
236 #
    ##############################################################################

237 # Checks for illegal structures such as ones that contain coordinates like
    (1,1)
238 #
    ##############################################################################

239 def is_wall_a_legal_structure(corners):
240     for corner in corners:
241         x, y = corner
242         if x == 1 or y == 1:
243             return False
244     return True
245
246
247 def is_corner_pair_usable(c1: tuple, c2: tuple, axis):
248     # axis = x = 0, y = 1
249     if c1[1] and c1[2] and c2[2] and c2[3] and axis == 1:
250         return True
251     elif c1[3] and c1[2] and c2[0] and c2[3] and axis == 0:
252         return True
253     elif c1[3] and c1[0] and c2[0] and c2[1] and axis == 1:
254         return True
255     elif c1[1] and c1[2] and c2[0] and c2[1] and axis == 0:
256         return True
257     else:
258         return False
259
260
261 def find_distance_between_corners(c1, c2):
262     if is_in_range(c1[0], c2[0]):
263         return abs(c1[1] - c2[1])
264     else:
265         return abs(c1[0] - c2[0])
266
267
268 def find_shortest_pairs(corner_pairs, limit):
269     if limit != 0:
270         sorted_cps = sorted(corner_pairs, key=lambda corner_pair:
    corner_pair[1])[:limit]
271     else:
272         sorted_cps = sorted(corner_pairs, key=lambda corner_pair:
    corner_pair[1])
273     return sorted_cps
274
275
276 def find_viable_corner_pairs(wall, contour_img):
```

```
277     corner_pairs = []
278     for corner in wall:
279         x, y = corner
280         t1, r1, b1, l1 = check_direction(corner, contour_img)
281         corners1 = (t1, r1, b1, l1)
282         for corner2 in wall:
283             x2, y2 = corner2
284             x_in_range = is_in_range(x, x2)
285             y_in_range = is_in_range(y, y2)
286             if x_in_range or y_in_range:
287                 t2, r2, b2, l2 = check_direction(corner2, contour_img)
288                 corners2 = (t2, r2, b2, l2)
289                 axis = 0 if x_in_range else 1
290                 # print(corner, corner2, is_corner_pair_usable(corners1,
    corners2, axis))
291                 if is_corner_pair_usable(corners1, corners2, axis):
292 
293                     distance = find_distance_between_corners((x, y), (x2,
    y2))
294                     # print(corner, corner2, distance)
295                     corner_pairs.append((((x, y), (x2, y2)), distance, axis
    ))
296     return corner_pairs
297 
298 
299 def find_outer_wall_closure_areas(corner_pairs):
300     closure_areas = []
301     closest_corner = ((0, 0), (0, 0)), 0
302     # Corner_pair contains an array of x2 consisting of 2 corners of a wall
    , distance and the axis
303     # axis 0=x, 1=y
304     # Loop through all corner pairs
305     for i, cpd in enumerate(corner_pairs):
306         for cp in cpd:
307 
308             for ii, cpd2 in enumerate(corner_pairs):
309                 for cp2 in cpd2:
310                     # If corners are on the same axis AND and have the same
     width or height AND are not the same corners
311                     if cp[2] == cp2[2] and is_in_range(cp[1], cp2[1]) and
    cp != cp2:
312                         if i == ii:
313                             continue
314                         # If corner is on x axis AND
315                         if cp[2] == 0:
316 
317                             if is_in_range(cp[0][0][1], cp2[0][0][1]):
```

```
318                                 distance = find_distance_between_corners(cp
    [0][0], cp2[0][0])
319                             elif is_in_range(cp[0][0][1], cp2[0][1][1]):
320                                 distance = find_distance_between_corners(cp
    [0][0], cp2[0][1])
321
322                             if distance < closest_corner[1] or
    closest_corner[1] == 0:
323                                 closest_corner = ((cp2[0][0], cp2[0][1]),
    distance)
324
325                         elif cp[2] == 1:
326
327                             if is_in_range(cp[0][0][0], cp2[0][0][0]):
328                                 distance = find_distance_between_corners(cp
    [0][0], cp2[0][0])
329                             elif is_in_range(cp[0][0][0], cp2[0][1][0]):
330                                 distance = find_distance_between_corners(cp
    [0][0], cp2[0][1])
331
332                             if distance < closest_corner[1] or
    closest_corner[1] == 0:
333                                 closest_corner = ((cp2[0][0], cp2[0][1]),
    distance)
334
335             closure_areas.append([cp[0][0], cp[0][1], closest_corner[0][0],
     closest_corner[0][1]])
336             closest_corner = ((0, 0), (0, 0)), 0
337             remove_duplicate_closure_areas(closure_areas)
338     return closure_areas
339
340
341 def find_inner_wall_closure_areas(found_objects, corner_pairs):
342     viable_closure_pairs = []
343     for obj in found_objects:
344         x, y, w, h = obj
345         min_coord = (x-PIX_RANGE, y-PIX_RANGE)
346         max_coord = (x+w+PIX_RANGE, y+h+PIX_RANGE)
347         areas = []
348         for cpd in corner_pairs:
349             for cps in cpd:
350                 for cp in cps[0]:
351                     if cp[0] >= min_coord[0] and cp[1] >= min_coord[1] and
    cp[0] <= max_coord[0] and cp[1] <= max_coord[1]\
352                             and cps[1] < w:
353
354                         if [(cps[0][0], cps[0][1]), cps[1], cps[2]] not in
    areas \
```

```
355                                 and [(cps[0][1], cps[0][0]), cps[1], cps
    [2]] not in areas:
356                             areas.append([cps[0], cps[1], cps[2]])
357
358         if 2 <= len(areas):
359             viable_closure_pairs.append(areas)
360
361     closure_areas = []
362     closest_corner = ((0, 0), (0, 0)), 0
363
364     for i, pairs1 in enumerate(viable_closure_pairs):
365         for pair1 in pairs1:
366
367             for ii, pair2 in enumerate(viable_closure_pairs[i]):
368                 if is_in_range(pair1[1], pair2[1]) and pair1[2] == pair2[2]
     and pair1 != pair2:
369
370                     if pair1[2] == 0:
371                         if is_in_range(pair1[0][0][1], pair2[0][0][1]):
372                             distance = find_distance_between_corners(pair1
    [0][0], pair2[0][0])
373                         elif is_in_range(pair1[0][0][1], pair2[0][1][1]):
374                             distance = find_distance_between_corners(pair1
    [0][0], pair2[0][1])
375
376                         if distance < closest_corner[1] or closest_corner
    [1] == 0:
377                             closest_corner = ((pair2[0][0], pair2[0][1]),
    distance)
378
379                     elif pair1[2] == 1:
380                         if is_in_range(pair1[0][0][0], pair2[0][0][0]):
381                             distance = find_distance_between_corners(pair1
    [0][0], pair2[0][0])
382                         elif is_in_range(pair1[0][0][0], pair2[0][1][0]):
383                             distance = find_distance_between_corners(pair1
    [0][0], pair2[0][1])
384
385                         if distance < closest_corner[1] or closest_corner
    [1] == 0:
386                             closest_corner = ((pair2[0][0], pair2[0][1]),
    distance)
387
388                     closure_areas.append([pair1[0][0], pair1[0][1],
    closest_corner[0][0], closest_corner[0][1]])
389                     closest_corner = ((0, 0), (0, 0)), 0
390                     remove_duplicate_closure_areas(closure_areas)
391
```

```
392     return closure_areas
393
394
395 def remove_duplicate_closure_areas(closure_areas):
396     for area in closure_areas:
397         for area2 in closure_areas:
398             if area[0] in area2 and area != area2:
399                 closure_areas.remove(area2)
400
401
402 def draw_closure_areas(img, closure_areas):
403     for area in closure_areas:
404         if is_in_range(area[0][0], area[2][0]):
405             cv2.rectangle(img, area[0], area[3], (0, 0, 0), -1)
406         elif is_in_range(area[0][0], area[3][0]):
407             cv2.rectangle(img, area[0], area[2], (0, 0, 0), -1)
408         elif is_in_range(area[0][1], area[2][1]):
409             cv2.rectangle(img, area[0], area[3], (0, 0, 0), -1)
410         elif is_in_range(area[0][1], area[3][1]):
411             cv2.rectangle(img, area[0], area[2], (0, 0, 0), -1)
412
413
414 def build_wall_contour_image_list(edges_img, contours):
415     for contour in contours:
416         wall_contour_images.append(draw_wall_contour(edges_img, [contour]))
417
418
419 def readjust_closure_areas(closure_areas):
420     readjusted_closure_areas = []
421     for area in closure_areas:
422         if is_in_range(area[1][0], area[2][0]) or is_in_range(area[1][1],
    area[2][1]):
423             readjusted_closure_areas.append([area[0], area[1], area[2],
    area[3]])
424         else:
425             readjusted_closure_areas.append([area[0], area[1], area[3],
    area[2]])
426     return readjusted_closure_areas
427
428
429 ########################################
430 # For each contour extract all corners
431 ########################################
432 def extract_corners_from_contours(edges_img, contours):
433     new_corners = []
434     wall_contour_images.clear()
435     build_wall_contour_image_list(edges_img, contours)
436
```

```
437     for i, img in enumerate(wall_contour_images):
438         contour_img, corners = extract_wall_corners_sub(img)
439         if is_wall_a_legal_structure(corners):
440             new_corners.append(walk_shortest_corners(corners, contour_img))
441     return new_corners
442
443
444 def extract_closure_areas(edges_img, contours, objects=None, wall_position=
    'outer'):
445     corner_pairs = []
446     max_corner_pairs = 2
447     wall_contour_images.clear()
448
449     if wall_position == 'inner':
450         max_corner_pairs = 0
451
452     build_wall_contour_image_list(edges_img, contours)
453
454     for i, img in enumerate(wall_contour_images):
455         contour_img, corners = extract_wall_corners_sub(img)
456         if is_wall_a_legal_structure(corners):
457             cp = find_viable_corner_pairs(corners, contour_img)
458             scp = find_shortest_pairs(cp, max_corner_pairs)
459             corner_pairs.append(scp)
460
461     if wall_position == 'inner':
462         closure_areas = find_inner_wall_closure_areas(objects, corner_pairs
    )
463     else:
464         closure_areas = find_outer_wall_closure_areas(corner_pairs)
465
466     return readjust_closure_areas(closure_areas)
```

Listing A.2: Analysis Stage: Wall analysis algorithms

```
1 import cv2
2
3 defaultKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
4
5
6 def morph_operation(img, erosion_iter=0, opening_iter=0, dilation_iter=0,
    kernel=defaultKernel):
7     erosion = cv2.dilate(img, kernel, iterations=erosion_iter)
8     opening = cv2.morphologyEx(erosion, cv2.MORPH_OPEN, kernel, iterations=
    opening_iter)
9     dilation = cv2.erode(opening, kernel, iterations=dilation_iter)
10    return dilation
```

Listing A.3: Analysis Stage: Morphological Algorithms

```
1
2 import cv2
3
4
5 def find_objects(img, classifier, scale_factor, min_neighbors):
6     objects = classifier.detectMultiScale(img, scaleFactor=scale_factor,
      minNeighbors=min_neighbors)
7     return objects
8
9
10 def draw_bounding_rectangles(img, objects):
11     img_copy = img.copy()
12     for (x, y, w, h) in objects:
13         cv2.rectangle(img_copy, (x, y), (x + w, y + h), (0, 255, 0), 2)
14     return img_copy
```

Listing A.4: Analysis Stage: Haar Cascade Classifier

```
1 import numpy as np
2
3 import cv2
4
5
6 def extract_rooms(closed_wall_image):
7     gray_img = closed_wall_image.copy()
8     color_img = cv2.cvtColor(closed_wall_image, cv2.COLOR_GRAY2RGB)
9
10     dist_transform = cv2.distanceTransform(gray_img, cv2.DIST_L2, 3)
11     ret, sure_fg = cv2.threshold(dist_transform, 0.1 * dist_transform.max()
      , 255, 0)
12
13     sure_fg = np.uint8(sure_fg)
14     unknown = cv2.subtract(gray_img, sure_fg)
15     # Marker labelling
16     ret, markers = cv2.connectedComponents(sure_fg)
17     # Add one to all labels so that sure background is not 0, but 1
18     markers = markers + 1
19     # Now, mark the region of unknown with zero
20     markers[unknown == 255] = 0
21     markers = markers.astype('int32')
22
23     markers = cv2.watershed(color_img, markers)
24     canvas = np.zeros(color_img.shape, np.uint8)
25
26     canvas[markers == -1] = [255, 0, 0]
27
28     return canvas,markers
```

LISTING A.5: Analysis Stage: Room detection algorithm

```
<!DOCTYPE html>
<html>
<head>
    <title>My first three.js app</title>
    <style>
        body { margin: 0; }
        canvas { width: 100%; height: 100% }
    </style>
</head>
<body>
    <script src="js/libs/three.js"></script>
    <script src="js/libs/OrbitControls.js"></script>
    <script src="js/main.js"></script>
</body>
</html>
```

LISTING A.6: Reconstruction Stage: index

```
var container, stats;
var camera, scene, renderer, controls;
var group;

init();
animate();
function init() {
    container = document.createElement( 'div' );
    document.body.appendChild( container );
    var info = document.createElement( 'div' );
    info.style.position = 'absolute';
    info.style.top = '10px';
    info.style.width = '100%';
    info.style.textAlign = 'center';
    info.innerHTML = 'Simple procedurally-generated shapes<br/>Drag to spin
    ';
    container.appendChild( info );

    renderer = new THREE.WebGLRenderer({antialias: true});
    renderer.setSize( window.innerWidth, window.innerHeight );
    document.body.appendChild( renderer.domElement );

    scene = new THREE.Scene();
    scene.background = new THREE.Color( 0xf0f0f0 );

    camera = new THREE.PerspectiveCamera( 65, window.innerWidth / window.
    innerHeight, 1, 3000 );
```

```
26      camera.position.set( 0, 100, 1500 );
27      controls = new THREE.OrbitControls(camera)
28      controls.update();
29
30      var light = new THREE.PointLight( 0xffffff, 0.8 );
31
32      group = new THREE.Group();
33      group.position.y = 0;
34
35      camera.add( light );
36      scene.add( group );
37      scene.add( camera );
38
39      var walls = 'input/walls.json';
40      var doors = 'input/doors.json';
41      var windows = 'input/windows.json';
42
43      loadJSON(walls,function(response) {
44          // Parse JSON string into object
45          var actual_JSON = JSON.parse(response);
46          drawWall(actual_JSON)
47      });
48
49      loadJSON(doors,function(response) {
50          // Parse JSON string into object
51          var actual_JSON = JSON.parse(response);
52          drawDoor(actual_JSON)
53      });
54      loadJSON(windows,function(response) {
55          // Parse JSON string into object
56          var actual_JSON = JSON.parse(response);
57          drawWindows(actual_JSON)
58      });
59
60  }
61
62  function drawWall(walls){
63      var extrudeSettings = { amount: 200, bevelEnabled: true, bevelSegments:
        2, steps: 2, bevelSize: 1, bevelThickness: 1 };
64      var floor_extrudeSettings = { amount: 10, bevelEnabled: true,
       bevelSegments: 2, steps: 2, bevelSize: 1, bevelThickness: 1 };
65      var big_y = 0, small_y = 0, big_x = 0, small_x = 0
66
67      for(var i in walls){
68          var wallshape = new THREE.Shape();
69          wallshape.moveTo(walls[i][0][0],walls[i][0][1])
70          for(var x in walls[i]){
71              wallshape.lineTo(walls[i][x][0],walls[i][x][1])
```

```
72                  if( walls[i][x][0] < small_x || small_x == 0){
73                      small_x = walls[i][x][0]
74                  }
75                  if( walls[i][x][1] < small_y || small_y == 0){
76                      small_y = walls[i][x][1]
77                  }
78                  if( walls[i][x][0] > big_x || big_x == 0){
79                      big_x = walls[i][x][0]
80                  }
81                  if( walls[i][x][1] > big_y || big_y == 0){
82                      big_y = walls[i][x][1]
83                  }
84              }
85              addShape( wallshape,extrudeSettings, 0xf08000, -750, -750, 0,
    0, 0, 0, 1);
86      }
87      // Draw a simple all encompassing floor
88      var texture = new THREE.TextureLoader().load( "textures/retina_wood.png
    " );
89      var floor_shape = new THREE.Shape()
90      floor_shape.moveTo(small_x,small_y)
91      floor_shape.lineTo(small_x,big_y)
92      floor_shape.lineTo(big_x,big_y)
93      floor_shape.lineTo(big_x,small_x)
94      addShape( floor_shape,floor_extrudeSettings, 0x6e2c00 , -750, -750,
    -10, 0, 0, 0, 1);
95
96      wallshape.moveTo(walls[i][0][0],walls[i][0][1])
97
98  }
99
100 function drawDoor(doors){
101     var extrudeSettings = { amount: 200, bevelEnabled: true, bevelSegments:
     2, steps: 2, bevelSize: 1, bevelThickness: 1 };
102     for(var i in doors){
103         var doorShape = new THREE.Shape();
104         doorShape.moveTo(doors[i][0][0],doors[i][0][1])
105         for(var x in doors[i]){
106             doorShape.lineTo(doors[i][x][0],doors[i][x][1])
107         }
108         addShape( doorShape,extrudeSettings, 0xf302013, -750, -750, 0, 0,
    0, 0, 1);
109
110     }
111 }
112
113 function drawWindows(windows){
```

```
114     var extrudeSettings = { amount: 40, bevelEnabled: true, bevelSegments:
        2, steps: 2, bevelSize: 1, bevelThickness: 1 };
115     for(var i in windows){
116         var windowShape = new THREE.Shape();
117         windowShape.moveTo(windows[i][0][0],windows[i][0][1])
118         for(var x in windows[i]){
119             windowShape.lineTo(windows[i][x][0],windows[i][x][1])
120         }
121         addShape( windowShape,extrudeSettings, 0xf08000, -750, -750, 0, 0,
        0, 0, 1);
122         addShape( windowShape,extrudeSettings, 0xf08000, -750, -750, 160,
        0, 0, 0, 1);
123 
124     }
125 }
126 
127 function addShape( shape, extrudeSettings, color, x, y, z, rx, ry, rz, s )
        {
128 
129     var geometry = new THREE.ExtrudeGeometry( shape, extrudeSettings );
130     var mesh = new THREE.Mesh( geometry, new THREE.MeshPhongMaterial( {
        color: color}));
131     mesh.position.set(x, y, z);
132     mesh.rotation.set( rx, ry, rz );
133     mesh.scale.set( s, s, s );
134     group.add( mesh );
135 }
136 
137 function animate() {
138     requestAnimationFrame( animate );
139     controls.update();
140     renderer.render( scene, camera );
141 }
142 
143 function loadJSON(location, callback) {
144     var xobj = new XMLHttpRequest();
145     xobj.overrideMimeType("application/json");
146     xobj.open('GET', location, true); // Replace 'my_data' with the path to
         your file
147     xobj.onreadystatechange = function () {
148         if (xobj.readyState == 4 && xobj.status == "200") {
149             // Required use of an anonymous callback as .open will NOT
        return a value but simply returns undefined in asynchronous mode
150             callback(xobj.responseText);
151         }
152     };
153     xobj.send(null);
154 }
```

Listing A.7: Reconstruction Stage: Reconstruction algorithm

Appendix B

Wireframe Models