Computer Science
Carnegie Mellon University in Qatar

SCS Honors Undergraduate Research Thesis

Spring 2024

Indoor Mapmaking: From Map Drawing to Navigable Representations

by

Huda Baig

Advisors

Khaled A. Harras
Eduardo Feo Flushing

Carnegie Mellon University Qatar

Contents

Chapter 1

Introduction

1.1 What are Maps?

The term 'map' can be understood under a variety of semantics by different people. Each semantic understanding of maps lends itself to a different map format. According to the Merriam-Webster Dictionary, a map is "a diagram or other visual representation that shows the relative position of the parts of something" [11]. Maps have particularly been used to visually represent positions in outdoor spaces, so that travellers can determine where they are and gain an understanding of the relative space around them. These outdoor area maps were always imprinted on physical mediums, like paper, until the age of digitisation.

With the mass digitisation of information, paper maps were transformed into a digitised format that emerging systems could leverage in order to integrate location data into their functionality. Over time, these digitised outdoor maps were refined to embody more and more semantic information about the relative outdoor space so that applications could better perform context-aware functions.

In supporting context-aware functions, we focus our scope on human-centred functions. The reason we feel that human-centred applications deserve particular focus with regard to indoor maps is because there is huge quality-of-life potential in developing applications that can service human users with location-based services, as a reflection of how reliant we as human users have become on systems providing human-centred location functionality in the outdoor space, such as navigation systems, live tracking systems, among others.

Given the importance of systems having location information indoors, this thesis focuses on the digitised map representation that systems leverage to become more contextually aware of location information. Based on this focus, we define maps to be an analogical representation [8] that is structured in a way that systems can *directly access* semantic details and navigable links for key areas in the map's described space. Here, semantic details of key areas indicate important positional details, such as room name or coordinate location, relative to the larger described space. Coupled with semantics, navigable links for each key area in the described space indicate which other areas can be directly reached from the current one.

1.2 Motivation

By our definition of maps, there is already a standardised and efficient process for representing outdoor spaces as outdoor maps. In contrast to outdoors, indoor spaces don't have a standardised and efficient process to represent them as indoor maps.

1.2.1 Outdoor Mapmaking

Outdoor maps are created using satellite images of outdoor spaces as input. The input satellite images are imposed with location data from various organisations and governments [19]. Human annotators take these satellite images with imposed data and manually label key semantic details and navigable links necessary to form the outdoor map. These human-sourced labels have been collated into an outdoor map database that systems can query to directly access the location information they need to provide context-aware services for outdoor spaces.

Over the last few years, Google has optimised outdoor mapmaking by training machine learning models to automatically label input satellite images [19]. However, it is crucial to note that this automation was trained using the existing outdoor database, sourced on the accumulated effort of human annotators over decades.

1.2.2 Indoors vs Outdoors

Outdoor mapmaking cannot be applied directly to indoor spaces.

Firstly, there is no indoor map database for indoor mapmaking to be automated through, so manual effort by human annotators would have to be employed to label indoor spaces. Indoor maps created through the manual efforts of human annotators would require decades of work before sufficient data [19], matching the scale of outdoor spaces, is accumulated to explore automation, as we have seen with outdoor mapmaking.

Secondly, the nature of the semantic details and navigable links in indoor spaces is very different from that of outdoor spaces [17]. For example, we must account for separate structures like doors, stairwells, etc. Due to the difference in fundamental nature between indoor and outdoor spaces, indoor spaces need a completely separate labelling scheme.

Finally, we note that satellite images are not available for indoor spaces [20]. Instead, the most basic description format that can serve as input for indoor mapmaking is architectural blueprints of indoor spaces, known as CAD drawings. These CAD drawings are created when indoor spaces are physically constructed and can be accessed as vector-based CAD files or PNG image exports.

Given the key contrasts between indoor and outdoor spaces, we need to develop an indoor mapmaking process that can produce indoor maps in an accessible and efficient way. This will enable context-aware systems to operate in the domain of indoor spaces just as they operate in outdoor spaces.

1.2.3 General Mapmaking Approaches

In terms of the actual process for gathering the indoor space information required to build maps, there are two main methods that research pursues. Firstly, the robotics domain uses specialised hardware to extract information related to sensing and modelling indoor environments [12]. Using this indoor information, researchers have been able to construct spatial maps that allow robots to effectively navigate indoor spaces. Unfortunately, these spatial maps are not sufficient in enabling context-aware systems to perform the functions we are looking for indoors. This is because we focus on human-centred location functions, which require more semantics and information compared to the geometric information encoded by spatial maps. Even arguing that semantics can be manually added to spatial maps in order to bring them up to our requirements, we argue that the reliance spatial maps have on specialised hardware reduces their accessibility, as experts are needed to operate the hardware and building owners must be willing to supervise the access of such experts within their indoor spaces.

To address the drawbacks of robotic-sourced indoor space information, some research turns to data-driven approaches of extracting this information. Data-driven approaches generally follow the approach of performing segmentation to identify key areas within an indoor space [15], followed by separate algorithms to draw a topology that connects the areas identified [25]. Finally, manual work or inference models (of inconsistent accuracy) are used to add semantic details to the key areas. Unfortunately, these data-driven approaches suffer from many issues including every mapmaking stage being performed manually or being performed with inaccurate methods.

1.3 Challenges

The data-driven approaches, which attempt to increase the accessibility of indoor maps by removing the specialised hardware requirement, suffer from one or more of three major flaws. To contribute a mapmaking process that is more effective than the current approaches, we aim to address these challenges as much as possible. The main challenges we have found are:

- **Automation:** Related research proposed the use of data serialisation file formats to store semantic details and navigable links present in indoor spaces. In particular, IndoorGML proposes the use of XML-based files to form indoor maps, while IMDF proposes the use of JSON files to form indoor maps. Unfortunately, these maps require manual effort to be created.

  As we have already discussed, mapmaking processes that require manual effort to create indoor maps will need decades of work to reach the coverage that context-aware systems already have available for outdoor spaces. Due to this time investment demand, mapmaking methods that require manual effort are very inefficient in creating maps for a wide variety of indoor spaces.

- **Navigable Links:** As an alternative to data serialisation files to be used as indoor maps, some research proposed more visual file formats as indoor maps. These formats include 2D and 3D models, as well as SVG image files.

  Visual format-based mapmaking gives a human looking at the resulting map information about how key areas connect with each other. However, systems using these map formats cannot interpret these visual format maps in the same way to access navigable links. Without navigable links, systems cannot access key information such as how to navigate from point A to point B, severely limiting their functionality.

- **Ability to Scale:** Along with functional limits come scale limits for many proposed indoor maps. Indoor spaces come in a huge variety of sizes, from small house apartments to large public buildings like universities or airports. If mapmaking cannot accommodate large indoor spaces, it is severely restrictive on what types of spaces context-aware systems can work within.

Our goal is to design an indoor mapmaking method that can provide an indoor map while addressing all the shortcomings of current indoor mapmaking methods.

1.4 Contribution

In pursuit of our goal, we explored different methods of extracting information from CAD drawings and combining the different parts to form a cohesive indoor map that can facilitate a wide variety of context-aware location functions. Note that we restricted our input to PNG image exports of CAD drawings as vector-based CAD files expose architectural information that building owners are often reluctant to share, making PNG image exports more widely available.

1.4.1 Mapmaking Method

Our proposed mapmaking method is a processing pipeline consisting of six separate stages. The pipeline takes a PNG input with text labels of key areas as input. Our pipeline's modules are broken down as follows:

- **Input Slicing:** divides the input image into several smaller slices so that larger input spaces can be handled.
- **Area Extraction:** identifies all the key areas in every slice of the input image.
- **Door Extraction:** identifies all the doors in every slice of the input image.
- **Door Linking:** forms connections between key areas and doors identified.
- **Door Completion:** Goes through all identified doors to fill in missing navigable links.
- **Area Completion:** Goes through identified key areas to fill in missing navigable links.

Section 4 of this thesis will explain the rationale and methods behind each module.

Our mapmaking pipeline outputs a graph as the underlying structure for the final indoor map. Our graph structure represents key areas as nodes and navigable links as edges, while imposing additional properties to store the necessary semantic details that an indoor map needs.

1.4.2 Evaluation

To demonstrate the effectiveness of our mapmaking pipeline, we perform two separate sets of evaluations. Firstly, we investigate the navigable information that our output map, based on the graph data structure, encodes by checking the information related to the shortest paths to every area node from some arbitrarily chosen source node. To supplement our evaluation of navigable links, we evaluate the semantics of our output by checking the correctness of the semantic information of every node in the graph. We perform our graph evaluations based on ground truth graphs that we manually created of three large areas in the CMUQ building, which contain significant structural and spatial challenges that our pipeline must tackle to create the output map. Overall, our evaluation covers tests over $\sim 20,000m^2$ of indoor space within the CMUQ building.

To explain our research and rationale in detail, we structure this thesis as follows: Section 2 covers the research we have found related to data-driven approaches to indoor mapmaking and their associated challenges or flaws. Section 3 covers the mapmaking methods from related research that we experimented with to verify the components of these methods that work and could be incorporated into our pipeline, as well as the components that are the source of major flaws or inaccuracies. Section 4 covers the system architecture of our pipeline, with an in-depth description and algorithms of how every stage in our pipeline operates. Section 5 explains the evaluation metrics we use in detail, the results we obtained in evaluating our testbeds, and the results of experiments we performed on key parameters that our pipeline relies on. Finally, Section 6 summarises the current state of our work and ways to extend this research by considering its shortcomings.

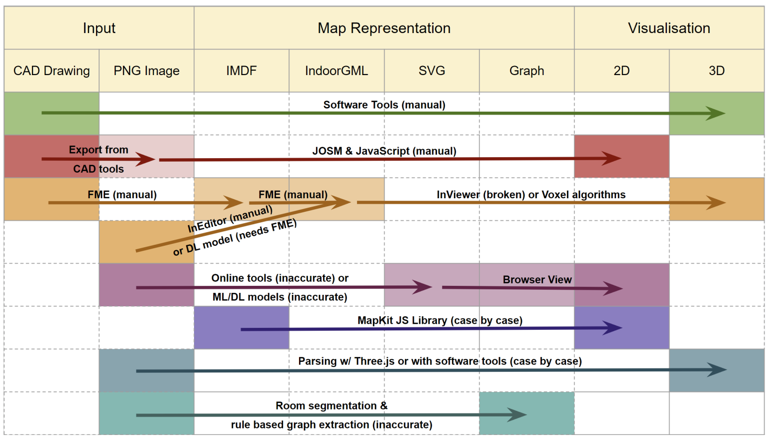
You can view our Mapmaking Pipeline implementation on GitHub[1]

[1]`https://github.com/morshed-research/cad2map`

Chapter 2

Related Work

Over the last few years, there has been a wide variety of research looking at indoor mapmaking. The indoor mapmaking methods we investigated explored either the capability to produce some existing image format as the final indoor map or to produce a new format that is tailored to fulfil the requirements of indoor maps. From our investigation, we found that the indoor mapmaking methods in current research can be divided into three categories based on the intended purpose behind the method. We summarise all the indoor mapmaking methods we found in Figure 2.1.

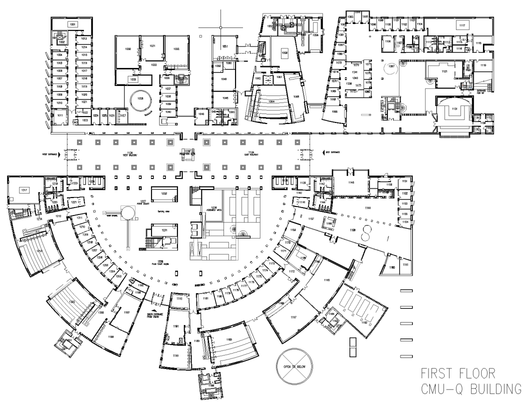


Input
Map Representation
Visualisation
CAD Drawing
PNG Image
IMDF
IndoorGML
SVG
Graph
2D
3D
Software Tools (manual)
Export from CAD tools
JOSM & JavaScript (manual)
FME (manual)
FME (manual)
InViewer (broken) or Voxel algorithms
InEditor (manual) or DL model (needs FME)
Online tools (inaccurate) or ML/DL models (inaccurate)
Browser View
MapKit JS Library (case by case)
Parsing w/ Three.js or with software tools (case by case)
Room segmentation & rule based graph extraction (inaccurate)

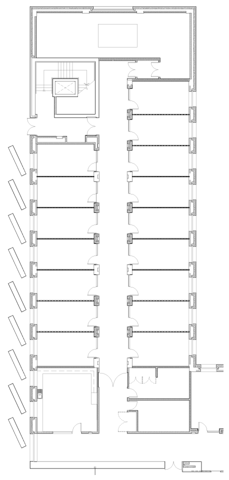
Figure 2.1: Summary of Indoor Maps present in research and their related mapmaking methods

2.1 Input

Input refers to file formats that are widely available and are used as industry standards for representing indoor spaces. These industry-standard formats aren't for use within context-aware location systems but rather for architectural design and construction purposes [8]. Almost all indoor spaces, if not all, will have one of these formats [1] available to provide details on their layout and features, making them very useful and practical input for indoor mapmaking.



(a)



(b)

Figure 2.2: (a) A CAD dwg of CMUQ's first floor (b) A PNG export of the CS Corridor from CMUQ's CAD dwg

2.1.1 CAD

CAD stands for Computer-Aided Design and is used to depict 2D drawings or 3D models. CAD files, which come as '.dwg' files and are referred to as drawings, are closely associated with designs and layouts for manufacturing and construction processes in domains such as architecture, engineering, city planning, etc. [7]. CAD '.dwg' files are vector-based file formats that come to scale.

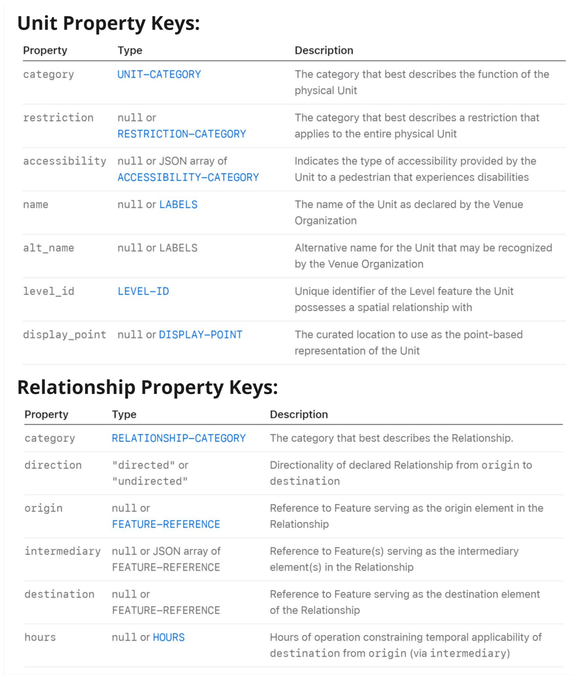
2.1.2 PNG Export of CADs

PNG images are a file format that is used to store computer images in high quality [14]. Although PNG images are not a specific file format for indoor spaces or any construction domain, CAD files can be exported as PNG images.

The PNG export of CAD files takes all the drawn structures for indoor spaces and flatten them into a single-layered pixel image. The flattened image retains visual shapes that build different parts of the indoor space within the CAD drawing, but it doesn't carry over the structural metadata such as geometric details, among others [18]. Because PNG exports expose less structural details about indoor spaces, building owners feel it is safer for the design integrity of their building to share PNG exports rather than the vector-based CAD file itself. The flattened PNG image is also easier for some systems to handle as less information has to be processed.

2.2 Map Representations

Back-end representations refer to indoor maps that focus on encoding semantic information about indoor spaces for specific use cases. While some representations also encode navigable links alongside semantics, back-end representations emphasise storing semantic information in a parsable format that can provide a descriptive representation of indoor spaces rather than a representation that can *directly* facilitate context-aware functionality.



Unit Property Keys:

Property	Type	Description
category	UNIT-CATEGORY	The category that best describes the function of the physical Unit
restriction	null or RESTRICTION-CATEGORY	The category that best describes a restriction that applies to the entire physical Unit
accessibility	null or JSON array of ACCESSIBILITY-CATEGORY	Indicates the type of accessibility provided by the Unit to a pedestrian that experiences disabilities
name	null or LABELS	The name of the Unit as declared by the Venue Organization
alt_name	null or LABELS	Alternative name for the Unit that may be recognized by the Venue Organization
level_id	LEVEL-ID	Unique identifier of the Level feature the Unit possesses a spatial relationship with
display_point	null or DISPLAY-POINT	The curated location to use as the point-based representation of the Unit

Relationship Property Keys:

Property	Type	Description
category	RELATIONSHIP-CATEGORY	The category that best describes the Relationship.
direction	"directed" or "undirected"	Directionality of declared Relationship from origin to destination
origin	null or FEATURE-REFERENCE	Reference to Feature serving as the origin element in the Relationship
intermediary	null or JSON array of FEATURE-REFERENCE	Reference to Feature(s) serving as the intermediary element(s) in the Relationship
destination	null or FEATURE-REFERENCE	Reference to Feature serving as the destination element of the Relationship
hours	null or HOURS	Hours of operation constraining temporal applicability of destination from origin (via intermediary)

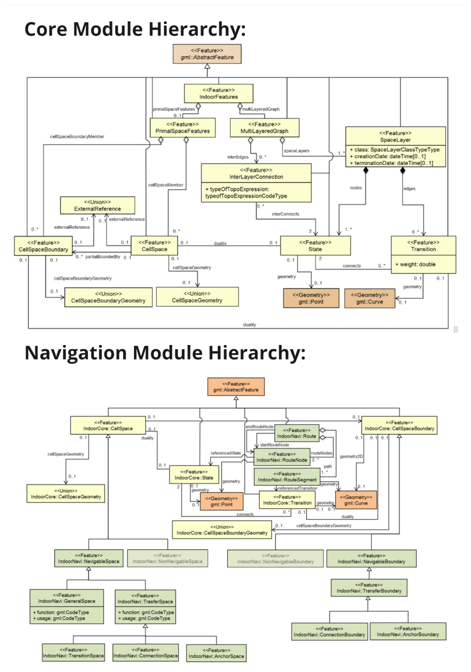


Figure 2.3: Left: JSON Properties for IMDF. Right: XML Structure for IndoorGML.

2.2.1 IMDF

The Indoor Mapping Data Format (IMDF) is a file format that was specifically designed to create accurate maps for indoor spaces [3]. It represents and stores information about indoor spaces using property-value pairs arranged in a JSON structure.

Feature Representation

IMDF generally represents the components of indoor spaces in two categories. The first category is called a 'Unit', which models bounded areas such as rooms, hallways, etc. Here, we note that bounded areas may not necessarily be bounded by walls, but instead can be bounded by anything that denotes a boundary that cannot be walked past (such as a train platform) [3]. The second category is called a 'Relationship', which models some association between two existing objects. Its properties depend on the values of its property fields.

IMDF-based Mapmaking

While IMDF's information structure stores everything we need for our indoor map, the mapmaking process of producing an IMDF-based indoor map is completely manual. Manual means that a human is expected to take an indoor space and label it from scratch using the JSON-object structure of IMDF. Any software claiming to generate the IMDF JSON-structure in the back-end still requires an annotator to input all semantic properties and relations themselves, which would require hours of work.

2.2.2 IndoorGML

IndoorGML is another file format specifically designed to create accurate maps for indoor spaces [16]. It represents information about indoor spaces with an XML-based file structure.

Feature Representation

Like IMDF, IndoorGML breaks down the representation of indoor spaces into representing spaces and their relationships.

For spaces (such as rooms, corridors, stairs, etc. [16]), IndoorGML refers to these as cells and aims to associate each with a unique id and coordinate to be identified by. These uniquely identified cells can also be broken down into subspaces and accessibility layers for more close-up details regarding the space. Cells representing indoor areas have their own XML class of 'State', which has certain properties to describe cells.

For relationships between spaces, IndoorGML refers to these as transitions. Transitions are represented by two combined XML classes to depict all their properties, 'RouteNode' to depict the cells they connect and 'RouteSegment' to describe the properties of the actual transition between the cells.

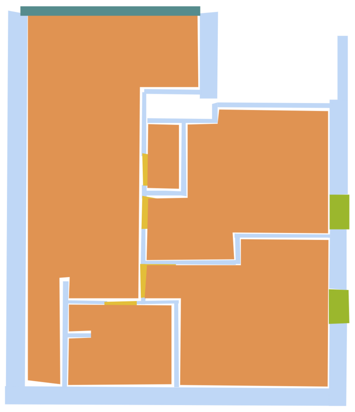


Figure 2.4: An example SVG where all rooms have been labelled in orange, walls are labelled in blue, windows in green and doors are labelled in yellow. [10]

IndoorGML-based Mapmaking

There are several mapmaking methods that attempt to produce IndoorGML. Unfortunately, all of these have a high manual overhead, as they need a human annotator to manually input detailed labelling of the desired indoor space properties at different stages of the mapmaking process.

2.2.3 SVG

Scalable Vector Graphics are an XML-based file format for representing images that can be scaled while maintaining their quality. In particular, SVGs can be searched since their properties are represented in text with XML-based tags [23]. While SVG files are not a file format created specifically for representing indoor spaces, many researchers make attempts to transform CAD drawings (or PNG images of CAD drawings) into an SVG image with semantic labelling of indoor areas embedded into their tags. Representing indoor areas and their labelling with SVGs is beneficial due to the standardisation of SVG by the World Wide Web Consortium, which means it is a standard that is constantly maintained and upgraded, as well as the fact that most web browsers can easily display SVGs [10].

Feature Representation

SVG images represent shapes using tags which depict individual shapes, such as ‘polygon’, and draw these shapes using coordinate and CSS properties, such as vertices for polygon tags, specified in the tag’s attribute fields [23]. Using the colour attribute tag, SVGs can depict basic semantics about key areas, such as their type or their coordinate location on the indoor space as a whole.

SVG-based mapmaking

At present time, SVG cannot currently encode any navigable information between areas (for example, can I walk between two rooms or not, or explicitly represent if two rooms are adjacent, etc.). Due to the lack of navigable information between areas, SVG cannot serve as our final indoor map.

Beyond the lack of navigable links, the mapmaking methods that produce SVG suffer from inaccuracies in representing indoor spaces. These methods often incorrectly interpret key information such as neighbouring rooms or boundary walls. Even if key information can be corrected, these methods also fail to scale to large indoor spaces.

2.2.4 Graph

Graphs refer to the graph data structure that is often used in many applications of the computer science field. In related research, researchers try to convert base representations into graph structures in order to perform algorithmic comparisons between the structure of apartments [25]. Searching by structure is very important for apartment buyers [24].

Feature Representation

In general, it is expected that when representing indoor spaces, the graph nodes will represent key areas like rooms (as well as storing relevant semantics about these areas), and the edges will represent adjacency information (i.e., a threshold of closeness) between areas from which navigation can be derived [25].

Graph-based Mapmaking

These graphs choose to represent edges as some threshold adjacencies [25] between areas (because only adjacencies are relevant when depicting the structure of an indoor space), they don't hold direct navigable information as closeness does not always guarantee navigable links (a wall may stop a person from passing through, etc.)

In terms of actual mapmaking, these adjacency graphs are produced with segmentation methods that are similar to SVG-based mapmaking. Similarly, they suffer from the same shortcomings of inaccuracy.

2.3 Visualisation

Visualisations in Figure 2.1 refer to methods that produce a visual interpretation or replication of indoor spaces. Although it may be argued that these visualisations could be considered an indoor map because it allows humans to orient themselves around an indoor space with its semantic information, they cannot be used by systems. Based on the fact that systems cannot use visualisations on their own to access semantic details and navigable links within an indoor space, we do not consider visualisations as indoor maps and only include them to give a complete view of mapmaking methods in related work.

2.4 Human in the Loop Approaches

Human in the Loop approaches refer to mapmaking methods which require humans to contribute some level of information and then operate to create some map format based on the information that a human manually provided. Note that these approaches are distinct from completely manual processes as manual work goes into providing only a part of the necessary information and algorithms are applied onto this partial information to complete the picture. In this way, the manual work involved is minimised.

Our work is classified under human-in-the-loop approaches as we take a small level of annotation from a human in our input images. Because of this classification, we looked into whether there are any related mapmaking methods that also employ human-in-the-loop approaches. We found one approach that creates geometric-based SLAM maps from CAD files [9]. However, we do not include this work in our summary table because it generates information that is very specific to robot movement and cannot be generalised to providing any related semantics or navigable link information that our indoor map needs.

Aside from the different information set that this SLAM human-in-the-loop mapmaking process provides, we also note that it has a high information demand from human annotators that will prevent it from scaling to large spaces. In essence, this method aims to generate geometric structure maps (i.e., only defining room shapes with no links or semantics) by asking a user to define all possible trajectories [9], which would be impossible for large indoor spaces, that may go through an indoor space. Based on the trajectories, parametric models are used to infer the relevant geometric structure for robotic movement. Ignoring the lack of applicability of SLAM maps in deriving semantic and navigable information, the high manual demand of even human-in-the-loop approaches to related mapmaking, it is clear that there is no mapmaking method that minimises manual work while ubiquitously fulfiling the requirements of our indoor map, especially for large spaces.

Chapter 3

Exploring Mapmaking Methods

The first step in devising our proposed mapmaking pipeline was to experiment with existing indoor mapmaking methods. From our exploration of these methods, we hoped to gain insight on any successful modules we could leverage in our pipeline, as well as understanding what types of information could be extracted from our input CAD drawing.

Unfortunately, some of the mapmaking methods described in the literature are locked behind paid subscriptions, and we could not experiment with these, so we opted to read the documentation to understand how they work.

3.1 IMDF-based Mapmaking

Currently, the only mapmaking process that can be used to create an IMDF representation of an indoor space is to use a conversion tool called Feature Manipulation Engine (FME). Using FME, a user can supply the tool a CAD dwg of indoor space and then use the interface to manually label indoor areas and properties they want to add as part of their IMDF representation. Once the user has manually entered all the information they want to include in their IMDF representation, FME generates the final IMDF JSON objects.

The most significant flaw with this FME-based mapmaking process is that it is completely manual, meaning it has to be individually done for every building that wants to use an indoor map. Because of this manual overhead in the mapmaking process, along with the previous flaws noted in its representation, IMDF cannot fulfil the requirements of the navigable indoor map we are seeking.

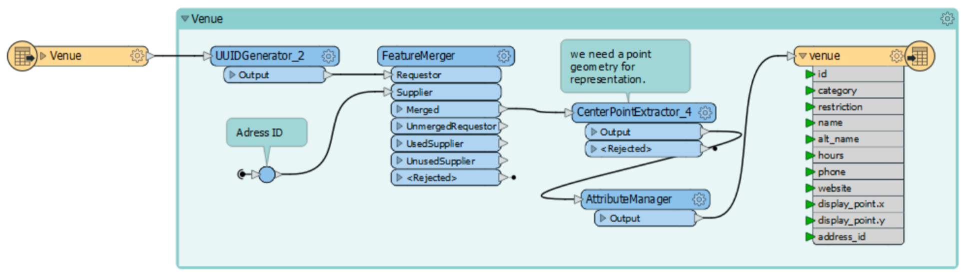


Figure 6.5 Transforming the Venue layer in the AutoCAD file to the feature venue in IMDF, in FME.

Figure 3.1: An example of using FME to build an IMDF representation [2]

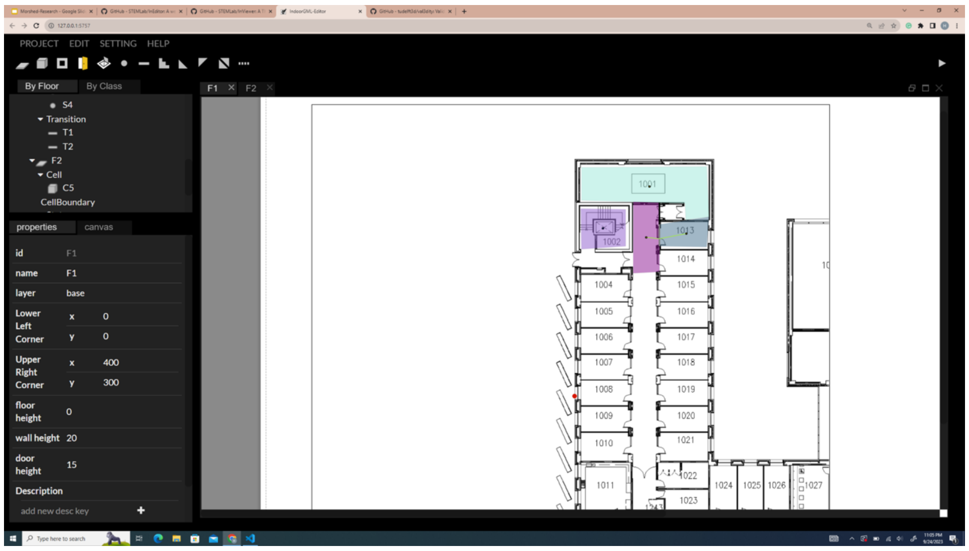
3.2 IndoorGML-based Mapmaking

The first available mapmaking process to produce an IndoorGML representation of an indoor space is again using the Feature Manipulation Engine tool. The process of using FME to produce an IndoorGML representation is almost identical to IMDF's process, except that some property labelling by the user will differ. Once the user has entered all information regarding the indoor space, FME will generate the final IndoorGML. As we have already mentioned, the FME-based mapmaking process does not meet our requirements of a navigable indoor map as it has too high of a manual overhead.

The second available mapmaking process to produce an IndoorGML representation of an indoor space is to use a deep learning model that takes in a PNG image of a CAD drawing and produces a vector representation of the image. The vector data is then used with FME to produce a final IndoorGML representation [13]. Because this mapmaking process again relies on FME to be fully completed, we regard it to have the same issue of high manual overhead as we found with the first mapmaking process.

The final available mapmaking process to produce an IndoorGML representation of an indoor space is to use the software tool named 'InEditor'. InEditor is a software interface where a user can upload a PNG of a CAD drawing and use the provided tools to manually outline (by selecting vertices) what boundaries depict cells from IndoorGML [22]. Upon creation, users can manually fill in the properties of cells in a side pane. Once all cells have been identified, the user must use another tool to connect cells that have transitions between them. Once again, users can manually fill in transition properties. The use of InEditor to create an IndoorGML representation is a completely manual process guided by interface tools, as demonstrated in Figure 3.2.

Overall, we have found that IndoorGML poses issues with its representation not supporting access to localisation and navigation information directly, and its mapmaking issues having high manual overhead. Because of



id		F1
name		F1
layer		base
Lower Left Corner	x	0
	y	0
Upper Right Corner	x	400
	y	300
floor height		0
wall height		20
door height		15
Description		

Figure 3.2: Using InEditor to create IndoorGML for a subset of the CS corridor

these two issues, IndoorGML cannot fulfil our requirements for a navigable indoor map.

3.3 SVG-based Mapmaking

Scalable Vector Graphics are an XML-based file format for representing images that can be scaled while maintaining their quality. In particular, SVGs can be searched since their properties are represented in text with XML-based tags [23]. While SVG files are not a file format created for specifically representing indoor spaces, many researchers make attempts to transform CAD drawings (or PNG images of CAD drawings) into an SVG image with semantic labelling of indoor areas embedded into their tags. Representing indoor areas and their labelling with SVGs is beneficial due to the standardisation of SVG by the World Wide Web Consortium, meaning it is a standard that is constantly maintained and upgraded, as well as the fact that most web browsers can easily display SVGs [10].

SVG images represent shapes using tags which depict individual shapes, such as ‘polygon’, and draw these shapes using coordinate and CSS properties, such as vertices for polygon tags, specified in the tag’s attribute fields [23]. For example, Figure 2.4 shows an example SVG where all rooms have been labelled in orange, walls are labelled in blue, windows in green and doors are labelled in yellow.

Using the colour attribute tag, SVGs can depict basic semantics about key areas such as their type or their coordinate location on the indoor space as a whole. Beyond these features, SVG cannot currently encode any relational information between areas (for example, can I walk between two rooms or not, or explicitly represent if two rooms are adjacent etc.). Because of the lack of navigable information between areas, SVG cannot serve as our final navigable indoor map. However, if SVG is a format that can be produced easily with its mapmaking methods, we could leverage its capabilities by creating a new mapmaking process that starts with SVG instead of CAD due to the improved parsability of XML-based files.

Mapmaking Process

SVG-based indoor maps have two mapmaking processes. Firstly, there are many tools available online that can convert a PNG image into an SVG image. Given any one of these SVG tools, we can supply a PNG image of a CAD drawing and receive an equivalent SVG image. Unfortunately, the SVG image produced by these tools may look visually accurate, but it usually represents shapes in the image through one or two XML tags and fails to separate individual rooms and areas with their own tags and properties, thereby losing a lot of the semantic detail we wish to represent.

The second available mapmaking method is the use of deep-learning models to produce SVG-style floor plan images by inputting a PNG image of a CAD drawing into the model [15]. These deep-learning models attempt to segment the given PNG with different colour labels for walls, doors and different room types. To derive the labelling, the PNG image is passed to the model to create a pixel-based, labelled image. This label image is then passed to a post-processing algorithm to separate rooms, walls and other components into strict, separate shapes [15].

We spent a lot of time experimenting with different indoor spaces as input determine how accurately the model could label indoor spaces. We initially tried a low quality input of a simple corridor with many offices, finding that no wall except the outer boundary wall of the building was detected. After this we experimented with higher quality images, smaller segments and different shaped areas. Overall, our experimentation showed that any indoor space cut down to roughly 4 rooms could be segmented and labelled accurately, likely because the model was trained on houses of the same size [15]. Areas that exceeded this size were often missing many walls in the final labelling. We also found the labelling produced after post-processing was unable to account for any curved shapes, even if the model's segmentation detected these curves to some extent (even then the model's detection of curves was not too accurate). Overall, we found that SVG's model-based conversion process was accurate on a very particular subset of indoor spaces, and depended a lot on the quality of the input image.

Given the inaccuracy of SVG's mapmaking processes, as well as the missing details in the information it can store about indoor spaces, SVG does

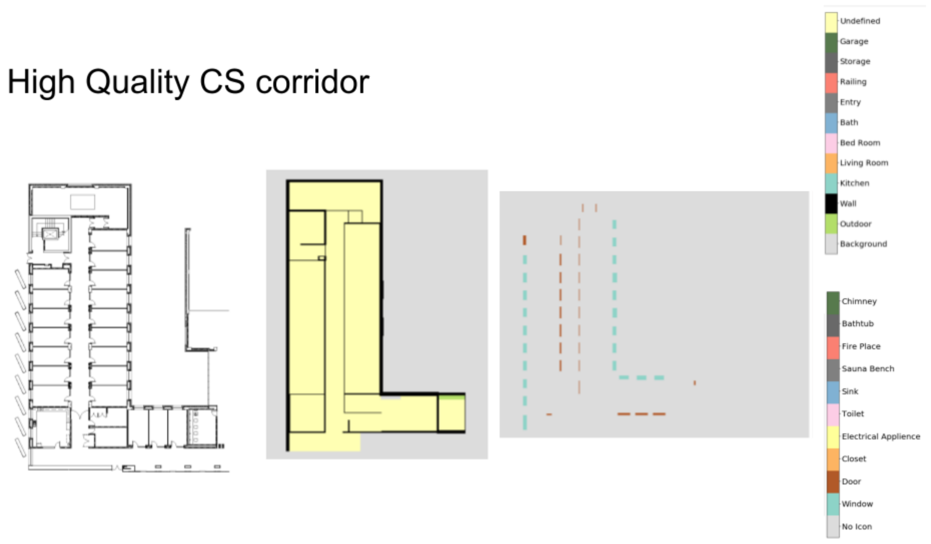


Figure 3.3: An attempt to use the deep-learning model segmentation on the CS Corridor. Left: PNG image, Middle: Segmented rooms and walls after post-processing (many walls are missed), Right: Segmented windows and doors after post-processing

not meet the requirements of our indoor navigable map. We also considered whether, in building our navigable indoor map, SVG could be leveraged as a starting point since XML is easier to parse than image format of our base representations. Unfortunately, SVG cannot serve as an intermediate representation in our new mapmaking process due to the inaccuracies in its own mapmaking process, meaning we cannot assume that any base representation could be reliably converted into SVG.

3.4 Graph-based Mapmaking

Currently, there is only 1 mapmaking process associated with graphs. This mapmaking process consists of two stages. Firstly, similar to [15], it performs a deep-learning-based segmentation of floor plans to extract details of where walls are and what room types exist with what boundaries. Once segmentation has been performed successfully, the resulting output goes through a rule-based extraction to map the found areas to nodes and derive edges based on distance thresholds (usually edges are drawn between areas that are no more than 30m apart [25]).

Unfortunately, as we have already seen, deep-learning-based segmentation is inaccurate for many types of indoor spaces. Since this indoor mapmaking method depends on accurate segmentation as its first stage, it is not ubiquitously applicable to any indoor space so it cannot be used as our

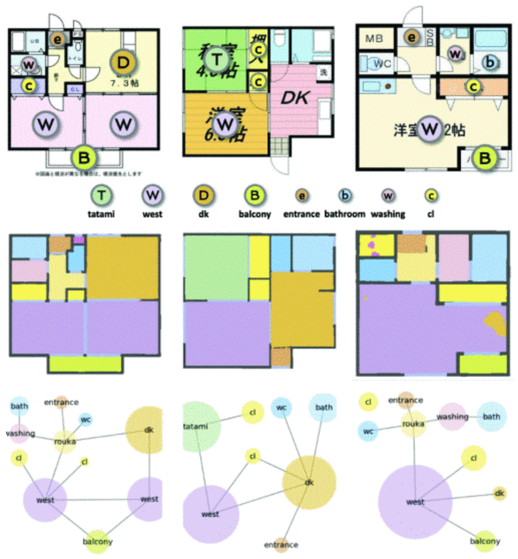


Figure 3.4: The process of taking a PNG image of an indoor space, segmenting it and extracting the graph [24]

navigable indoor map in its current inaccurate state.

Chapter 4

Our Mapmaking Pipeline - System Architecture

From the current state and flaws in existing indoor mapmaking methods, it is clear that we need a new mapmaking method that is capable of producing a map tailored to the requirements of context-aware location systems. The novel mapmaking method should be easy and efficient to use so that it is accessible to most, if not all, systems that wish to make use of it.

We formulate a novel indoor mapmaking method that takes a PNG image of a CAD drawing as input, applies the image to a multistage pipeline, and outputs a graph structure tailored to meet the requirements of an indoor map. Our complete pipeline is illustrated in Figure 4.1

Based on our discoveries among existing indoor mapmaking methods, we created a pipeline consisting of six stages to produce an indoor map. Our pipeline is devised in such a way that it addresses the major flaws that most indoor mapmaking methods face, being automated, efficient, scalable, and containing all the necessary semantic details and connective links.

4.1 Pipeline Input

The input to our mapmaking pipeline is a PNG export of a CAD drawing. Here, we note that we chose to have the input of the PNG export image rather than the vector-based CAD file. There are several reasons why we favoured the PNG as input, the most important of which was that we want our input to be as accessible as possible. Unfortunately, the accessibility of vector-based CAD files is limited as they are known to expose key architectural information about buildings [8]. Given their exposing nature, many building owners are reluctant to share their vector-based CAD file and instead prefer to share the PNG export of the CAD drawing.

Aside from how PNGs are less exposing, in older buildings, the original vector-based CAD file is sometimes lost, with only the PNG export remain-

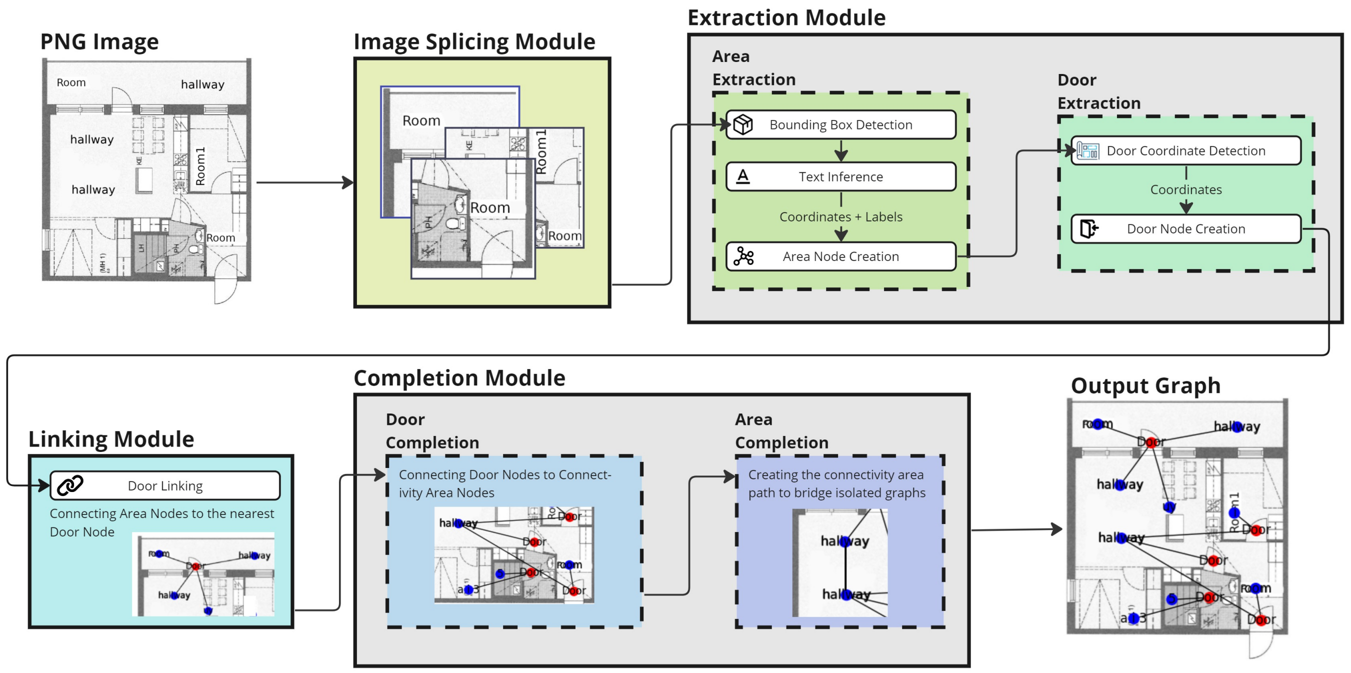


Figure 4.1: Our proposed system pipeline

ing. By choosing PNG images as our input, our indoor mapmaking pipeline becomes accessible to even these older buildings.

4.1.1 Input Assumptions

Because our input is a PNG image, which is a fairly flat file format, we don't have much information to work with beyond the layout and colours of pixels in the image. In order to add more detailed information for our pipeline to derive an indoor map from, we make several assumptions about the PNG. To give a sense of the actual distance within the image, we assume that the input PNG will have a minimum of A0 resolution (150 or 300 dpi) and display the indoor space with a mapping of 50 pixels per metre. We justify this chosen mapping in Section 5.

Aside from the pixel-to-distance mapping, we assume that our input PNG will have text as part of the image, which will label all key areas relevant to the indoor map in the image. Examples of these key area labels for maps include room numbers for offices, shared areas like lounges, and "passage way" areas, which we describe as connective areas moving forward, such as hallways or walkways.

Within the time frame of this thesis, we were unable to solve the issue of breaking down large connective areas, such as hallways, into smaller subareas. Having subareas is desirable because it makes it easier to identify which sections of a larger space destination, like offices, connect to, and it allows systems to better gauge distances for areas in the map as there is less contrasting size between larger and smaller areas. To simulate this large area breakdown, we will assume that large areas will be labelled with multiple sequential labels.

We argue that our assumed labelling is reasonable as it is a low-effort assumption. Labelling a PNG is low-effort because:

- It does not need expertise
- It does not need customised tools
- It can be crowdsourced at a reasonable cost, if needed

4.1.2 Label Categories

The labels for key areas in our PNG input are divided into three categories. We define different categories for key areas to not only give them richer semantic information but also because areas in each category share common properties that change how navigable links can be inferred for each of these area categories. Particular stages of our mapmaking pipeline leverage properties for inferring navigable links for specific categories.

The first category of label is "Destination". This label specifies key areas of indoor spaces that are notable as places where people go. We select destinations as our first category because we have observed that destinations branch out into more open areas from their enclosed space. To leverage how destinations branch out into the rest of their wider indoor space, we mark them out as a label category. Possible destinations include an office, a cafe, or a prayer room.

The second type of label is "Connective Areas". This label specifies the areas of indoor spaces that are used to pass through from one key area to the next. We've observed that connective areas are typically the areas that lie between enclosed spaces like rooms, offices and bathrooms, making connective areas a convergence point of key areas within an indoor space. In particular, we note that beyond being a convergence point, connective areas form pathways between one another to allow people to move from one convergence point to the next and access a different set of converging key areas. Possible connective areas include hallways and bridges.

The last type of label is "Shared Spaces". This label marks key areas that are used as communal, for multiple people to gather. Shared spaces come into play mainly when enclosed rooms are nested inside a larger enclosed space rather than converging onto a connective area. To better handle larger enclosed spaces with 'sub-rooms', we mark the larger space as a shared area but we treat it similarly to a connective area, as it is a convergence point on a smaller, enclosed scale. Possible shared spaces include shared department spaces and lounges.

Based on the observations that inspired our label categories, we also assume that destinations will only directly connect to shared spaces or connective areas.

4.2 Task Formalisation

Based on our input requirements, we formalise our task as follows:

> We are given a PNG image representing an indoor space according to our input specifications, with text labels $< x_1, x_2, ..., x_n >$ marking key areas within the PNG image itself. Then our task is to find a graph $G = (V \cup D, E)$. G's nodes are such that each v_i for $1 \leq i \leq n$ is a node that represents the area denoted by the text label x_i in the image and each $d \in D$ represents a door in the PNG image. G's edges are such that for each edge $(u, v) \in E$, a person can directly navigate (i.e. walk/travel) from the area or door represented by node u to the area or door represented by node v.

4.3 Image Slicing

The Image Slicing module divides the input PNG into multiple smaller images. We divide our input image into smaller images so that the door and area extraction modules have a smaller pixel area to operate on. A smaller area is crucial for the following models as they use detection algorithms that are optimised for small-scale indoor spaces. To make these small-scale algorithms applicable to large indoor spaces, we divide our input image to smaller images of fixed width and height.

We formulate our smaller images in a sliding window fashion. Based on a fixed width w and height h and an overlap percentage p, we start at the image coordinate origin $(0, 0)$ at the top left and create a cropped image that covers a region of w x h for the image. We refer to this cropped image as a slice. We then continually shift the top left x-coordinate to create the next cropped image by taking the previous x-coordinate and adding to it $w * p$. Once we have covered the full width of our original input image with slices, we repeat the process for a top left y-coordinate of $previousy + (h * p)$. We repeat the y-direction shift until the full width of our original input image has been converted into slices. Each new slice is saved as a new image file under the name "panel-*sliceindex*.png" and its filename, top left x coordinate, top left y coordinate, bottom right x coordinate and bottom right y coordinate is saved in a python Pandas DataFrame. We display our algorithm in detail in Algorithm 1

We note that we choose to create a region of overlap between our slices to reduce the likelihood some label or door is not detected due to it being cut off between slice boundaries. We discuss optimising the percentage overlap to minimise cutoffs in Section 5 and as such we assume that at least 1 slice has each label/door without any cut-off.

Algorithm 1 Image Slice algorithm

```
Input: A PNG image that was exported from an Indoor CAD, each slice's width and height and the overlap percentage between slices
Output: A DataFrame containing the filename of each slice, its top left and bottom right coordinates

function SLICE_RECTANGLES($w_{img}$, $h_{img}$, $w_{slice}$, $h_{slice}$, $p$)
    $boxes \leftarrow []$
    $y_{min}, y_{max} \leftarrow 0$
    $y_{overlap} = h_{slice} * p$
    $x_{overlap} = w_{slice} * p$

    while $y_{max} < h_{img}$ do
        $x_{min}, x_{max} \leftarrow 0$
        $y_{max} \leftarrow y_{min} + h_{slice}$

        while $y_{max} < h_{img}$ do
            $x_{max} \leftarrow x_{min} + w_{slice}$
              ▷ bound between 0 and image width/height when necessary
            $boxes \leftarrow boxes + [x_{min}, y_{min}, x_{max}, y_{max}]$
            $x_{min} \leftarrow x_{max} - x_{overlap}$
        end while
        $y_{min} \leftarrow y_{max} - y_{overlap}$
    end while
end function

$image\_boxes \leftarrow$ SLICE_RECTANGLES($width(image)$, $height(image)$, $w$, $h$, $p$)
$df \leftarrow$ iterate over $image\_boxes$        ▷ store each box with panel name
```

4.4 Extraction Module

Our sliced images are passed to the Extraction Module. The Extraction Module is responsible for creating all the necessary nodes and adding them to the global graph that will serve as our output. Using a series of sub-modules, the Extraction Module first identifies key areas, including all label types of destination, shared spaces and connective areas, and adds each of these as area nodes of each label type. Then it goes on to identify all the doors present in the original input image and adds each of these as door nodes. The sub-modules that create these nodes leverage deep-learning-based techniques to perform node extraction.

4.4.1 Area Extraction

The Area Extraction module identifies key areas in the input image, converts these into area nodes, and adds these nodes to a global graph. The graph will be used to represent the indoor space at the end of our pipeline. Each node is added to the global graph with a rough coordinate location of the area it represents and a label describing the area. The node and its associated properties are extracted by three separate parts.

Bounding Box Detection

Bounding box detection is used to identify the coordinate region associated with each text label that pinpoints a key area in the indoor space. We obtain the coordinate region of each label by using a deep learning model to output the top left and bottom right coordinates of the tightest bounding rectangle that encloses each label. Because text bounding box detection is a well-established task in the field of deep learning, we leverage an existing Convolutional Neural Network designed by Clova AI [6], known as Character Region Awareness for Text detection, or CRAFT for short, to extract the bounding boxes of the labels in our input image. Because the CRAFT model is tailored to images with a few words, we apply the model individually to the image slices produced by the previous stage in our pipeline rather than to the input PNG as a whole.

Integrating the CRAFT model into our pipeline required us to implement function wrappers [21] to modularise the input of each image slice, detection on the image, and output of the bounding boxes. In particular, we focused on ensuring that the output was directly passed out of the module rather than being saved to intermediary files like CSVs to reduce computational time and memory.

Text Inference

Once the bounding boxes are obtained, we use another deep-learning model to extract the textual meaning of the bounded label so that its node can be associated with this label. For this text inference on labels, we leverage the deep text recognition model by Clova AI [5]. We also put in effort to connect this second model to the existing bounding box detection module, so that the output bounding boxes would be directly used to infer the text associated with each individual bounding box. The connected models then serve to output the bounding box and associated text as one output to the area extraction module as a whole.

Notably, we crop out each label from the input slice before passing it for text inference. Making each text label the only text inference improves the accuracy of the inference. However, we note that text inference is an established OCR task, so moving forward, we assume that the module will infer text with complete accuracy. In cases where the text is incorrect, we leave it to optimisations within the field of OCR.

Area Node Extraction

After identifying the bounding box coordinates and associated text inference for each label in our input image, we work on creating nodes for each label. We note that, since our models operated on individual slices, we create nodes per slice and then perform a merging operation across nodes for all slices.

In terms of creating nodes, for each bounding box we calculate the centre coordinate and make a node for this center coordinate. We store the centre coordinate and the inferred label as part of the node's properties. We also mark the node as an "area" node in its properties. Flagging the node's type will serve the purpose of distinguishing actual areas and the door nodes we will be creating in the next pipeline stage. Every created node is added to a global graph that is common to all input slices.

Because our input slices are formed with a region of overlap, it is possible that some area labels have more than 1 node created for them. To resolve this duplication of nodes, we perform a merge operation. Our merge operation combines nodes that are within 2 metres of real distance, taking the new centre coordinate to be the midpoint of the coordinate from the nodes being merged. We reason that 2 metres is an acceptable lower bound for the widths of rooms because it is close to most architectural standards for minimum room width [4].

For the label of the merged node, because we assume that at least 1 slice has our label without cut-offs, we take the longer label from the two nodes being merged because this will eventually converge to keeping the label from the slice with no cut-offs.

4.4.2 Door Extraction

The Door Extraction module identifies all doors in the input image, converts them to door nodes, and adds these nodes to the global graph. Each node is added to the global graph with a rough coordinate location of the door. We perform door extraction in 2 parts.

Door Coordinate Detection

We extract the (roughly) centre coordinate of the door by training our own deep learning model to recognise and output the bounding boxes of doors in the input image. To make detection easier, this model operates on the slices of the input image produced by the Image Slicing module.

We train our model by performing fine-tuning on the Faster-RCNN object detection model. We fine-tune the model using a dataset that consists of PNG images exported from CAD drawings of home apartments for the model's input and coordinates of the top left & bottom right of the door's bounding box for the model's output. We created this custom dataset by leveraging the CubiCasa5k project dataset [15].

The CubiCasa5k dataset consists of PNG exports of CAD drawings for home apartments and SVG reconstructions of the CAD drawing to serve as labelling for the apartment space. We use the PNG images as they are, and we extract the bounding box coordinates for our doors based on the attributes that doors are labelled within the SVG files.

Algorithm 2 Algorithm to extract door bounding boxes from SVG

```
1:  function DOOR_BOXES($SVG\_file$)
2:      $door\_tags = t : t \in tags(SVG\_file) \mid t.id =$ "Door"
3:      $boxes = []$
4:      for $tag \in door\_tags$ do
5:          $polygon = tag.find($"polygon"$)$
            ▷ find first polygon subtag, e.g. BeautifulSoup tag find function
6:          $X = []$
7:          $Y = []$
8:          for $s \in polygon[$"$points$"$]$ do          ▷ data of points attribute
9:              $X \leftarrow X + s[0]$
10:             $Y \leftarrow Y + s[1]$
11:         end for
12:         for $s \in polygon[$"$path$"$]$ do            ▷ data of path attribute
13:             $X \leftarrow X + s[0]$
14:             $Y \leftarrow Y + s[1]$
15:         end for
16:         $boxes \leftarrow boxes + (SVG\_file, min(X), min(Y), max(X), max(Y))$
17:     end for
18: end function
19: DOOR_BOXES(file) for $file \in all\_files$
```

Door Node Creation

Like with the area nodes, we create nodes for each door after identifying the bounding box coordinates for each door in our input image. We note that since our model was trained on small apartments, we operate it on the image slices instead of the whole input image. We then create nodes per slice and perform a merging operation across nodes for all slices.

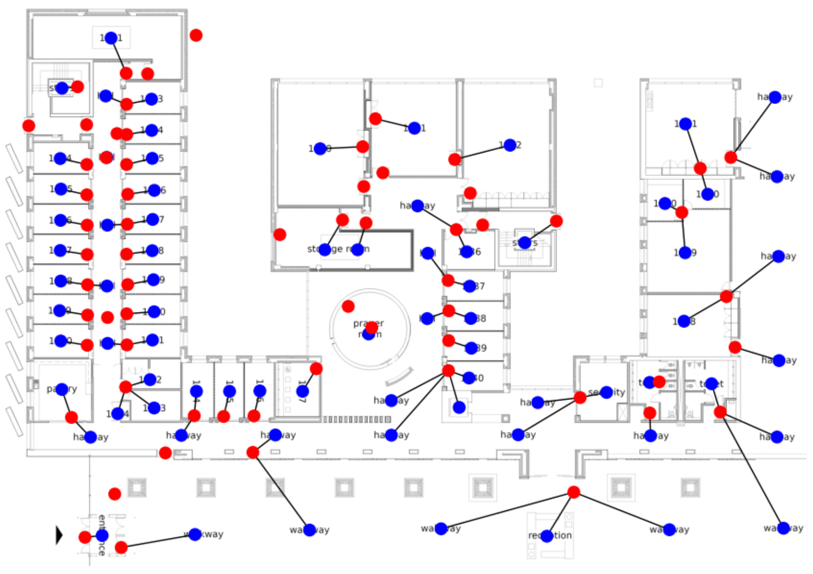


Figure 4.2: A demonstration on creating navigable links through doors on a subset of CMUQ. Blue dots mark area nodes and red dots mark door nodes.

In terms of creating door nodes, we use the same method as for area nodes, but specify the label as "Door" for every node. We also change the node type to be the door for all nodes.

We also use the same method for merging door nodes as we did for area nodes. Here, the label update is not relevant, as all the nodes have the same labels.

4.5 Linking Module

The Linking Module is responsible for adding edges as an initial, albeit incomplete, sense of connectivity between nodes. Our Linking Module consists of one key algorithm that leverages connective properties between area nodes and doors, mainly the fact that when moving from one area to the next, a person must pass through either a door or an opening (in the wall of the area). Our algorithm is named "Door Linking".

4.5.1 Door Linking

With all the necessary nodes created, we move on to creating the edges between the nodes that represent the navigable links between areas. We connect key areas through the doors that people would have to pass through.

To associate every area node with the door that has to be passed through to exit the said area, we connect every area node to its nearest door. To do this connecting, form two lists of area nodes and door nodes, respectively (identified by the node type). For every area node in the list, we calculate the Euclidean distance from the area node's centre coordinate to every door node's centre coordinate. We then draw an edge between the area node and the door node with the smallest Euclidean distance. This results in navigable edges as shown in Figure 4.2.

Algorithm 3 Algorithm to link areas to doors

```
1: function DOOR_LINKING($G$ : global graph)
2:     $door\_nodes = n : n \in nodes(G) \mid n.type =$ "Door"
3:     $area\_nodes = n : n \in nodes(G) \mid n.type \neq$ "Door"
4:     for $n \in area\_nodes$ do
5:         $d =$ CLOSEST_DOOR($n, door\_nodes$)
6:         G.ADD_EDGE($n, d$)
7:     end for
8: end function
9: DOOR_LINKING($G_{map}$)
```

4.6 Completion Module

The Completion Module is responsible for forming missing navigable links. We identified that after the Linking Module, we are missing edges, representing navigable links, through door nodes and through connective area nodes. Based on this observation, the Linking Module consists of two submodules to address each shortcoming. In particular, the Door Completion sub-module addresses missing links through doors, while the Area Completion Module addresses missing links through connective areas.

4.6.1 Door Completion

Doors connect two spaces through their front and back. On the basis of this reasoning, we ensure that all door nodes are connected to at least two area nodes. The Door Completion module enforces this reasoning by forming a list of door nodes and a list of area nodes *that are connective areas or shared spaces*, then going through all door nodes and checking how many edges each has. If any door node has less than 2 edges, we go through the list of area nodes and calculate the Euclidean distance to each. We then add an edge between the door node and the closest area node from our list.

Note that we connect doors to connective areas or shared spaces because of our assumption that destination areas do not connect to other destination areas directly. Because destination areas would then connect to connective areas or shared spaces, multiple doors associated with destination areas would lead to connective spaces to bridge these destination areas to connective areas or shared spaces. Thus, connective areas or shared spaces would have edges to multiple doors while destination areas would be connected to 1.

Algorithm 4 Algorithm to complete door links

```
function DOOR_COMPLETION($G$ : global graph)
    $door\_nodes = n : n \in nodes(G) \mid n.type =$ “Door”
    $area\_nodes = n : n \in nodes(G) \mid n.type =$ “Connective”
    for $d \in door\_nodes$ do
        $n =$CLOSEST_AREA$(d, area\_nodes)$
        G.ADD_EDGE$(n, d)$
    end for
end function
DOOR_COMPLETION$(G_{map})$
```

4.6.2 Area Completion

After the Door Completion sub-module, we note that our graph has many isolated components as there are some navigable links missing. From the investigation of various samples, we found that missing links can be traced to two issues. The first issue is that some areas may branch out to converge onto connective areas without passing through a door. The second issue is that we have not made an effort to link connective areas to form our path of convergence points, or connective areas. The Area Completion stage attempts to resolve these two issues.

Firstly, to create a connected path between connective area nodes, we take all the nodes that are connective areas and operate on them as an individual subset. Within this node subset, we connect every node to its nearest two nodes in the subset to link each connective area to the connected path forward and backward (whether this is up and down or left and right). By performing this linking step on every node in our subset of connective nodes, we converge onto a continuous path that links all isolated graph components that contain a connective area node as part of their subgraph. After this step, our graph should be connected by convergence over the numerous edges.

4.7 Indoor Map Output

With the execution of the final stage in our pipeline, we have a graph structure that fully represents the indoor space passed into the pipeline as a PNG image. We argue that this graph is sufficient to serve as our indoor map.

4.7.1 Semantic Representation

The first requirement for our indoor map was that it represents the semantic details of an indoor space in a way that context-aware location systems can easily access. Our graph-based map enables the coordinate location and label of an area to be accessed by simply accessing a node.

We also note that graphs are a common data structure software often uses, so context-aware location systems can easily use and access them.

4.7.2 Navigable Links

The second requirement for our indoor map was that it represents the key navigable links between key areas in the input indoor space, doing so in a way that context-aware location systems can access directly. Our graph-based map enables these links to be accessed by simply traversing the edges of the graph, which are undirected and unweighted.

Chapter 5

Evaluation

We put considerable effort into evaluating the accuracy of our graph-based indoor map output. We first formulate ground truth graphs to evaluate our outputs against, followed by several experiments to evaluate the accuracy of our outputs as a whole as well as variable parameters in our system. We employ two different sets of metrics in our evaluation process. Firstly, we use a set of metrics to evaluate the navigable connectivity represented in our graph. To evaluate this connectivity, we leverage the Dijkstra algorithm that addresses the traditional single-source shortest-path problem for nodes in a graph. Secondly, we use metrics to evaluate the accuracy of the nodes themselves. We compare nodes from our ground truth to generated nodes to determine how well key areas were detected by our pipeline and added to the generated output graph.

5.1 Creating the Ground Truth

Initially, we searched for existing data sets that contain CAD drawings or equivalent floor plans as PNG files, as well as an associated graph that represents key areas as nodes and some form of area connectivity as edges. While we found some research that encoded key areas as nodes [25] in graphs, these graphs encoded distance-based adjacency with their edges, rather than the navigation-based connectivity that we require. Given the lack of appropriate ground truth graphs to evaluate against, we set out to create our own ground truth graphs for key areas within CMUQ.

Our ground truth graphs are created as a JSON object that encodes an array of node objects under the key "nodes" and edge objects under the key "edges" for each graph. Each node object is structured as:

- "id" - specifies id of node
- "name" - specifies label of node
- "x" - x coordinate of node
- "y" - y coordinate of node

Each edge object is structured as:

- "id_1" - id of first node on edge
- "id_2" - id of second node on edge

We created ground truth graphs for three sample subareas within the first floor of the CMUQ building, covering about $22,000m^2$ of indoor space between each of them. The chosen areas cover about $\frac{1}{4}$ of the first floor. Note that we chose areas within CMUQ that represent structural variations that pose significant challenges in generating the output graph. For example, we chose the space around the CMUQ majlis stage as it contains a curved corridor with rooms and doors attached at multiple angles. The curved structure can pose a significant challenge, as it requires detection models to pick up relevant objects at slanted angles. Similarly, we chose the area spanning from the ARC to Student Affairs because it has large hallways that could cause difficulties in edge convergence even with multiple hallway labels. Finally, we chose the area spanning from the West Entrance to the ARC as its long hallway of multiple clustered offices could cause our detection models to miss out areas and doors that are very close together. We provide a summary of our chosen areas in Table 5.1, Table 5.2 and detailed images of our chosen areas in Section 7.

Testbed Name	Approximate Actual Area (m^2)	Number of Destination Areas	Number of Doors
West Walkway	5000	44	49
Student Affairs	7200	58	70
Majlis	10,000	39	43

Table 5.1: Summary of Area Properties for Each Testbed

Testbed Name	Number of Paths	Shortest Path	Average Path	Longest Path
West Walkway	132	0	2475.7	5895
Student Affairs	181	0	3163.9	8249
Majlis	118	0	3663.9	6865

Table 5.2: Summary of Paths Properties for Each Testbed. Here, we use the Total Euclidean Distance along the path to describe path lengths

To create our ground truth graphs, we created a JavaScript-based annotation software that allows a user to upload a PNG image of a CAD drawing through the web browser interface. Once the PNG image is uploaded, users can enter edit mode and then click on different points in the image to add a node at that point. Existing nodes can be double-clicked on to add area labels, and a user can drag their mouse between two existing nodes to add an edge between them.

We note that the existence of our annotation software does not overshadow the value of our mapmaking pipeline. This is because creating

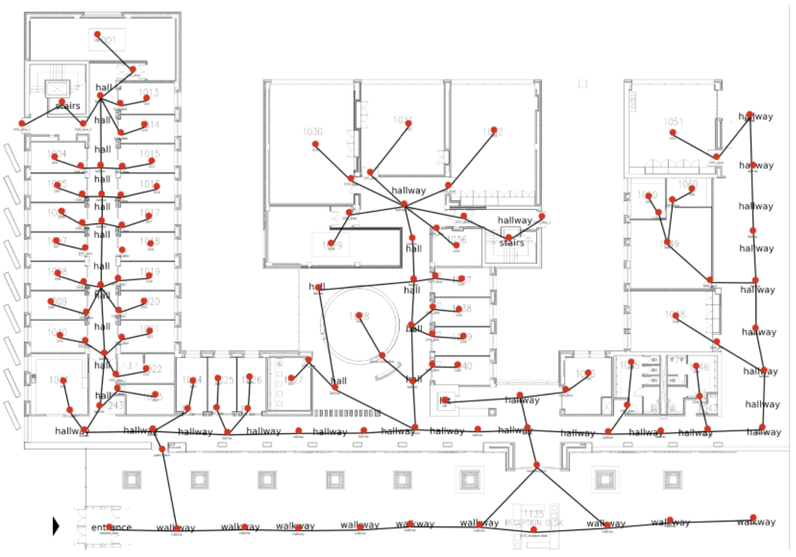


Figure 5.1: Sample Annotation Made on the West Walkway Area

graphs representing indoor spaces with our annotation tool requires a certain level of expertise and time investment to create accurate graphs for the indoor space being represented. Even if a basic user can mark key areas by simply adding nodes at the location of text labels in the PNG image, understanding of graph structures and how their edges must exactly represent connectivity between areas is essential to correctly add navigable edges between placed nodes. We also note that adding appropriate labels to each node can consume a large amount of time. For example, an expert user from our team took 30 minutes to annotate a $7200m^2$ area within the CMUQ building (this area ranged from the ARC to the end of Student Affairs, down to the West Walkway). A non-expert user would reasonably take much more time to perform the same task.

5.2 Evaluation Metrics

Using our established ground truth, we evaluated the graphs produced by our pipeline using two different categories of evaluation metrics.

5.2.1 Shortest Paths Similarity

Looking back at our motivation, we aim to create an indoor map that can enable context-aware systems to operate indoors with location-based information. Based on this end goal, rather than evaluating the structure of our output graph as a whole, we aim to evaluate the indoor information potential our graph provides. We explore this information potential by evaluating the quality of the navigation paths our graph gives us because navigation paths contribute the most information about the structural pathways relevant to human movement. In particular, many context-aware systems leverage location information to track movement on maps, making the paths that can be used to move very important to systems that use our graph output as their map. To perform all our shortest paths evaluations, we pick an arbitrary, common node between our output and ground truth graphs and calculate single-source shortest paths from this common node in each graph.

Distance-Based Shortest Paths Similarity

We first evaluate our generated navigation paths obtaining the Euclidean distance across each shortest path. By using Euclidean distances, we get a sense of the lengths of our paths relative to the actual distance in the represented indoor space. In considering actual path lengths, we can judge just how efficient our generated paths are in representing navigation.

We accumulate the euclidean distance by taking the euclidean distance between the coordinates of nodes u and v for each edge (u, v) traversed in our path. Based on the total Euclidean distance of each path, we report the average percentage match of total euclidean distance between the generated graph's path compared to the ground truth graph's path. Our average is taken across the shortest paths from the common source node s to each destination area node. Note that we only include paths to destination area nodes because destination area nodes mark out the key areas that people would want to visit.

Semantics-Based Shortest Path Similarity

We note that, by only considering the total Euclidean distance across the shortest paths in our graph, we disregard the fact that some navigable edges may go through walls, rendering them illegal. To evaluate the correctness of our paths rather than just the length efficiency, we introduce a second criterion where we compare the nodes present on each ground-truth path to the nodes present on the equivalent path in our output graph. In essence, for each path from a common source to each destination area node, we check the percentage of nodes in the ground truth path that have a matching node in the output path. We then report the average number of matching nodes.

Metric Formalisation

We formalise the Path Similarity Metrics as follows:

Note that a path from a node v_1 to a node v_n refers to a sequence of vertices $P =< v_1, v_2, ..., v_n >$ such that every v_i and v_{i+1} for $1 \leq i \leq (n-1)$ shares an edge. We say that such a path has length n and has a total weight equal to the sum of the edge costs between each v_i and v_{i+1}

We first find all the shortest weighted paths starting from a source vertex s to all *destination* nodes in our ground-truth graphs, then we do the same thing for our predicted graphs starting from the same common source vertex s. Based on that, we define two metrics: **Distance-Based Shortest Paths Similarity** and **Semantics-Based Shortest Path Similarity**. Here, the first metric measures our system's performance in predicting **efficient** paths, and the second metric measures our system's performance in predicting **correct** paths.

Now, Let $SP_{gt} =< P_{gt1}, P_{gt2}, ..., P_{gtM} >$ be a set of all shortest paths starting from node s in the ground-truth graph, where M is the number of destination nodes in the building. Let $SP_{pr} =< P_{pr1}, P_{pr2}, ..., P_{prM} >$ be a set of all shortest paths starting from vertex s in the predicted graph.

Distance-Based Shortest Paths Difference is defined as:

$$Sim_{distance}(G_{gt}, G_{pr}) = 1 - \frac{\sum_{i=1}^{M} \frac{|\Delta(P_{pri}) - \Delta(P_{gti})|}{\Delta(P_{gti})}}{M} \tag{5.1}$$

where $\Delta(P)$ is the distance in metres of the path P.

Semantics-Based Shortest Path Similarity as:

$$Sim_{semantic}(G_{gt}, G_{pr}) = 1 - \frac{\sum_{i=1}^{M} \Pi(P_{gti}, P_{pri})}{M} \tag{5.2}$$

Where $\Pi(P_i, P_j)$ is the number of nodes in the path P_i that do not have a matching node in P_j.

5.2.2 Node Matches

Having looked into the structural information that our graph gives, we next verify the semantic information our graph provides by evaluating the correctness of the nodes present in the graph. We separately consider the destination area nodes in our graph as these mark the key areas that people would want to go to, and the door nodes in our graph, as these provide an indication of how the door model we trained through fine tuning is performing in door detection.

We formally define Node Matches as the problem in which given two sets of vertices $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$, the percentage of matching vertices represents the number of u_i that have an equivalent vertex v_j, where i may or may not be equal to j.

A vertex is a match if:

- For all node types: they have a Euclidean distance less than 2 meters.
- For destination area node types: they additionally have the same label.

Let $d(u_i, v_j)$ denote the Euclidean distance between the vertices u_i and v_j. The percentage of matching vertices can be calculated as follows:

$$\text{Matching Percentage} = \frac{\text{Number of matching pairs}}{n}$$

Where:

- n is the total number of vertices in U.
- The number of matching pairs is the count of pairs (u_i, v_j) where $d(u_i, v_j) < 2$, for at least one pair.

Mathematically, the number of matching pairs can be expressed as:

$$\text{Number of matching pairs} = \sum_{i=1}^{n} max_{1 \leq j < n} \delta(d(u_i, v_j) < 2)$$

Where $\delta(x)$ is the Kronecker delta function, defined as:

$$\delta(x) = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{if } x \text{ is false} \end{cases}$$

Therefore, the percentage of matching vertices can be formally expressed as:

$$\text{Matching Percentage} = \frac{1}{n} \sum_{i=1}^{n} max_{1 \leq j < n} \delta(d(u_i, v_j) < 2)$$

5.3 Evaluation on Different Locations

We evaluate each of our 3 testbed areas with both categories of metrics.

From Figure 5.2, we see that our pipeline accuracy is fairly consistent among our three testbed areas, with more than 90% average matching for the euclidean distance of the paths. Although the Semantics-Based Shortest Paths similarity is also consistent at around 70%, we note that it is much lower compared to the average Euclidean distance matching for paths. From this result, we conclude that while we can represent a good approximation on physical distance, we have a fair number of illegal edge instances because we obtain edges by attempting convergence across our series of modules and make no attempt to verify information like the presence of walls. Thus, while the correctness of our paths is not especially poor, there remains much room for improvement.

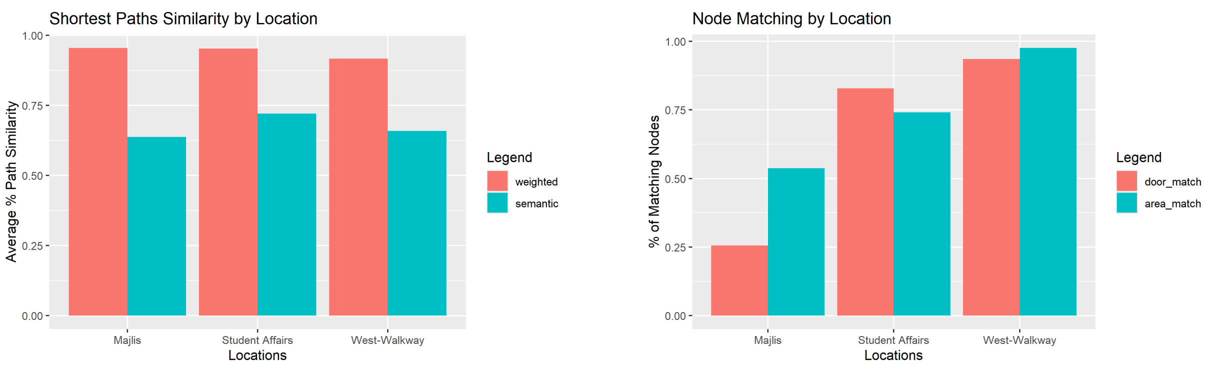


Figure 5.2: Left: Shortest Path Similarities for different testbed areas. Right: Node Matching Percentages for different testbed areas

Despite some accuracy issues related to correctness, we note that our pipeline is operating on significantly large areas, such as the Majlis testbed being $\sim 10000m^2$. Thus, we can accurately report on at least 70% of such large spaces, which amounts to a large amount of physical space.

We see significant variance in node detection among our three testbed areas. Firstly, we note that there is a significantly lower match percentage in the Majlis testbed area. From this, we can understand that our pipeline stages relying on detection models (i.e. the Extraction Module) to identify key areas and doors struggle when faced with curved spaces. We reasonably pick out curved spaces as a challenge because it is the most different feature that the Majlis testbed area has compared to our other two testbed areas.

Next, our Student Affairs testbed area has a significantly more similar matching percentage with the remaining West Walkway testbed area compared to the Majlis testbed area. However, there is still a noticeable difference in the percentages of node matching. In node detection, we reason that open spaces should not create noise in simply detecting labels and doors. Other than its open spaces, the Student Affairs area does not present significant structural differences compared to the West Walkway area. Instead, we argue that the Student Affairs testbed area has lower node matching percentages in both doors and areas simply due to its larger size, it covering $\sim 7200m^2$ of physical space while the West Walkway covers $\sim 5000m^2$. In this way, we see that there is some decline in accuracy for area and door extraction within our pipeline as the indoor space being operated on increases in physical size. However, we argue that, compared to the 44% increase in physical area, our accuracy decrease is not as significant as it may seem.

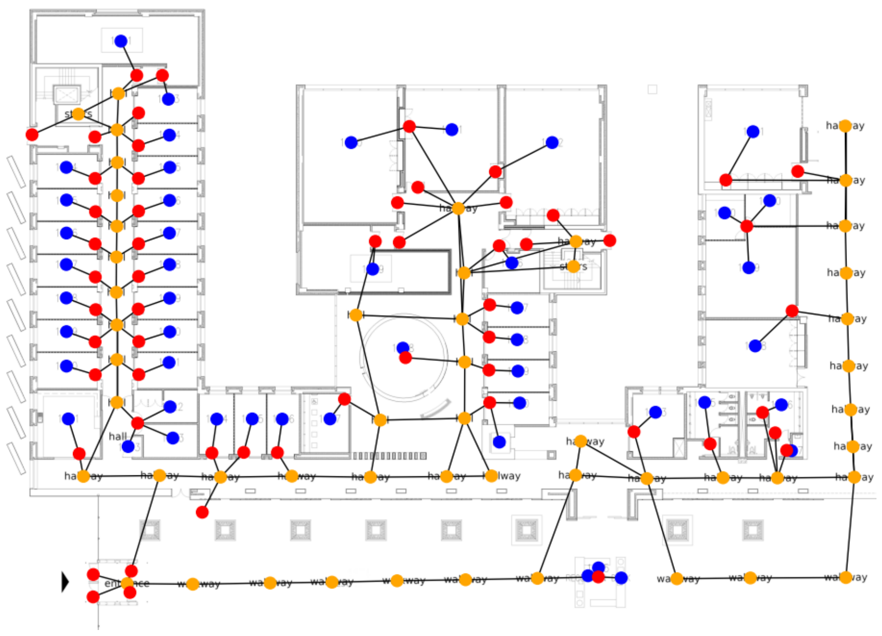


Figure 5.3: Generated Graph for the West Walkway Area of CMUQ

5.4 Experiments with Pipeline Parameters

Having evaluated our output graphs on different testbed areas, we then move onto investigating key parameters that our system relies on. By running evaluation experiments on these parameters, we hope to find the optimal value to increase the accuracy of our pipeline. The key parameters we investigate are Pixel to Distance ratio, Overlap Percentage between Image Slices from the Slicing Module, and the Confidence Threshold for a detected door to be taken as a node.

5.4.1 Pixel to Distance Ratio

We previously stated that we assume that we will receive a PNG image representing an indoor space with a mapping of 50 pixels per metre. Our pipeline relies on this mapping to ensure the actual area space of image slices (which are fixed at 1000 x 1000 pixel images) is a significant enough portion to not waste computation only detecting one or two “objects” (i.e. the doors or areas) per slice, while also not clustering too many detection “objects” into one slice. We additionally rely on this mapping to determine when to merge nodes during slice merging.

We want to balance the pixel to distance ratio such that the image matching this mapping is not too large, as this will lead to slow computation from operating on a large number of slices, but not too small such that

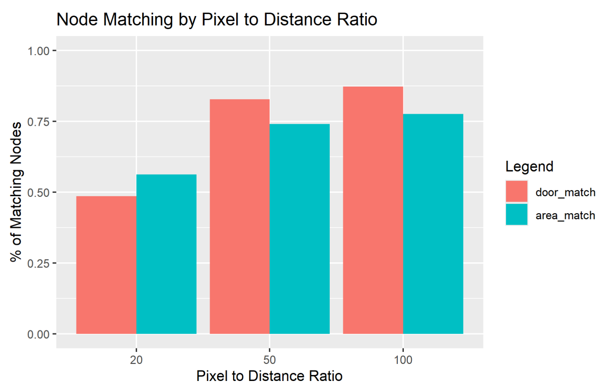


Figure 5.4: Bar charts showing our Node Matching Metrics for different Pixel to Distance Ratios

detection of our "objects" becomes too difficult for our detection models in the pipeline's Extraction Module.

Given the nature of exporting PNG images from CAD drawings, testing Pixel to Distance ratios at small intervals posed a significant challenge. Due to this challenge in matching exact image dimensions, we test 3 pixel to distance ratios of significant intervals: 20 pixels per metre, 50 pixels per metre, and 100 pixels per metre.

As we can see from Figure 5.4, the percentage of matching nodes, in both doors and destination areas, significantly drops for the 20 pixels per metre ratio. Thus, 20 pixels per metre is not an appropriate mapping to use. We then argue that the difference in percentage of matching nodes between the 50 and 100 pixel per metre ratios is small enough to justify a selection of 50 pixels per metre to reduce the number of slices formed to cover the whole image.

Image Size Conversions

We formalise the calculation of the dimensions of the PNG image based on our pixel to distance ratio as follows:

> Let A be the actual area in square metres. We use a pixel-to-distance ratio of 50 pixels per metre. Thus, the dimensions of the input image, with a high A0 resolution (150 or 300 dpi), are given as follows from the actual area:
>
> 1. The width (W) of the image is the square root of the actual area times 50^2:
> $$W = \sqrt{A \times (50^2)}$$

2. The height (H) of the image is W times the aspect ratio of the area:

$$H = W \times \text{aspect ratio}$$

Where W is the width of the image, H is the height of the image and the aspect ratio is the ratio of width to height of the actual area.

5.4.2 Overlap Percentage of Image Slices

Continuing our optimisations for the Image Slicing Module of our pipeline, we next look at the percentage of overlapping area between vertically and horizontally adjacent slices. Note that we take some overlap between slices to reduce areas or doors that are not detected because they are cut off at an image boundary. However, having a very high overlap percentage significantly increases the number of slices formed for a single image, which in turn increases the computational load (e.g., execution time, memory, etc.). We aim to balance the accuracy achieved from the slice overlap percentage while minimising the percentage as much as possible.

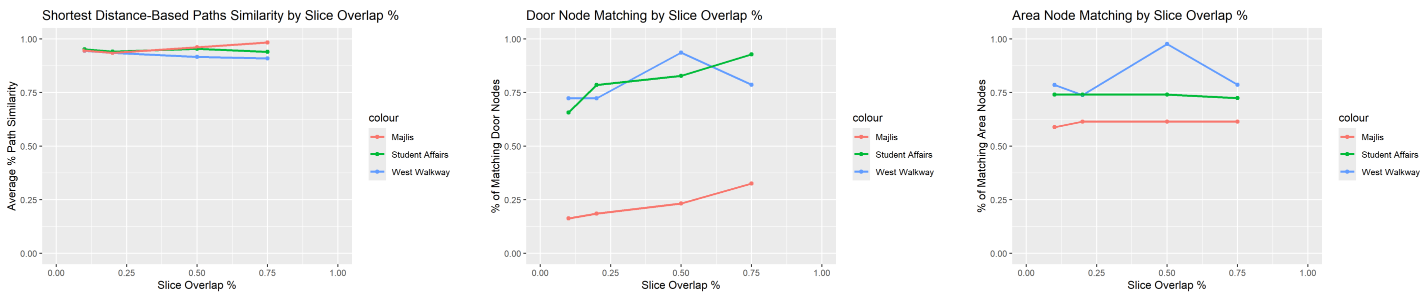


Figure 5.5: Left: Shortest Distance-Based Path Similarities for different Overlap Percentages in Image Slices. Middle: Door Node Matching Percentage for different Overlap Percentages in Image Slices. Right: Area Node Matching Percentage for different Overlap Percentages in Image Slices

First, we look at the percentage of matching nodes for different slice overlaps to see if smaller overlaps do indeed trigger many cut-off issues. From Figure 5.5, we see that the number of matching door nodes increases significantly as the percentage of overlap increases. However, we note that it falls on a 75% overlap for the West Walkway. It is possible that the overlap cutoffs at 75% triggered extra doors to be placed at slice boundaries, causing a drop in matching door nodes due to the structural layout of the West Walkway testbed area.

For area node matching, we see that the slice overlap has a much less significant impact on the percentage of matching area nodes. We reason that this could be due to the size of area text labels in our input images compared to the size of doors. Because our text labels are generally much wider than doors, it is less likely that an area label is undetected completely just because it was placed at a boundary. In the case of boundary labels, the area will be at least partially detected, especially considering the high performance of OCR detection models in research today.

Next we look into Path Similarity based on overlap percentages. Looking at the Shortest Distance-Based Path Similarity, as the overlap percentage would not directly affect the semantics of the paths beyond its impact on door and area detection, we also notice that the overlap percentage has little effect on the Shortest Distance-Based Path Similarity. We reason that this is likely due to the fact that we derive our edges by convergence based on the coordinates of detected doors and areas; thus our overlap percentage would not affect paths beyond its impact on node detection.

5.4.3 Door Confidence Score Threshold

Finally, we look at the minimum confidence score that a door within the input PNG image, by our door detection model, must receive to be taken as a node. This is crucial to our pipeline as we depend on doors to make edges to destination areas. If many false doors are detected, we may get invalid edges through the walls. In contrast, if many doors are missed, we may get destination areas that cannot correctly connect to the rest of the graph.

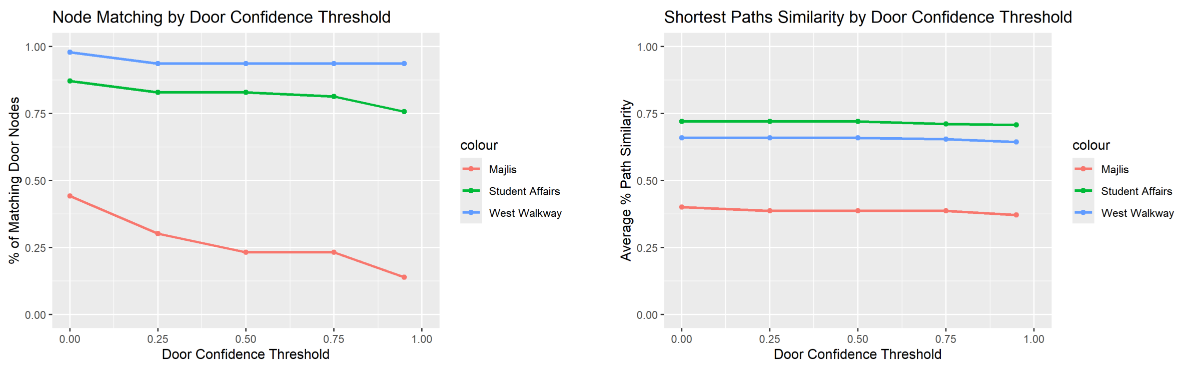


Figure 5.6: Left: Node Matching Percentages for different Door Confidence Thresholds. Right: Shortest Path Similarities for different Door Confidence Thresholds

Looking at the percentage of matching door nodes, it is clear that increasing the confidence threshold decreases the percentage of matching nodes. This is reasonably because fewer doors would have a higher confidence threshold. However, we note that for the West Walkway and Student Affairs testbed areas, the drop in the matching percentage is not very significant from 0 to the 0.5 confidence threshold. This is not the case for the Majlis testbed area, but the results are most likely conflated by the difficulty in detecting doors in its curved hallway.

Next we look at the Semantics-Based Shortest Paths Similarity. We reason that this is important to consider as many of our edges to destination area nodes are through doors. However, we see that the Semantics-Based Shortest Paths Similarity does not change much with an increase in the confidence threshold. Based on this observation, we reason that the additional doors kept by lower confidence thresholds are most likely invalid doors that are not meant to be present on the semantic shortest path.

Chapter 6

Conclusion

In this thesis, we presented a new mapmaking process capable of producing a graph-based indoor map structure for context-aware systems to leverage for accessing indoor-based location information. We iterated that it is essential to design a mapmaking process that is efficient and easy to integrate into existing systems, as no current approach to representing indoor spaces has been sufficiently ubiquitous enough to have been widely adopted in industrial applications.

Our proposed mapmaking process makes use of well-established detection methods that are known to have high-performing models, as well as some object detection models that we fine-tuned, to extract key areas and doors within our indoor space. After completing our extraction process, we build navigable links between our nodes by leveraging connections going from areas to doors, as well as forming the connective area to connective area navigable pathway that usually travels around a building.

6.1 Future Work

Despite the satisfactory results we presented in the evaluation of our pipeline, there are still several key points in the pipeline that could be improved. Firstly, the accuracy of our pipeline's door detection model needs to be improved. Because our pipeline struggled to detect doors in specific structural scenarios such as curved hallways, the door detection model should be improved to better handle these scenarios. By improving door detection, we would not only increase the percentage of matching door nodes from our evaluation metrics but also improve the navigable links derived from destination areas through doors.

Secondly, we point out that our pipeline will not accurately converge for indoor spaces where destination areas connect to connective areas and other areas without passing through doors, such as a doorless shop opening. We believe that further investigation needs to be performed to find a good connective rule for these doorless cases.

Finally, we point out that the semantic paths of our output graphs need to be improved. For this, we propose that a highly accurate method of detecting walls could be used to refine edges based on whether they pass through a wall or not. We note that detecting black pixels is not sufficient for wall detection, as image noise could cause black pixels in open spaces and furniture could be perceived as walls, disregarding edges where a user simply has to walk around a sofa.

Chapter 7

Appendix

7.1 CubiCasa5k Segmentation

Results from experimenting with the indoor space segmentation [15] for SVG's mapmaking process:

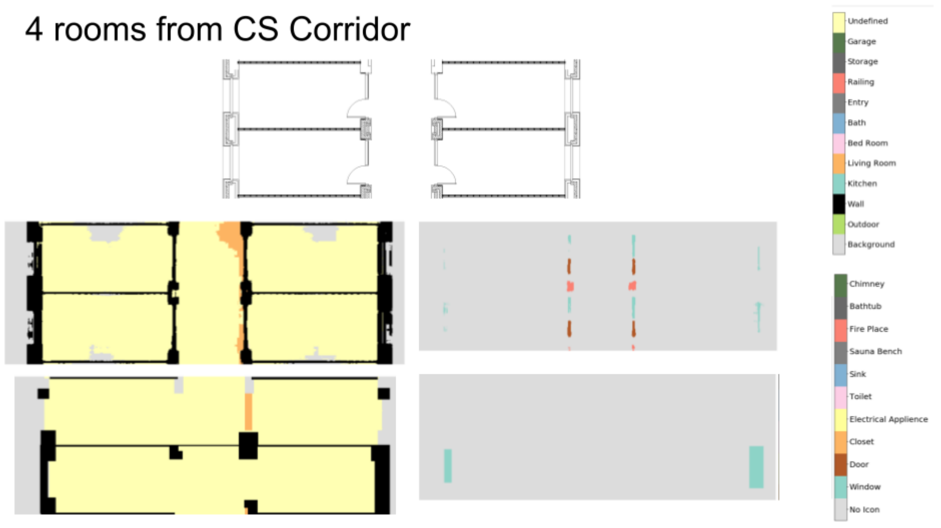


Figure 7.1: Using the model to segment 4 rooms of the CS Corridor. Top: PNG image, Middle-Left: Room and wall segmentation by model, Middle Right: Door and window segmentation by model, Bottom-Left: Room and wall segmentation after post-processing, Bottom-Right: Door and window segmentation after post-processing

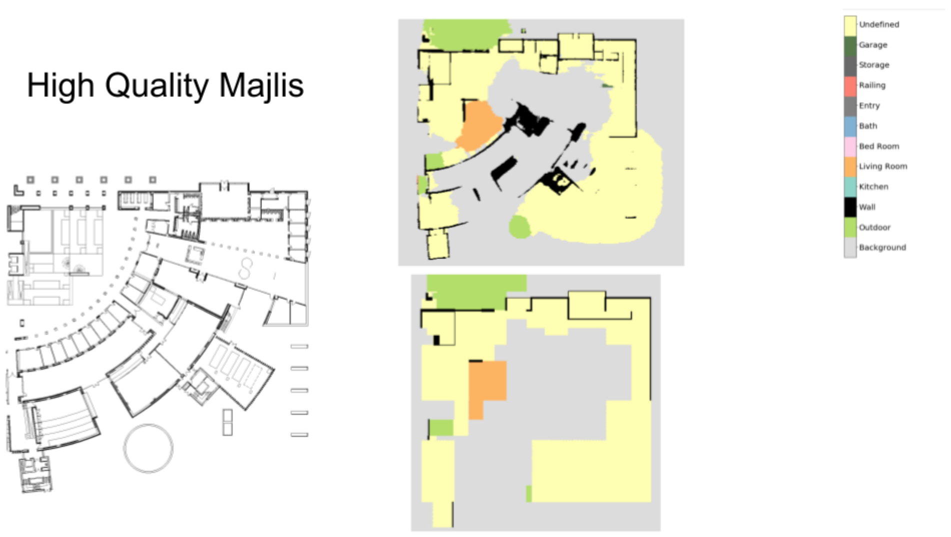


Figure 7.2: Using the model to segment the Majlis Area of CMUQ. Left: PNG image, Top: Room and wall segmentation by model, Bottom: Room and wall segmentation after post-processing

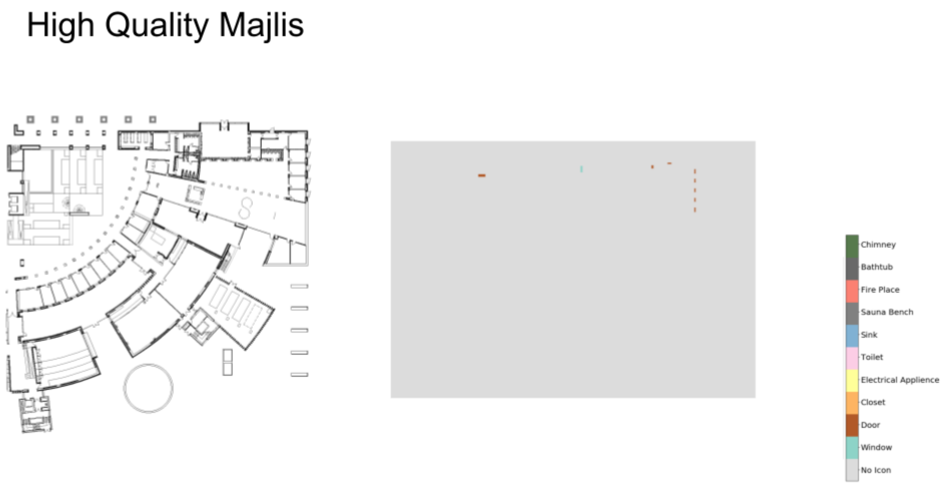


Figure 7.3: Using the model to segment the Majlis Area of CMUQ. Left: PNG image, Right: Door and window segmentation after post-processing

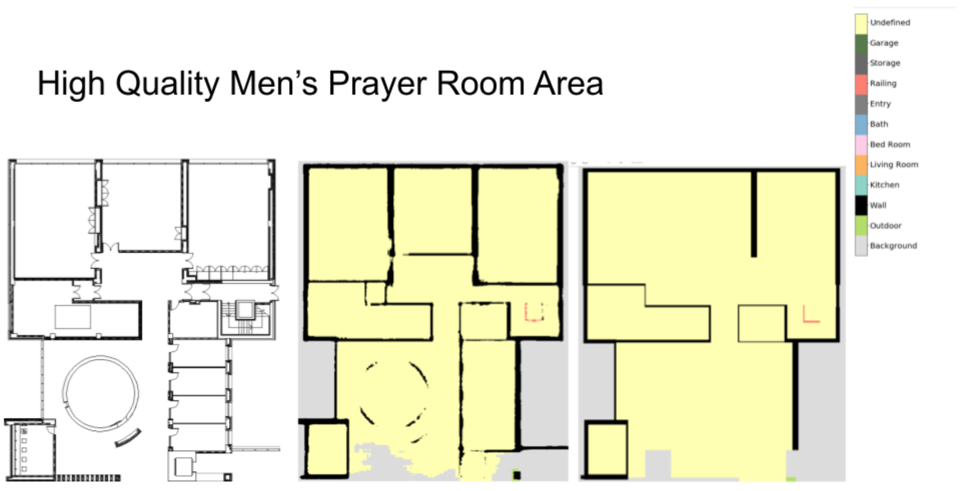


Figure 7.4: Using the model to segment the Men's Prayer Room of CMUQ. Left: PNG image, Middle: Room and wall segmentation by model, Right: Room and wall segmentation after post-processing

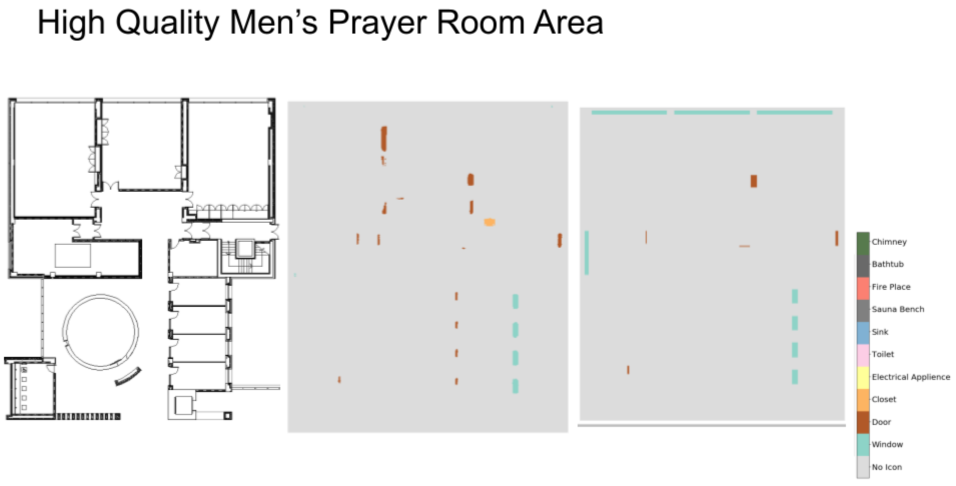


Figure 7.5: Using the model to segment the Men's Prayer Room of CMUQ. Left: PNG image, Middle: Door and window segmentation by model, Right: Door and window segmentation after post-processing

7.2 Ground Truth Testbeds

7.2.1 Raw Images

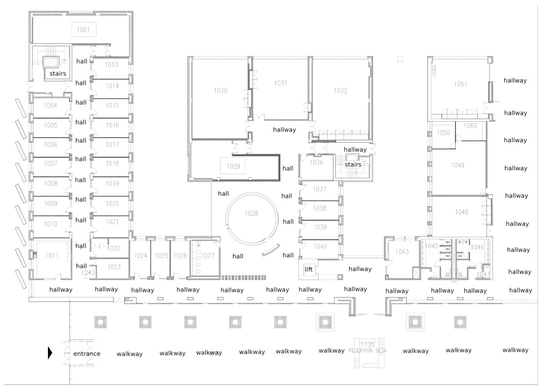


Figure 7.6: An image representing the West Walkway Upper Area

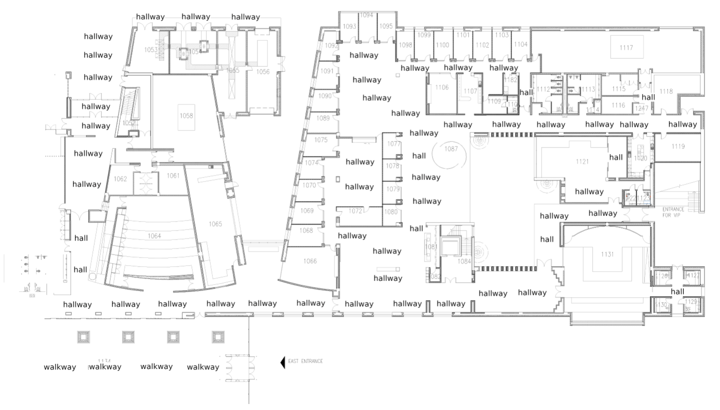


Figure 7.7: An image representing the Student Affairs Surrounding Area

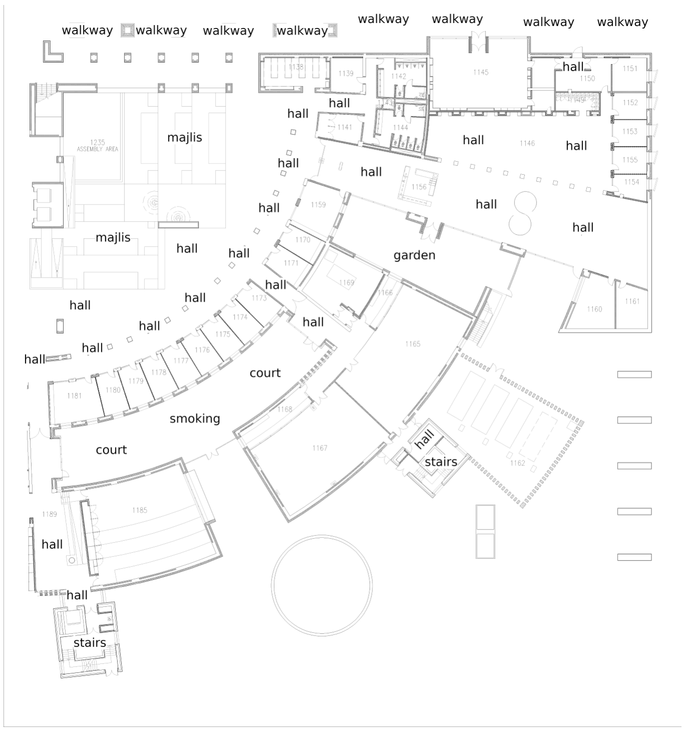


Figure 7.8: An image representing the Majlis Area

7.2.2 Annotated Ground Truths

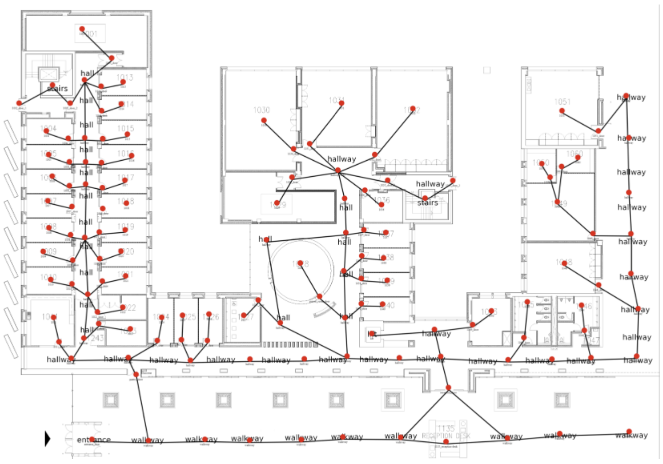


Figure 7.9: An image representing the ground truth graph of West Walkway Upper Area

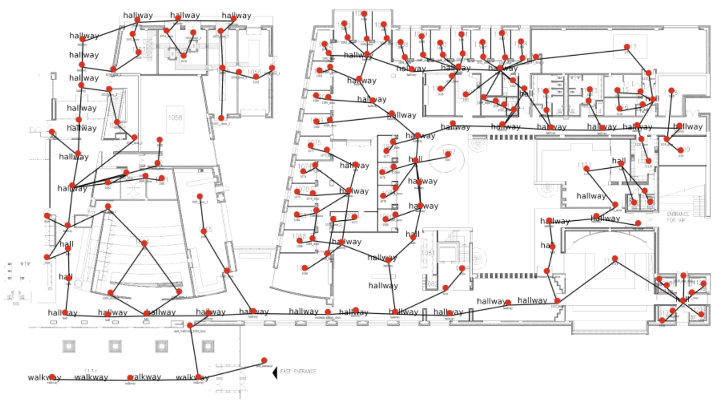


Figure 7.10: An image representing the ground truth graph of the Student Affairs Surrounding Area

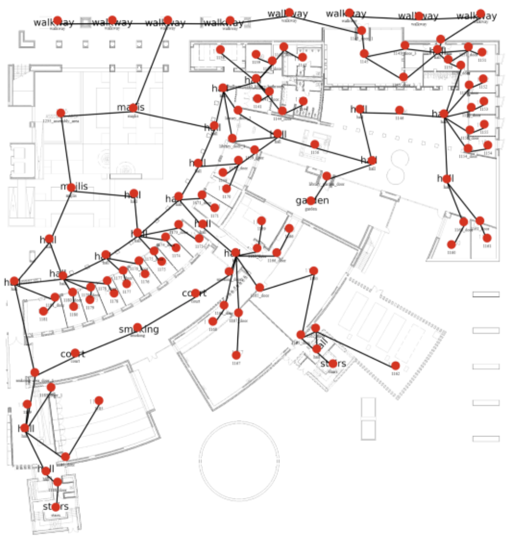


Figure 7.11: An image representing the ground truth graph of the Majlis Area

7.2.3 Pipeline generated graphs

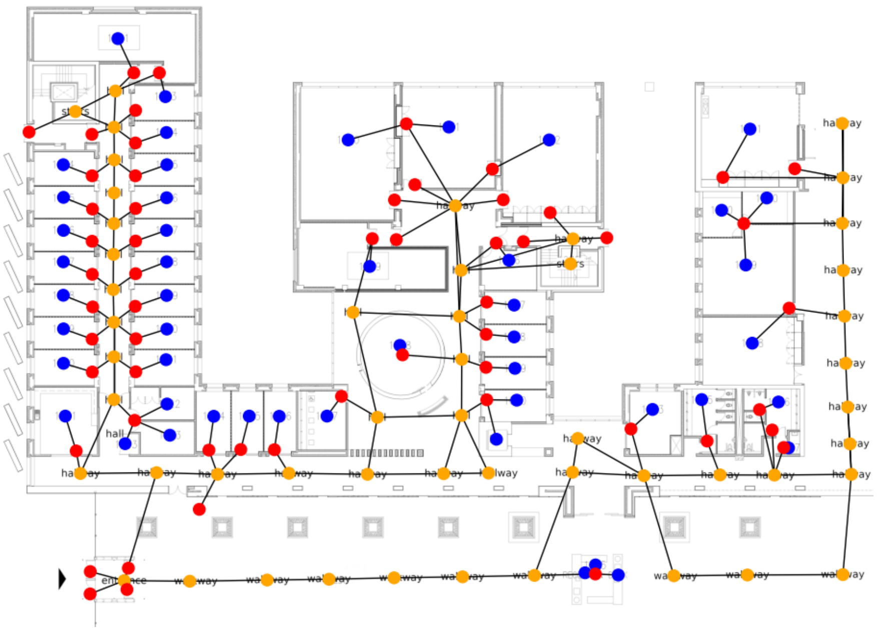


Figure 7.12: An image representing the generated graph of West Walkway Upper Area

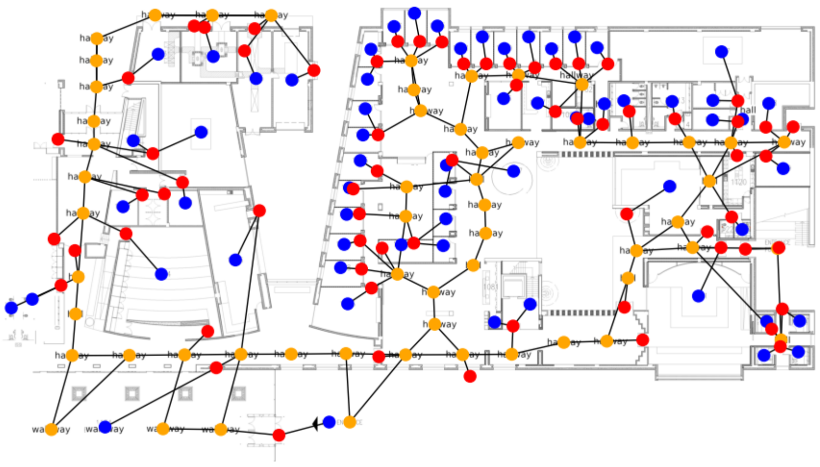


Figure 7.13: An image representing the generated graph of the Student Affairs Surrounding Area

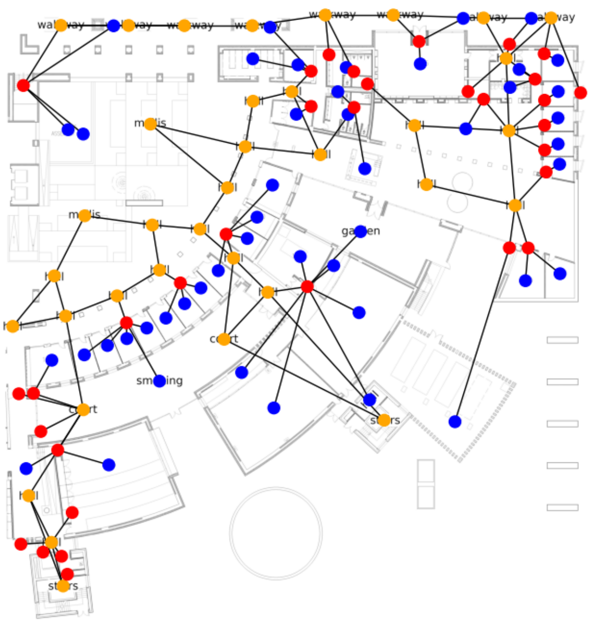


Figure 7.14: An image representing the generated graph of the Majlis Area

Bibliography

[1] Adobe. Learn about dwg files. `https://www.adobe.com/africa/creativecloud/file-types/image/vector/dwg-file.html`. (Accessed on 12/03/2023).

[2] I. Andersson. Indoor positioning systems in office environments-a study of standards, techniques and implementation processes for indoor maps. *Examensarbete i geografisk informationsteknik*, 2020.

[3] Apple. Indoor mapping data format - indoor mapping data format, Oct. 2021.

[4] L. Appolloni and D. D'alessandro. Housing spaces in nine european countries: A comparison of dimensional requirements. *International Journal of Environmental Research and Public Health*, 18(8):4278, 2021.

[5] J. Baek, G. Kim, J. Lee, S. Park, D. Han, S. Yun, S. J. Oh, and H. Lee. What is wrong with scene text recognition model comparisons? dataset and model analysis. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4715–4723, 2019.

[6] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee. Character region awareness for text detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9365–9374, 2019.

[7] W. Chai. Cad (computer-aided design). *Tech Target*, Dec. 2020.

[8] J. Chen and K. C. Clarke. Indoor cartography. *Cartography and Geographic Information Science*, 47(2):95–109, 2020.

[9] R. Daher, T. Chakhachiro, and D. Asmar. From slam to cad maps and back using generative models. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 2979–2985. IEEE, 2022.

[10] L.-P. de las Heras, O. R. Terrades, S. Robles, and G. Sánchez. Cvc-fp and sgt: a new database for structural floor plan analysis and its groundtruthing tool. *International Journal on Document Analysis and Recognition (IJDAR)*, 18:15–30, 2015.

[11] M.-W. Dictionary. Map definition & meaning - merriam-webster. `https://www.merriam-webster.com/dictionary/map`. (Accessed on 03/15/2024).

[12] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13, 2006.

[13] H. Jang, K. Yu, and J. Yang. Indoor reconstruction from floorplan images with a deep learning approach. *ISPRS International Journal of Geo-Information*, 9(2):65, 2020.

[14] D. Johnson. What is a png file? how to open or convert the popular graphic file format. *Business Insider*, Jan. 2022.

[15] A. Kalervo, J. Ylioinas, M. Häikiö, A. Karhu, and J. Kannala. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In *Image Analysis: 21st Scandinavian Conference, SCIA 2019, Norrköping, Sweden, June 11–13, 2019, Proceedings 21*, pages 28–40. Springer, 2019.

[16] J. Lee, K.-J. Li, S. Zlatanova, T. H. Kolbe, C. Nagel, T. Becker, and H.-Y. Kang. Ogc® indoorgml 1.1. `https://docs.ogc.org/is/19-011r4/19-011r4.html`, Nov 2020. (Accessed on 12/05/2023).

[17] K.-J. Li, S. Zlatanova, J. Torres-Sospedra, A. Pérez-Navarro, C. Laoudias, and A. Moreira. Survey on indoor map standards and formats. In *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8. IEEE, 2019.

[18] C. Liu, J. Wu, P. Kohli, and Y. Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2195–2203, 2017.

[19] A. Lookingbill and E. Russell. Google maps 101: how we map the world. `https://blog.google/products/maps/google-maps-101-how-we-map-world/`, July 2019. (Accessed on 03/15/2024).

[20] NESDIS. Can satellites see you? can you see a satellite? *National Environmental Satellite, Data, and Information Service*, Nov. 2017.

[21] N. Saxena. Pytorch: Scene text detection and recognition by craft and a four-stage network — by nikita saxena — towards data science, Aug 2020. (Accessed on 03/22/2024).

[22] STEMLab. Ineditor: A web-based editor for drawing and creating ogc indoorgml data, 2021.

[23] W3Schools. Svg tutorial. `https://www.w3schools.com/graphics/svg_intro.asp`, 2023. (Accessed on 12/06/2023).

[24] M. Yamada, X. Wang, and T. Yamasaki. Graph structure extraction from floor plan images and its application to similar property retrieval. In *2021 IEEE international conference on consumer electronics (ICCE)*, pages 1–5. IEEE, 2021.

[25] T. Yamasaki, J. Zhang, and Y. Takada. Apartment structure estimation using fully convolutional networks and graph model. In *Proceedings of the 2018 ACM Workshop on Multimedia for Real Estate Tech*, pages 1–6, 2018.