TUM Department of Civil, Geo and Environmental Engineering
Chair of Computational Modeling and Simulation
Prof. Dr.-Ing. André Borrmann

Matching construction drawings to Building Information Models using 2D shape distribution

Clément Thiele

Bachelorthesis
for the Bachelor of Science Course Civil Engineering

Author:	Clément Thiele
Student ID:	[redacted]
Supervisor:	Prof. Dr.-Ing. André Borrmann
	M.Sc. Maciej Trzeciak
Date of Issue:	08. November 2018
Date of Submission:	08. April 2019

Abstract

Building Information Modeling (BIM) is on the verge of being incorporated into the construction industry. Even though the advantages of using BIM are apparent, the industry is still mainly based on two dimensional (2D) construction drawings. BIM application have therefore been adapted to derive those drawings from the three dimensional (3D) model. Due to the high amount of parties involved in the building process, there was the need for a vendor-neutral format to accomplish this task. The disadvantage of such a format is the fact that the model and the drawings become disjointed once extraction is completed and, therefore, changes applied to one will not be adjusted automatically on the other. The discrepancies that arise during this process are the source of various possible errors which could be eliminated if the drawings were aligned to each other regularly to ensure their conformity. The automation of this process, also called the registration problem, has faced challenges in the past and an adequate solution for construction drawings is yet to be found.
This thesis focuses on the second step of the registration problem, the similarity measurement and matching, using the concept of 2D shape histograms as proposed by (Pu & Ramani, 2006) in their paper "On visual similarity based 2D drawing retrieval". The method delivered promising results when the drawings had similar contours with different interior features, yet as it does not support partial matching, it proves to be impractical for the purpose of this thesis.
However, two methods that could potentially solve this issue are proposed as possible future solutions. One being the combination of this method with 2.5D spherical harmonics. The other consists in trying to overlap two cross-sections of the same drawing at different heights to bypass the missing capability of partial matching observed on this tested descriptor.

Zusammenfassung

Building Information Modeling (BIM) steht kurz davor, in die Bauindustrie integriert zu werden. Die Vorteile des Einsatzes von BIM sind offensichtlich, dennoch greift die Industrie immer noch hauptsächlich auf zwei dimensionalen (2D) Konstruktionszeichnungen zurück. BIM-Anwendungen wurden aufgrund dessen angepasst, um diese Zeichnungen aus dem dreidimensionalen (3D) Modell abzuleiten. Aufgrund der hohen Anzahl von am Bauprozess beteiligten Parteien bestand die Notwendigkeit eines herstellerneutralen Formats zur Erfüllung dieser Aufgabe. Der Nachteil eines solchen Formats besteht darin, dass die Verbindung zwischen dem Modell und der Zeichnungen nach Abschluss der Extraktion fehlt und daher Änderungen, die auf eine dieser Formate angewendet werden, nicht automatisch auf die andere angepasst werden. Die dabei auftretenden Diskrepanzen sind die Ursache für verschiedene mögliche Fehler. Diese könnten beseitigt werden wenn die Zeichnungen regelmäßig überlagert werden würden, um ihre Übereinstimmung zu gewährleisten. Die Automatisierung dieses Prozesses, auch Registrierungsproblem genannt, stand in der Vergangenheit vor Herausforderungen und auch zum derzeitigen Stand ist eine adäquate Lösung für Konstruktionszeichnungen noch nicht gefunden. Dieses Thesis konzentriert sich daher auf den zweiten Schritt des Registrierungsproblems, die Ähnlichkeitsmessung und -anpassung, unter Verwendung des Konzepts der 2D-Form-Histogramme, wie von (Pu & Ramani, 2006) in ihrem Beitrag "On visual similarity based 2D drawing retrieval". Die Methode lieferte vielversprechende Ergebnisse, wenn die Zeichnungen ähnliche Konturen mit unterschiedlichen Innenausstattungen hatten. Sie unterstützt allerdings keinen Teilabgleich und erweist sich deswegen für die Zwecke dieser wissenschaftlichen Arbeit als unpraktisch.

Es werden jedoch im folgendem zwei Methoden vorgeschlagen, die dieses Problem möglicherweise zukünftig lösen könnten. Eine ist die Kombination dieses Verfahrens mit "2.5D spherical harmonics", welches besser mit unterschiedlichen Konturen umgehen kann. Die andere versucht zwei Querschnitte derselben Zeichnung in unterschiedlichen Höhen zu überlappen, um die fehlende Fähigkeit der partiellen Anpassung zu umgehen, die bei diesem getesteten Deskriptor beobachtet wurde.

Contents

List of Abbreviations

2D	two dimensional
3D	three dimensional
BIM	Building Information Modeling
IFC	Industry Foundation Class
DXF	Drawing Interchange File Format

Chapter 1

Introduction

With the rise of digitisation, the construction industry is facing challenges to incorporate the technological possibilities.
The apparent solution seems to be BIM, which uses digital models throughout the entire lifecycle of a built facility based on three dimensional models with great information depth. This transformation of the building process from separate two dimensional construction drawings to a central, intelligent, three dimensional model results in an improved quality of planning, minimization of errors, and therefore increased product quality as well as lower production costs. (Borrmann *et al.*, 2018)

Even though the advantages of such an evolution in the building process are blatant, the construction industry is still mainly based on 2D construction drawings. Reasons for this are the alleged lack of efficiency in using BIM, especially for smaller projects and that 2D construction drawings still represent the legally binding document. (Trzeciak, 2018)
For BIM to replace traditional construction drawings, the path needs to be paved by incorporating it as a mandatory part of public projects, as is the case in some countries (Borrmann *et al.*, 2018), as well as establishing the legal basis. As long as this is not achieved, traditional construction drawings and BIM models will have to coexist.
As it is possible to derive two dimensional drawings from a BIM model, and changes made to either of them will be applied to the other, the problem seems negligible.
However, these features are only applicable to the native formats of each vendor. As exchanges between parties are necessary, the use of a vendor-neutral format becomes unavoidable. The model and the drawing become therefore disjointed, which leads to discrepancies between the construction drawings and the BIM model while editing on third party applications. These changes cannot yet be detected automatically. (Trzeciak, 2018)
Part of this automation problem is called the registration problem, which describes a three step aligning process of two drawings in a shared coordinate system.

The purpose of this thesis is to test a method which could help with the second step of the process, similarity measurement and matching, in the context of matching a construction drawing and a BIM model to each other.

Chapter 2

State of the art

2.1 Building Information Modeling

As digitalization is getting omnipresent in the last years, the construction industry needs to adapt to it. This could lead to a significant quality improve in the industry with the help of BIM.

BIM is an idea that reaches back to the 1980s, yet the techniques required were only developed in recent years and is now finally seeing use in the construction industry. The main concept is the use of digital information over the entire lifecycle of a facility as shown in figure 2.1. This includes design, construction, maintenance of the facility in use, and when necessary renovation or recycling. (Borrmann *et al.*, 2018)

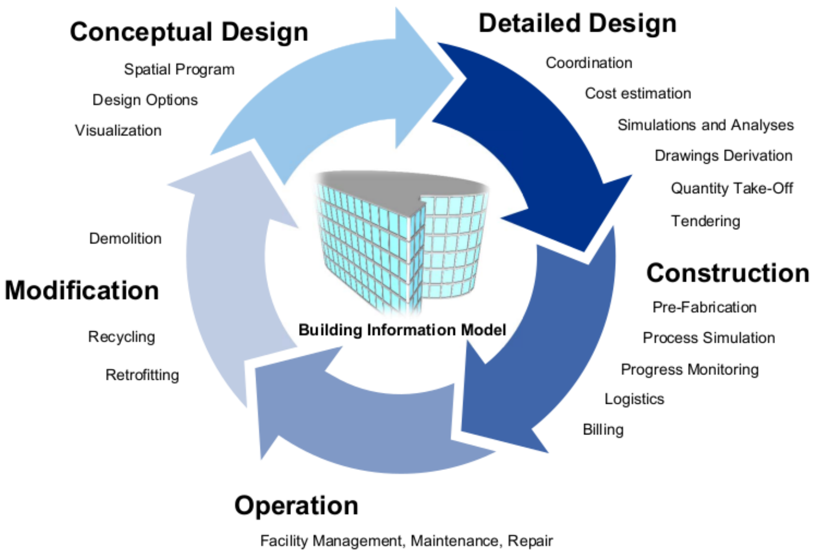


Figure 2.1: Lifecycle of a built facility as shown by (Borrmann *et al.*, 2018)

This process differs immensely from the current way, where the state of the art is the hand over technical, 2D construction drawings between stakeholders, which are a digital imitation of the antiquated drawing board method. Only graphical entities, such as lines, arcs and circles (Azhar *et al.*, 2008), are transmitted with this method, which computers fail to understand and process. A method solely based on this process inevitably misses to harness the potential which comes with the information technologies, such as calculations and simulations, or any other kind of analysis of the built facility. Furthermore, to extract the information required by any stakeholder or the building owner after the end of the construction process, and then transmit it to any application, the effort required is considerable. (Borrmann *et al.*, 2018)

This is where BIM presents a solution, the Building Information models, an object-oriented digital 3D model, as it incorporates, next to the basic outlines of a construction drawing, every information, properties (such as material,...) and purpose within the structure (such as a horizontal member on level X, spanning between column Y and girder Z), about each element included in the drawing, needed for later processing. (Schinler & Nelson, 2008)
With such a depth of information transmitted, the whole process of data re-entry at every stage of a built facility lifecycle, and the loss of information linked to it, is reduced to a minimum. As a result of reducing the time invested in this error-prone work, a significant melioration in the project delivery processes and work flow management, and therefore improved productivity and quality produced by the industry, is expected. (Borrmann *et al.*, 2018) (Hardin & McCool, 2015)

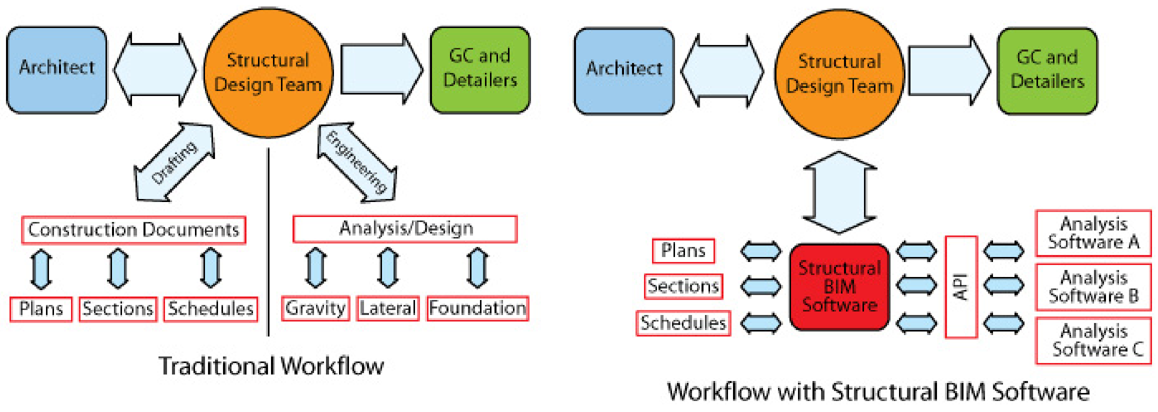
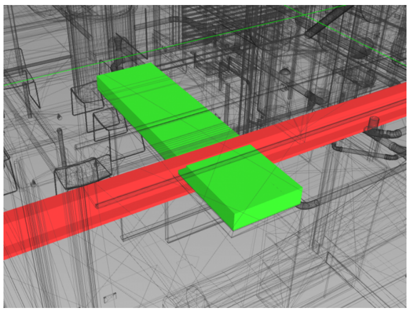


Figure 2.2: Comparison of traditional and BIM workflow structure as shown by (Schinler & Nelson, 2008)

The three dimensional geometry allows to derive, with the help of semantics, consistent sections which can be used as construction drawings compliant to norms and regulations or as only one model is used by all parties, it is possible to perform a general clash detection over the whole model. In the event of two elements from different categories, like a structural and an architectural one, are located in the exact same spot of the facility, the model will recognise and warn the user while displaying the error as in figure 2.3a. This process had to be done manually beforehand and, as this step is required after every change applied to any involved construction drawing, was very time consuming and an error-prone process that is one of the major causes of poor documentation (Azhar *et al.*, 2008).


(a) Clash detected and visually marked. Image courtesy (Lenihan, 2015)

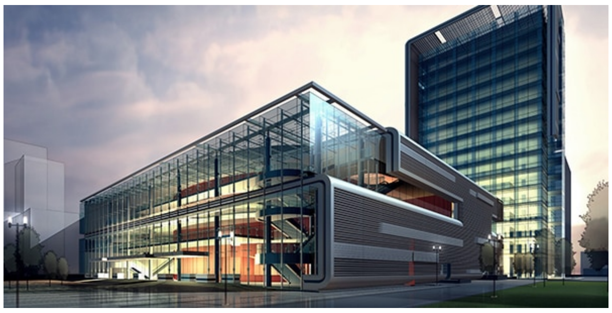

(b) Photorealistic render created by (Autodesk, 2019)

Figure 2.3: Clash detection and visualization obtained from a BIM model

Another advantage of the 3D model is the simplicity with which a realistic visualization can be obtained. In contrast to previous effort needed to obtain a presentable image, the models can now easily be rendered as shown in figure 2.3b. 3D-rendering is a method used to create a realistic picture based on a model, including shadows and photorealistic details in the case of cloud rendering where enough computational power is provided to do so.
This visualization process is mainly used by architects to present a project as it is more palpable and easily understood when compared to 2D construction drawings.

Furthermore, the semantics, which are inherent to each element, allows to perform any kind of analysis, including cost estimation, quantity take-off, structural, analysis or building performance simulations (Borrmann *et al.*, 2018).

Some kind of analysis have not been possible before BIM, like automatized construction scheduling which has been attempted before yet the problem of data extraction remained an issue (Kim *et al.*, 2013). Based on the data depth contained in BIM, it is now possible to generate activities and their duration. These can then be sequenced into a preliminary schedule which can be refined later on (Kim *et al.*, 2013).

Although the actual content of BIM depends on the usage, typically it includes visualization, design coordination, drawing generation, quantity take-off, progress monitoring and facility management (Borrmann *et al.*, 2018).
All these steps help to reduce costs and risks altogether in comparison to the ordinary approach to construction(Borrmann *et al.*, 2018).

2.1.1 BIM adoption

For BIM to be broadly adopted, the need of a vendor-neutral data format became apparent, because an exchange between different products will always be necessary in the context of BIM. Therefore Industry Foundation Class (IFC) has been created by an international non-profit organization, and in 2013 adopted to ISO standards. (Borrmann *et al.*, 2018)

In some regions of the world, like Singapore, Finland, Korea, the USA, UK and Australia, the adoption of BIM into the market is quite advanced, as most of them are integrating BIM as a part of their public projects (Borrmann *et al.*, 2018). Germany is trying to pave the path for this objective, with the help of the working group "BIM4INFRA2020", until 2020 (BIM4INFRA2020, 2017).

For this goal, in Germany as well as anywhere else, there is the need for BIM to replace construction drawings as the legally binding document (Trzeciak, 2018).
A template, called the BIM protocol, which defines the applied terminology as well as global responsibilities (Borrmann *et al.*, 2018), has been proposed by the British Construction Industry Council to ease this path.(Borrmann *et al.*, 2018)

Construction drawings are still often extracted from the models and then directly modified or additional information is added outside of the BIM model. This results in discrepancies between the drawings, which currently cannot be detected automatically and must be adjusted by hand, and, therefore, in errors during all steps of the facility built. (Trzeciak, 2018)

This leads directly into the next section as the shape registration can be used to find theses differences between the model and the drawing.

2.2 Registration problem

The idea of shape registration is to align two shapes in an identical coordinate system (Trzeciak, 2018).
The problem has mostly been approached as a model-to-image registration which, in most cases, showed promising results in a controlled environment but while trying to generalize the process all encountered severe difficulties (Jung *et al.*, 2016).
(Trzeciak, 2018) proposed to approach the problem as a drawing-to-model registration, in which the problem is translated from a 2D to 3D matching problem into a 2D to 2D one as shown in figure 2.4.

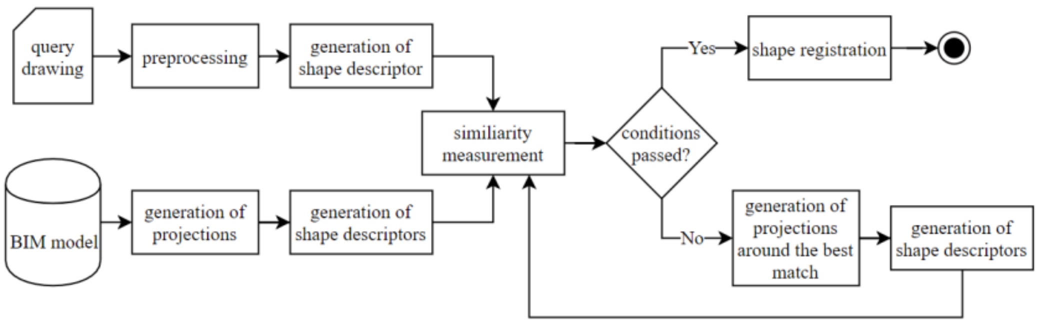

query drawing
preprocessing
generation of shape descriptor
similiarity measurement
conditions passed?
Yes
shape registration
No
BIM model
generation of projections
generation of shape descriptors
generation of projections around the best match
generation of shape descriptors

Figure 2.4: Proposed system for the drawing-to-model registration as shown by (Trzeciak, 2018)

Both approaches have a general three step process in common. They consist of feature extraction, similarity measure and matching, and transformation. (Trzeciak, 2018) (Jung *et al.*, 2016) (Avbelj *et al.*, 2014).

Feature extraction

The first step of the registration problem is the feature extraction. This step consists of detecting the main features of a drawing, such as closed outlines of an object, corner, edges, or even intersection points. These features are then saved in two distinct datasets, one for each image. (Jung *et al.*, 2016)
The main problem during this step is to find out which features need to be extracted, as they need to give the right information about the drawing for the similarity measures and the matching process to work properly (Trzeciak, 2018).

Similarity measure and matching

This is the main problem addressed in this thesis, as the shape histogram method is trying to match the datasets extracted in the previous step.
To find out if two drawings are identical, or partially identical, it is necessary to perform a similarity test on the two sets of data obtained from the feature extraction step with a dissimilarity measure (Tangelder & Veltkamp, 2004). To be able to perform the measure, the data extracted needs to be transformed into comparable descriptors. These can be feature based, shape based (as shown in figure 2.5), and more (Tangelder & Veltkamp, 2004), meaning that the original shapes are translated into feature vectors (Trzeciak, 2018).

The distance obtained during this measuring process gives an idea about the match of the two drawings. The smaller the distance the closer the two drawings are to each other. (Tangelder & Veltkamp, 2004). Though not being mandatory, the distance between two drawings is

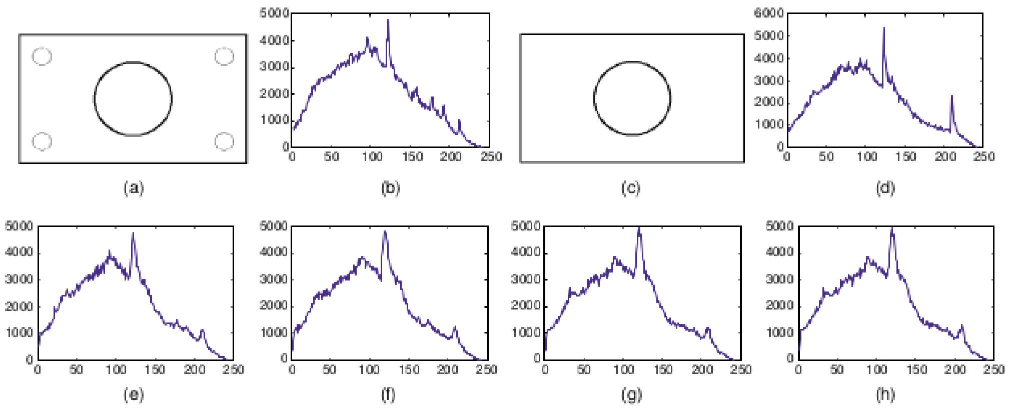


Figure 2.5: Example of descriptors as shape histograms as shown by (Pu & Ramani, 2006) and explained as: A biased sampling example: (a) and (c) are two 2D drawings; (b) and (d) are the shape histograms of the drawings in (a) and (c), respectively; (e)–(h) are the shape histograms with a supersampling of the rectangle and the largest circle, the rate ranging from 200 to 500%, respectively. (Pu & Ramani, 2006)

generally measured using the Minkowski distance L_p with $p = 2$ defined as follows 2.1. It is then basically equivalent to the Euclidean distance. (Trzeciak, 2018)

$$L_p(x, y) = \left[\sum_{i=0}^{N} |x_i - y_i|^p \right]^{1/p} \tag{2.1}$$

Another often required feature of the matching process is partial matching. This is applied when only a part of a could be matching another (Tangelder & Veltkamp, 2004). For instance, when a part of a finger- or palmprint is used to find out to which person it belongs 2.6.

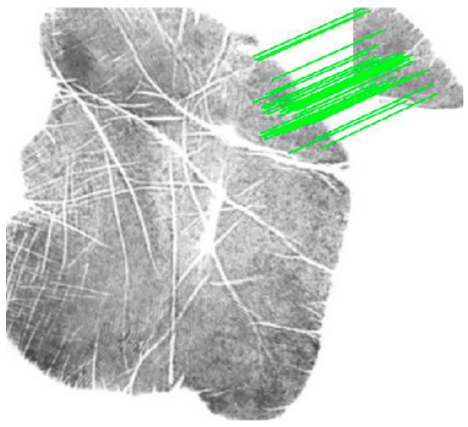


Figure 2.6: Part of a palm matched to a complete palm as shown by (Jain & Feng, 2009)

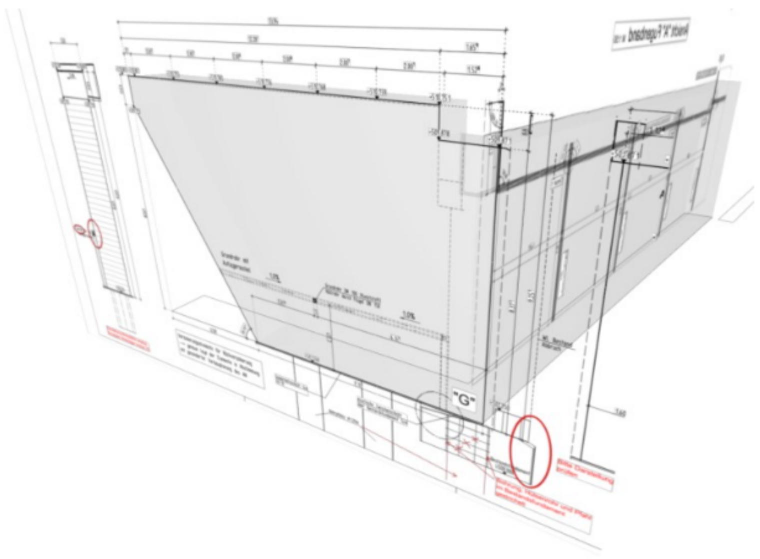


Figure 2.7: Construction drawing aligned to a BIM model as shown by (Trzeciak, 2018)

Transformation

The transformation step is the last step of the registration problem. It consists in actually aligning the two drawings, in a shared coordinate system, to each other as soon as the connection between them is positively established in the matching process (Trzeciak, 2018). Transforming means to translate, rotate or scale the drawings in a shared 3D-coordinate system, in order to have the 2D drawing and the 3D model overlap and positioned in the exact same spot to obtain a visual fit (Trzeciak, 2018) as show in figure 2.7.

2.3 Engineering drawing retrieval

The difficulty about comparing two construction drawings is that these drawings are never compared directly. They are transformed in such a manner that a matching process can be applied, also called shape descriptors (Trzeciak, 2018). The loss of information can be huge and the comparison therefore trivial. According to (Pu & Ramani, 2006), many methods for 2D shape recognition have been proposed but these concentrate on the contour of the object as 2.3.1 shows.

2.3.1 2D contour matching

A simple way of matching to objects is to only match the contour of it. This is widely used, with the use of different approaches (two of them roughly explained as per statement below), as it is regarded to be one of the most important features in Content Based Image Retrieval (Zhang & Lu, 2002).

- Fourier descriptor, which is mainly used for shape analysis but has also seen usage in shape retrieval, is the application of the Fourrier transformation to a shape signature (Zhang & Lu, 2002).
- Curvature scale space is a process where the boundary of an image is transformed to a graph where every concavity or convexity of the original boundary is represented as one contour with one maxima. For every image a set of locations of the maximas is created, and these are compared to each other (Mokhtarian *et al.*, 1997).

These descriptors work for 2D drawing only, but there are also methods to match the contour of a 3D drawing to a 2D one. As (Funkhouser *et al.*, 2003) developed a method where they use spherical harmonics to compare the contours.

2.3.2 2D drawing retrieval

The drawing retrieval problem will be explained in he following. One specific drawing **A** needs to be matched to a library of drawings. For this purpose, the similarity distance (for example: the Euclidean distance) of the drawing **A** to every drawing of the library is computed and, within a certain tolerance, the closest drawing from the library is chosen, as figure 2.8 shows.

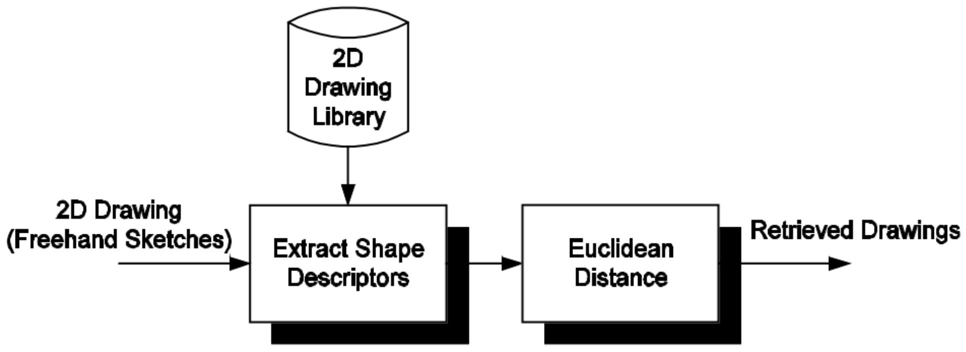


Figure 2.8: 2D Drawing retrieval process as shown by (Pu & Ramani, 2006).

The contour of a building can be a sufficient indicator if two construction drawings are matching, yet when the internal structures are ignored, important information of a construction will be missing (Pu & Ramani, 2006) as 2.9 shows.

(Love & Barton, 2001) have presented a coding system which provides automatic, or semi-automatic, coding of geometric data extracted from a Drawing Interchange File

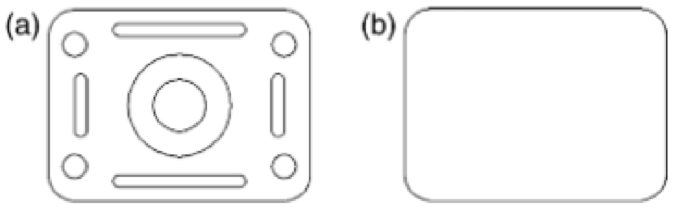


Figure 2.9: The difference between a drawing (a) and its contour (b) as shown by (Pu & Ramani, 2006).

Format (DXF). Although the results seemed promising, they came to the conclusion that better results could be achieved if additional geometric properties would have been taken into account (Love & Barton, 2001).

(Pu & Ramani, 2006) have presented two methods to address the retrieval problem.
One is based on the 2.5D spherical harmonics representation where the key idea is to represent a shape as a spherical function in terms of the amount of energy it contains at different frequencies (Pu & Ramani, 2006).
The other is based on 2D shape histograms, a method where all geometric information of each drawing are translated into a distance histogram.
This paper is based on the second method; it will therefore be specified later on.

Chapter 3

Approach

3.1 Methodology

This thesis is based on the methodology of Jiantao Pu and Karthik Ramani described in their paper "On visual similarity based 2D drawing retrieval" (Pu & Ramani, 2006) and will, therefore, not be explicitly cited at every step of this chapter. While they did present several other options to address the drawing retrieval problem, this thesis focuses on the 2D shape histogram method which will therefore be explained thoroughly.

The method has been tested on ordinary, meaning not construction related, 2D drawings. The compatibility with construction drawings and BIM models is to be determined within this thesis. The objective is to match drawings from two different datatypes, for cross-sections derived from IFC-based BIM models and for DXF. The process elaborated below is the same for both types.

Before being able to apply the methodology, there are a preliminary step required, as the method is only able to compare two 2D drawings. BIM models being 3D, has to be represented by several distinct 2D drawings. To obtain this representation, the 3D model is being cut at different levels, each level being the actual cutting height of a construction drawing on each floor.
The level which achieves the highest overlap between its histogram and the one from the DXF-file, as explained in 3.1.4, is most likely representing the identical floor. This allows not only to determine the likelihood of the buildings matching but also from which floor, or level, the DXF-file has been previously extracted.

3.1.1 Importing a drawing and creating a stroke

For simplification, the process of drawing importation is based on line segments. These are extracted from the DXF file and from the IFC cross-section. This accumulation of segments, represented by their start- and endpoint coordinates, is stored in an array which will be called a **stroke** from this point on and can be represented as 3.1:

$$stroke = [((x_0, y_0), (x_1, y_1)), ((x_i, y_i), (x_{i+1}, y_{i+1})), ...] \tag{3.1}$$

with $i \leq \ n - 1$ and **n** the total number of segments included in the stroke.

3.1.2 Random sampling

For the 2D shape histogram to be created, the segments need to be replaced by a series of points. This process which is called random sampling with uniformly distributed points is a seven step table-based algorithm.

1. The summed length of all line segments included in the drawing is computed by adding each segment length to the summed length one at a time. This value is saved in an array with size **n**, where **n-1** is the total number of line segments. The length table is an array with each entry **i** represented as follows 3.2:

$$length\_table(i) = \sum_{j=0}^{i} L((x_j, y_j), (x_{j+1}, y_{j+1})) \tag{3.2}$$

with $0 \leq \ \ i \leq \ \ n - 1$ and **L** being the length of a segment calculated by the Euclidean distance between the start- and the endpoint of each segment.

2. Create a random number **r** between **0** and the total length of the stroke, which is the last entry of the length table.

3. Find out on which segment the random number **r** is located by finding its two nearest neighbours in the length table. This segment will be called segment **m**, described by its coordinates: $((x_m, y_m), (x_{m+1}, y_{m+1}))$.

4. Create a random number **l**, called **random_l** in the code, between **0** and **1**, which will be used to find a random position on the segment **m**.

5. Find the coordinates of the randomly chosen sample point with the help of the following equation 3.3:

$$\begin{cases} x_k = x_m + random\_l\big(x_{m+1} - x_m\big) \\ y_k = y_m + random\_l\big(y_{m+1} - y_m\big) \end{cases} \tag{3.3}$$

6. Save the coordinates (x_k, y_k) in an array **A**.

7. Repeat the process until the desired number of sampling points is reached.

3.1.3 Creating a histogram

The histogram needed is a distance histogram which represents the dispersion of the sample points in relation to their distance to a common, and for all compared histograms identical, source point, as shown in figure 3.1. The reasoning behind the chosen source point will be explained in the case study.

Building the histogram can be described in 3 steps.

1. The distance of the sample points from a source point will be measured in regular intervals which can be represented as shells. A shell being a circular area within a fixed distance to the previous one, as shown in figure 3.1.
Set this distance.

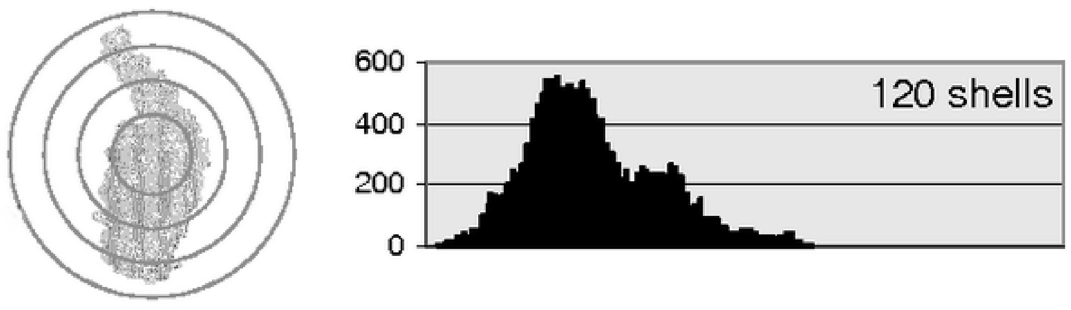


Figure 3.1: Distance histogram build from a shell based query with the source point being the center of the shells as shown from (Ankerst *et al.*, 1999).

2. Gather the number of sample points on each interval, one shell at a time and save this number as the value of a dictionary, and the distance of the outer border of the shell as the key.

3. Plot the dictionary by using the keys as **x**-coordinates and the values as **y**-coordinates.

3.1.4 Histogram comparison

To compare two histograms built with this method, comparing the values of the dictionaries which the histograms are built on is sufficient. The keys, and therefore the **x**-coordinates, are identical, as both histograms are created with the same interval distance set in the step **1.** of the histogram creation, and, therefore, do not need to be included in the comparison step.

For purpose of comparison, there are different possible approaches. Having tested some of these, including the two Minkowski distances **L1** and **L2**, it seems that the **Hellinger distance** gives the most distinct and, therefore, most accurate results for this specific use and impementation. The tests will be specified in the chapter **Evaluation and case study**.

For two discrete probability distributions the **Hellinger distance** is defined (J. Oosterhoff, 2011) as the following equation 3.4 shows:

$$Dist_{Hel} = \frac{1}{\sqrt{2}}\sqrt{\sum_{i=0}^{n}\left(\sqrt{H_1(i)} - \sqrt{H_2(i)}\right)^2} \tag{3.4}$$

with **n** being the size of the array, H_1 being the **y-** coordinates of the first histogram and H_2 the ones of the second.

The smaller the value of the **Hellinger distance** of two histograms the more analogous they are.

Chapter 4

Implementation

4.1 The code

The code is based on as well as linked to the **deep_linkage** (Trzeciak, 2019) project my academic advisor had built prior to this thesis. Packages used from his code will be explained by looking at their purpose and not their specific functioning.
Furthermore, the drawings and models 4.1 used for the comparison process are also taken from the **deep_linkage** project (Trzeciak, 2019).
This section will go through the code step by step, closely examining the methods created and used.

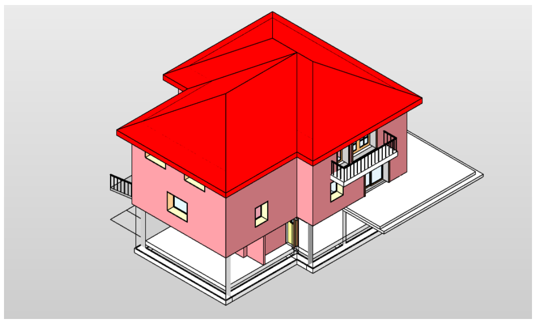
Step 1: number of sampling points

Before going through the process of the DXF-file and later the IFC-file, the number of sampling points is set. The sampling points are randomly chosen points along the drawings' lines. Thereby, the drawings will no longer be represented by segments with respective start- and endpoints but by a cloud of points.

Listing 4.1: Setting number of sampling points

```
# Define number of sampling points
    number_of_sampling_p = 10 ** 4
```

The higher this number, the more accurately the drawing will be represented by the list of points in 2D.


(a) IFC model

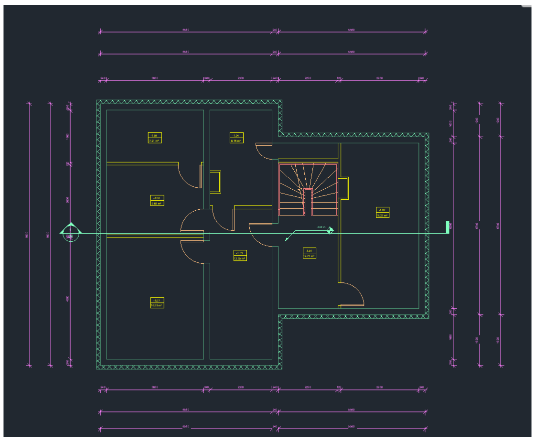

(b) DXF drawing

Figure 4.1: Model and drawing used in this thesis, provided by (Trzeciak, 2018)

4.1.1 DXF-file

Step 2: DXF drawing retrieval

The variable **dxf_extractor** is associated to a method which has been retrieved from **deep_linkage**. Its purpose is to extract all corners from the DXF-file and the segments associated to it.

Listing 4.2: Drawing retrieval

```
# DXF-file

    #  query_drawings
    dxf_extractor = DXFCornerExtractor("../DXF_files/drawing_-13.dxf")
```

Step 3: create a stroke for the DXF-file

A stroke is the accumulation of all segments embodied in the original drawing represented in one array, similar to the mark left by a pen on a piece of paper, hence the name. The name **stroke_dxf** also refers to the association of this stroke to the DXF-file.

Listing 4.3: Create stroke

```
 # numpy array with list of all segments in DXF files normed to [m]
    stroke_dxf = RandSamplTools.norm_drawing(np.array(
                        dxf_extractor.get_segment_wise_bases()))
```

This line of code has a couple of methods encapsulated into each other which will be explained from the inside out.

1. dxf_extractor.get_segment_wise_bases() creates an array with segment wise start- and endpoints listed as tuples one after the other as explained in chapter 3.1.1.

2. np.array() changes the format of this array to a numpy array for later processing. Numpy arrays can have different shapes, which causes problems with the next steps in this code. To address this issue, the **norm_drawing** method also includes a reshaping step in case the shape happens to differ.

3. norm_drawing() is a method, which is part of the class **RandSamplTools**, has been created for this thesis and contains all the methods necessary for the random sampling process.
The drawing issued from the DXF- and the IFC-file are either scaled in millimeters or in meters. This method is used to adapt them to the same scale.

Listing 4.4: norm_drawing method

```
class RandSamplTools:

    def norm_drawing(array):
        # bringing drawing from [m] to [mm] if necessary
        if np.shape(array)[1] and np.shape(array)[2] != 2:
            a = int(np.size(array) / 4)
            stroke = np.reshape(array, (a, 2, 2))
        else:
            stroke = array

        max_val = np.amax(stroke[:][:][0])
        if max_val < 1000:
            stroke = [(x * 1000, y * 1000) for (x, y) in stroke]
        return stroke
```

Once the numpy array is transferred to this method, to ensure the flawless performance of the rest of the code, the shape of the array is checked and if necessary changed to the required format, using the numpy **reshape** method.

To find out if the drawing is in meters or in millimeters, the highest value for the **x-** or **y**-coordinate is searched for and associated to the **max_val** variable. The next step is based on the assumption that this value, if smaller than 1000, must be in meters. This is based on the reflection that any drawing in millimeters will have a much larger number and a drawing in meters will unlikely exceed this limit, simply because of the size of an average building.
If the drawing turns out to be in meters, the next step multiplies every single coordinate elementwise per 1000 to scale it down to millimeters.

Step 4: Random Sampling for DXF

The name array_A is inherited from the paper 'On visual similarity based 2D drawing retrieval' (Pu & Ramani, 2006) which this thesis is based on.
This array is a collection of points which together represent the drawing as a list of points in 2D. After this step the drawing is no longer a series of segment, or 'stroke', which is essential for the next steps.

Listing 4.5: Random sampling for DXF

```
# array with all sample points along the dxf file
    array_A_dxf = RandSamplTools.sample_points(length_table=
                        RandSamplTools.length_table(stroke_dxf),
                        stroke=stroke_dxf,
                        number_of_sampling_p=number_of_sampling_p)
```

This line of code calls on two methods encapsulated into each other. They will be explained from the inside out.

1. length_table() is a method, also part of the **RandSampTools** class, which creates a linear array, here called **length_table**, with size **n**, and **n-1** being the total number of line segments contained in the drawing.
Each entry is the length of the stroke up to and including the segment **n-1** as explained in chapter 3.1.2 .

Listing 4.6: length_table method

```
    def length_table(stroke):
        # summed length
        length_table = []
        for i in range(0, len(stroke)):
            if i == 0:
                length_table.append(np.linalg.norm(stroke[i]
            if i != 0:
                length_table.append(length_table[i - 1] +
                 - stroke[i][1]))
        return length_table
```

The method is given the array **stroke**, created one step beforehand. It then creates an empty array, **length_table**.
For each entry **i** of the array **stroke**, the length of the segment is calculated using the numpy linalg.norm method.
The **length_table** array is then appended by the sum of the previous entry and the length calculated in this step.
When every segment is calculated the whole array is returned.

2. sample_points() is a method, contained in the **RandSamplTools** class. The purpose is to randomly select points on the segments and replace the lines by an accumulation of points. It needs the input of the array **length_table** calculated in the step beforehand as well as the array **stroke** and the number of sampling points, **number_of_sampling_p**, defined in the first step.

Listing 4.7: sample_points method

```
def sample_points(length_table, stroke,number_of_sampling_p):

    # find all sampling points
    n = 0  # counter for sampling points
    array_A = []  # array for sampling_points

    while n < number_of_sampling_p:
        # create random number and find segment on
        r = random.randint(0,
        round(length_table[len(stroke) - 1]))
        # print(r)

        m = bisect.bisect_left(length_table, r, 0, len(length_table))

        # create random number to find random position on segment
        random_l = round(random.uniform(0, 1), 10)

        x_coords, y_coords = zip(*stroke[m])

        x_k = x_coords[0] + random_l * (x_coords[1] - x_coords[0])
```

```
            y_k = y_coords[0] + random_l * (y_coords[1] - y_coords[0])

            array_A.append((x_k, y_k))

            n += 1

        return array_A
```

In a first step, an empty array, **array_A**, for the storage of the sampling points coordinates, and a variable **n**, as a sampling point counter, are set.
While the counter **n** did not reach the number of sampling points set in the beginning, a new point will be created.
A random real number **r**, between **0** and the rounded total length of the stroke (last entry of the array **length_table**), is created.
m is the segment of the stroke on which **r** is located. This is found with the help of a python inbuilt function: **bisect**.
Another random number is created, **random_l**, which is located between **0** and **1**.
This random number, **random_l**, represents a position on said segment.
With the help of the equation 3.3, described in chapter 3.1.2, the coordinates of this point can be found.

The array, **array_A**, is then appended by these, and the counter **n** is increased by 1.
This process is repeated until **n** is equal to **number_of_sampling_p**, and the array, **array_A** is returned.

Step 5: Point distance query for DXF-file

The next step towards the distance histogram is to create a dictionary, here called **length_prop_dict_dxf**, with the keys being the distance to a source point and the values being the numbers of points found from one distance to the next.

Listing 4.8: Point distance dictionary

```
# dictionary with the point distance distribution
    length_prop_dict_dxf = HistUtilities.point_query(array_A_dxf,
    number_of_sampling_p)
```

This line of code calls the methods **point_query()** from the **HistUtilities** class, which has been created for this thesis. It contains all the methods required to create and compare histograms.

Listing 4.9: point_query method

```
def point_query(array_A, number_of_sampling_p):

    np_array_A = np.array(array_A)  # create numpy array
    length_prop_dict = SortedDict()
    n2 = 0  # counter for sampling points
    n3 = 1000  # counter for distance from outer boundary shell to
                source point
    points_in_other_shell = []

    point_tree = spatial.cKDTree(np_array_A)

    while n2 < number_of_sampling_p:
        points_in_total_shell = point_tree.query_ball_point([0, 0], n3)
        points_in_shell = [x for x in points_in_total_shell
                           if x not in points_in_other_shell]
        points_in_other_shell = points_in_total_shell

        length_prop_dict[n3] = len(points_in_shell)

        n2 += len(points_in_shell)
        n3 += 1000

    return length_prop_dict
```

point_query() is a method which needs the input of **array_A** and the **number_of_sampling_p** which have been defined beforehand.

The first steps are preliminary measures to ensure the proper functioning of this method.
The **array_A** is transformed into a numpy array. An empty sorted dictionary, **length_prop_dict**, is created with the help of the python inbuilt method **SortedDict()**.
As well as 2 counters, **n2** and **n3**, which are set to **0**. **n2** being the counter for the sampling points and **n3** being the counter for the distance from the source point in millimeters.
points_in_other_shell is an empty array which will be filled with the points already counted.

The variable point_tree is associated with the class **scipy.spatial.cKDTree**, which gets the **array_A** as input argument. This class provides an index into a set of k-dimensional points which can be used to rapidly look up the nearest neighbours of any point (SciPy-Community, 2019).

The next few steps will be repeated until the **n2** counter is equal to the number of sampling points.

The **query_ball_point()** method from the **scipy.spatial.cKDTree** class is used to gather all the points which are within the distance **n3** of the source point, here defined as [**0, 0**]. The result is saved under the variable **points_in_total_shell**.
To find out how many points are present in this shell, the points already found in the previous steps, **points_in_other_shells**, are subtracted from the newly found total points, **points_in_total_shell**, and saved to the variable **points_in_shell**.
After this step, the variable **points_in_other_shells** is updated to include all the points found up to now.
A new pair of **key:value** is added to the dictionary, the **key** being the outer boundary of this shell (= the value of **n3**) and the **value** being the number of points on this shell(= the value of **points_in_shell**).

Finally, the counters **n2** and **n3** are increased. **n2** by the number of points found in this shell. **n3** is increased by 1000 millimeters, which after testing various other increment sizes has shown to provide the best results.

Step 6: Plots for DXF-file

In this step, the drawing, represented as a list of points in 2D, and the histogram, based on the dictionary, are plotted. They are represented in two different figures.

Listing 4.10: Plots

```
# plots for dxf file
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111)
    ax1.scatter(*zip(*array_A_dxf))

    fig2 = plt.figure()
    ax2 = fig2.add_subplot(111)
    ax2.plot(list(length_prop_dict_dxf.keys()),
                length_prop_dict_dxf.values(),
                color='g')
    plt.show()
```

The first figure, **fig1**, shows the construction drawing as a list of points in 2D. The coordinates of the points are given to the **scatter** method (from the **matplotlib** librabry) to plot these. The input needed are two separate lists of **x**- and **y**-coordinates. These are extracted from the **array_A_dxf**, using ***zip** a python standard library tool, which separates the coordinate tuples in two distinct lists.
The second figure, **fig2**, shows the corresponding distance histogram. It is built by using the **plot** method from the **matplotlib** library. The **x**-coordinates are given by the keys of the dictionary, **list(length_prop_dict_dxf.keys())**. The **y**-coordinates are given by the values, **length_prop_dict_dxf.values()**.

4.1.2 IFC-file

Step 7: IFC cross-section retrieval

The variable **ifc_file** is associated to the file of the 3D-model, for later processing.

Listing 4.11: IFC_file

```
# query drawing
ifc_file = "../IFC_models/BIMmodel.ifc"
```

The 3D-model will be cut at four different levels, these, and two counters that will be explained later on, are set before entering the for-loop.

Listing 4.12: Setting cutting levels and counters

```
levels = [-1.3, 1.2, 3, 4]
biggest_overlap = -1
closest_level = 0
```

Step 8: Start for-loop over each level

A for-loop over every level, on which the 3D-model will be cut, is started. The steps **9** to **13** will be repeated until they have been completed for every step of the loop.

Listing 4.13: For-loop

```
for level in levels:
```

Step 9: Create a stroke for IFC-file

Just as for the DXF-file, the goal is to create an array containing all the segments in the drawing. The first try to accomplish this task was with the **ifc_extractor_small.get_element_wise_bases()** from the **deep_linkage** prototype, which did not produce the required results as some segments where missing, or could not be processed by the code. This approach is still integrated to the code but has been commented out, signalled by the **pound** symbol.
A solution working for the code has been provided to me by a research assistant working for the same chair as this thesis is supervised from. It has been integrated in the code as the **IFCExtracting** class and will not further be discussed as it is not my work.

Listing 4.14: Stroke for IFC-file

```
# numpy array with list of all segments in ifc files normed to [mm]
# stroke_ifc = RandSamplTools.norm_drawing(np.array(
                    ifc_extractor_small.get_element_wise_bases()))
stroke_ifc = RandSamplTools.norm_drawing(np.array(
                    IFCExtracting.create_lines(IFCExtracting,
                    ifc_file, level)))
```

This last line of code, which creates the stroke for the IFC cross-section, has a couple of methods encapsulated into each other. They will be explained from the inside out.

1. IFCExtracting.create_lines() creates an array with segment-wise start and end points listed one after the other. The input needed next to itself, is the **ifc_file** and the current **level** we are operating on.

2. np.array() changed the format of this array to a numpy array for later processing. Numpy array can have different shapes, which causes problems with the next steps in this code. This is why the **norm_drawing** method also includes a reshaping step if the shape differs.

3. norm_drawing() is a method which is part of the class **RandSamplTools**, which contains all the methods necessary for the random sampling process and has been created for this thesis.

The drawing issued from the DXF- and the IFC-file are either scaled in millimeters or in meters. This method is used to bring them to the same scale.

Step 10: random sampling for IFC-file

This line of code functions analogous to **step 4**. Therefore, the explanation can be looked up there.

Listing 4.15: random sampling for IFC

```
# array with all sample points along the ifc file
        array_A_ifc = RandSamplTools.sample_points(
                        length_table=RandSamplTools.length_table(stroke_ifc),
                        stroke=stroke_ifc,
                        number_of_sampling_p=number_of_sampling_p)
```

Step 11: Point distance query for IFC-file

This line of code functions in analogous to **step 5**. Therefore, the explanation can be looked up there.

Listing 4.16: Point distance query for IFC

```
# dictionary with the point distance distribution
        length_prop_dict_ifc = HistUtilities.point_query(array_A_ifc,
                                        number_of_sampling_p)
```

Step 12: Plots for IFC-file

These lines of code function in analogous to **step 6**. Therefore, the explanation can be looked up there.

Listing 4.17: Plots for IFC

```
# plots for ifc file
        # plot drawing
        fig1 = plt.figure()
        ax1 = fig1.add_subplot(111)
        ax1.scatter(*zip(*array_A_ifc))

        # plot histogram
        fig2 = plt.figure()
        ax2 = fig2.add_subplot(111)
        ax2.plot(list(length_prop_dict_ifc.keys()),
                        length_prop_dict_ifc.values(),
                        color='g')
        plt.show()
```

Step 13: Histogram comparison

For the histogram comparison, four different methods have been tested, including the one chosen in the paper this thesis is based on.
The method which seemed to bring the most accurate results, for the way this program is implemented, is the **Hellinger distance**.
Hence the reduced explanation of the code to this methodology only.

To find out which level of the IFC-file is the closest to the DXF-file, the two step process is repeated for each level.

Listing 4.18: Histogram matching

```
hel_dist = HistUtilities.hist_comp_hel(length_prop_dict_dxf,
                            length_prop_dict_ifc, level)
closest_level, biggest_overlap = HistUtilities.max_similarity(hel_dist,
                            biggest_overlap, level, closest_level)
```

The histograms are compared by calling on two methods from the **HistUtilities** class. These two lines will be explained in the the next two sub points.

1. The Hellinger distance, **hel_dist**, for the level is determined by calling on the method **hist_comp_hel**.

The following code is based on equation 3.4 as shown in section **3.1.4** and is the method previously called on.

Listing 4.19: Hellinger distance

```
    def hist_comp_hel(h1_dxf, h2_ifc, level):

        hel_prelim = 0
        array_1 = list(h1_dxf.values())
        array_2 = list(h2_ifc.values())

        while len(array_1) < len(array_2):
            array_1.append(0)
        while len(array_2) < len(array_1):
            array_2.append(0)

        if len(array_1) == len(array_2):
            for i in range(0, len(array_1)):
                hel_prelim += (((array_1[i]) ** 0.5) -
                                       ((array_2[i]) ** 0.5)) ** 2

            hel_final = (1/(2 ** 0.5)) * ((hel_prelim) ** 0.5)

        return hel_final
```

The first step is to create an environment for the equation to work properly. From the two compared histograms, two arrays with the number of points per shell are extracted. This corresponds to the values of the point-distance dictionary mentioned in **Step 5 and 11**. It is then tested if the two arrays are similar in size. If they are not, the smaller array is appended by **zeros** until the size is the same.

The distance is then calculated in two steps.

1. First the sum of the equation is established and saved to a preliminary variable, **hel_prelim**. In a second step, the actual **Hellinger distance** is calculated by taking the square root of the preliminary value and multiplying it by $\frac{1}{\sqrt{2}}$.

2. After the Hellinger distance is calculated, the accuracy of the histogram overlap needs to be checked. The method **max_similarity** is called on and gives back two variables, the level on which, until now, the overlap is the greatest, called **closest_level**, and by how much they overlap, called, **biggest_overlap**.

The distance is compared to the previous one. If the fit is more accurate, this level is determined as the **closest_level**. Elsewise, the previous one remains.

Listing 4.20: Finding the matching level

```
def max_similarity(hist_similarity , biggest_overlap, level,
                    closest_level):

    if hist_similarity < biggest_overlap or biggest_overlap == -1:
        closest_level = level
        biggest_overlap = hist_similarity

    return closest_level, biggest_overlap
```

To be able to compare the first level properly, two variables, **closest_level** and **biggest_overlap**, have already been set in **Step 7**.

max_similarity is a method which takes three input arguments. **hist_similarity** is the **Hellinger distance** from this level, **biggest_overlap** is the **Hellinger distance** from the, up to this point, best fitting level, and **closest_level** is the level on which it is located.

In this method, the distance is compared to the previous one. If the fit is more accurate, this level is determined as the **closest_level** and the variable **biggest_overlap** is updated. Elsewise, the previous one remains.

Step 14: Result

After the steps **8** to **13** have been repeated for every level of the IFC-file, the result, meaning on which level the minimum distance between histograms is achieved, is printed.

Chapter 5

Evaluation and case study

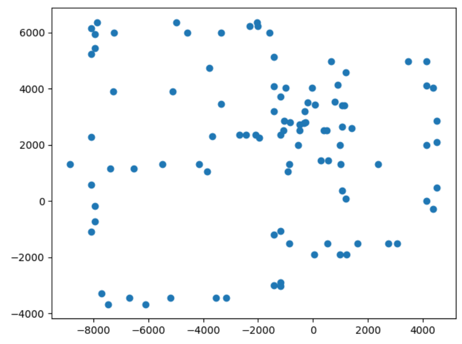
Four factors are determining the precision of the histogram build for each drawing, the number of sampling points, the size of the shells on which the number of points are counted on, the source point location from which the distance of the shells is measured and the accuracy of the method used to measure the distance between the histograms.

As (Pu & Ramani, 2006) showed in their paper, 10^5 points seemed to give the best results. While trying to match their number of sample points on this code, the runtime exceeded any reasonable measure with over 5 minutes for one drawing only.
To determine which values are giving the best results, a series of tests have been made.

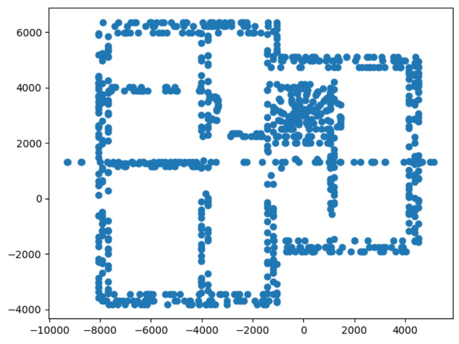
1. Analysing which number of sample points is giving a precise portrayal of the construction drawing as shown in the figure 5.1.
The 5.1d serves as a reference as it is the extracted drawing represented as line segments before the random sampling process is applied. The closer the plotted points get to this representation the more precise they are, which seems to be more than enough in 5.1c.

A direct correlation is to be found between the number of points and the precision of the representation. Consequently, an increased point count results in a more accurate representation as well as corresponding rise in runtime.

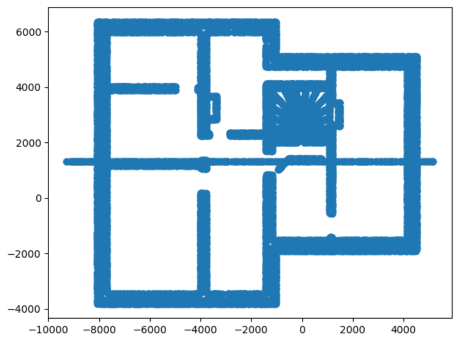
2. When the number of sampling points is lower than 10^4, the distribution of the points is not similar enough due to the lack of point density. The histograms of a same drawing will differ due to this as figures 5.2, 5.3, 5.4 show.



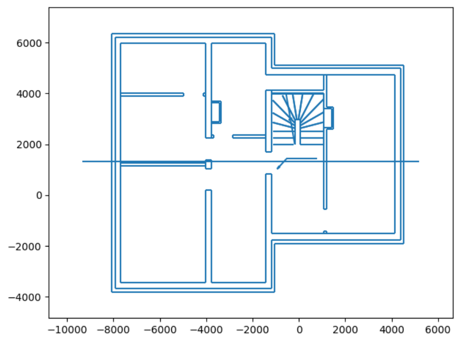
(a) 10^2 sampling points



(b) 10^3 sampling points

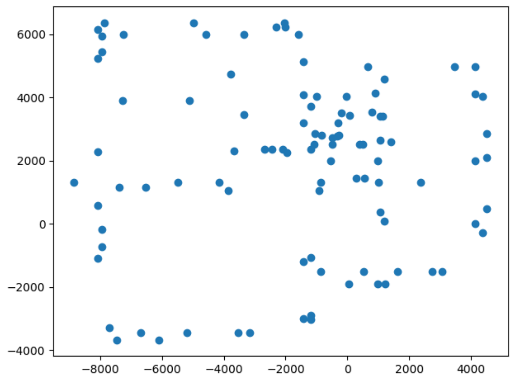


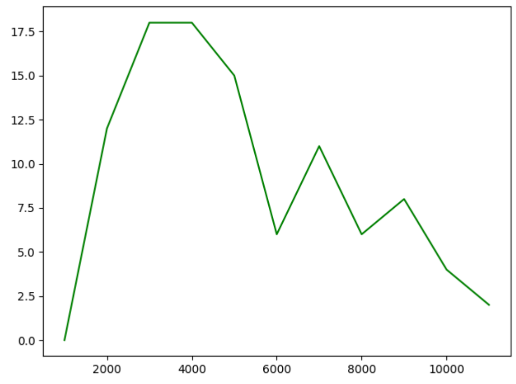
(c) 10^4 sampling points

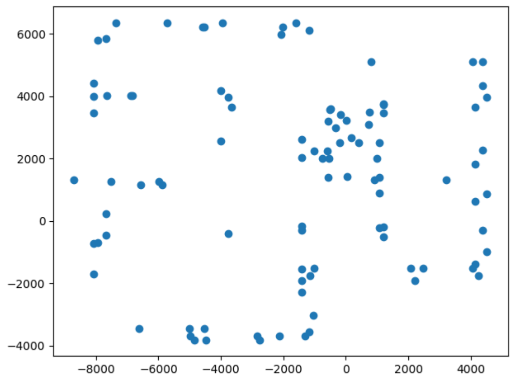


(d) lines

Figure 5.1: Plots of the DXF-file with different numbers of sampling points and the plot of the drawing before the random sampling process.


(a) First iteration


(b) According histogram to plot of first iteration


(c) second iteration

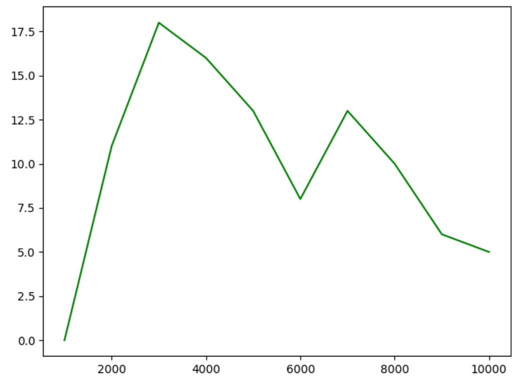
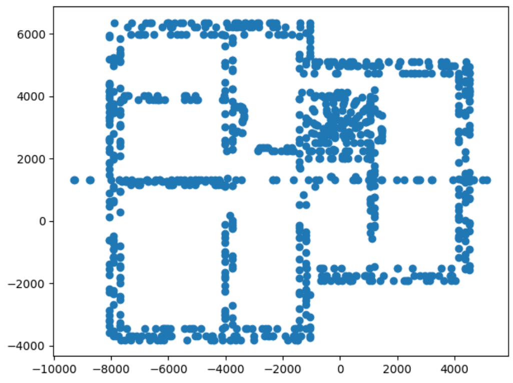
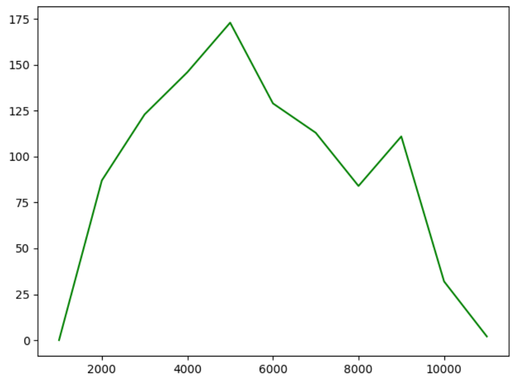

(d) According histogram to plot of second iteration

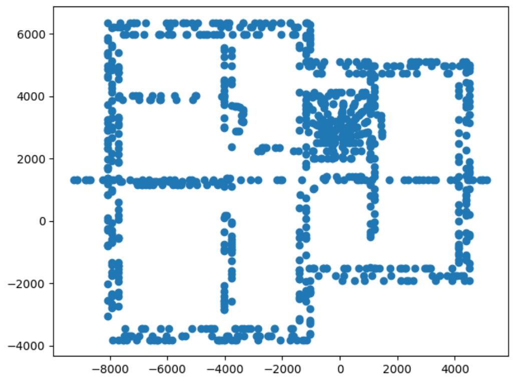
Figure 5.2: Plots of the DXF-file with 10^2 sampling points and according histograms after first and second iteration.



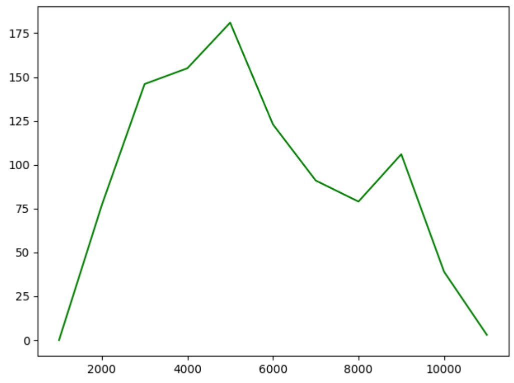
(a) First iteration



(b) According histogram to plot of first iteration

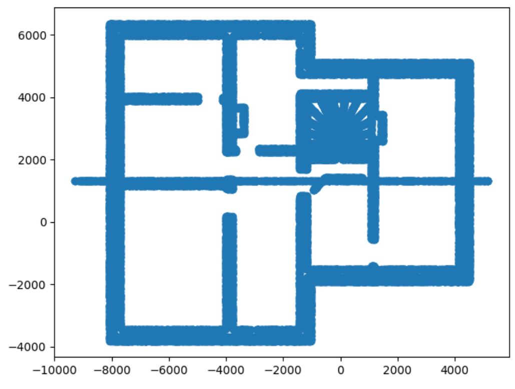


(c) Second iteration

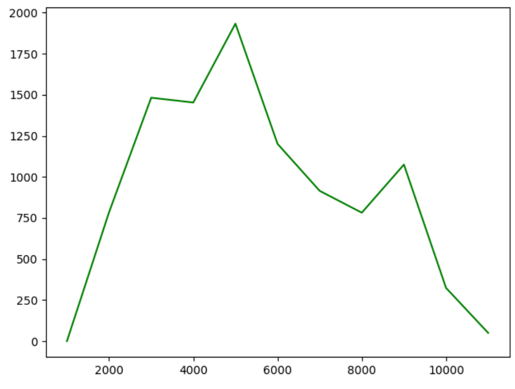


(d) According histogram to plot of second iteration

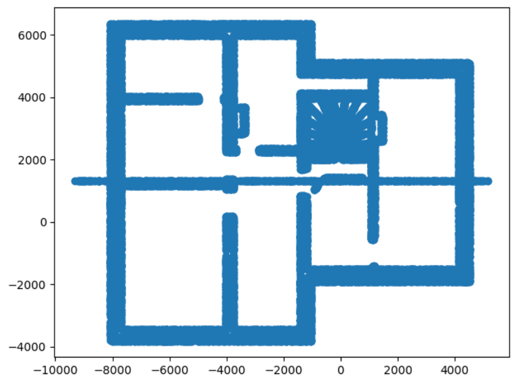
Figure 5.3: Plots of the DXF-file with 10^3 sampling points and according histograms after first and second iteration.



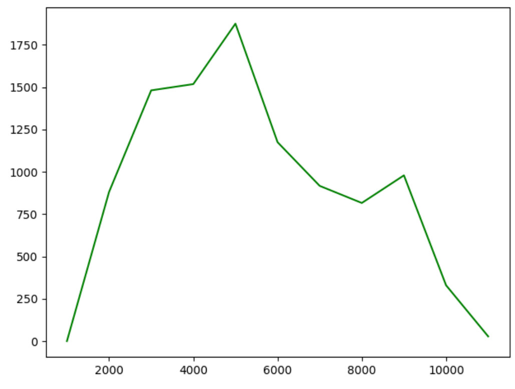
(a) First iteration



(b) According histogram to plot of first iteration



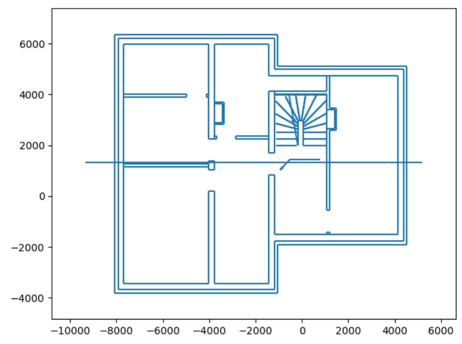
(c) Second iteration



(d) According histogram to plot of second iteration

Figure 5.4: Plots of the DXF-file with 10^4 sampling points and according histograms after first and second iteration.

3. Although the drawings are all issued from the same 3D model, extracting the drawing as a DXF-file beforehand and directly cutting the IFC based BIM model does not result in an identical representation of the drawing as figure 5.5 demonstrates.


(a) DXF drawing

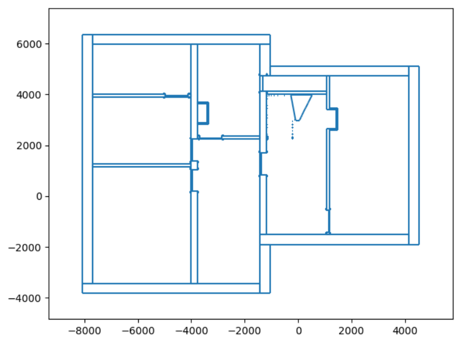
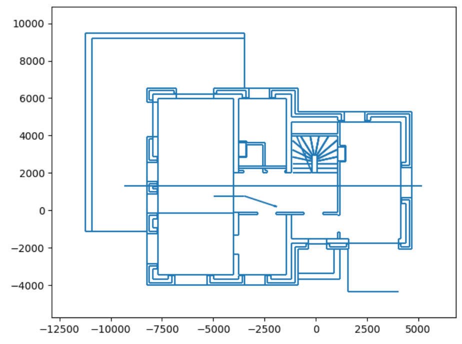

(b) IFC drawing

Figure 5.5: Plots of DXF drawing and IFC cross-section on the same level (-1.3)


(a) DXF drawing

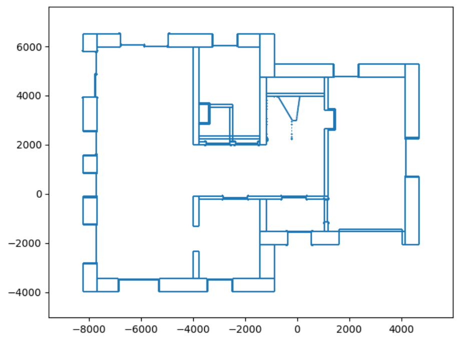
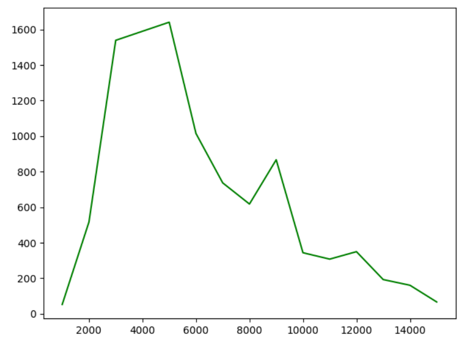

(b) IFC drawing

Figure 5.6: Plots of DXF drawing and IFC cross-section on the same level (1.2)


(a) Histogram based on DXF drawing

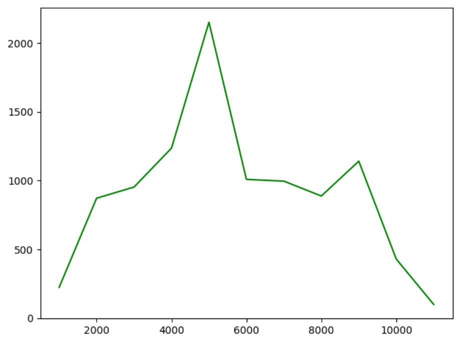
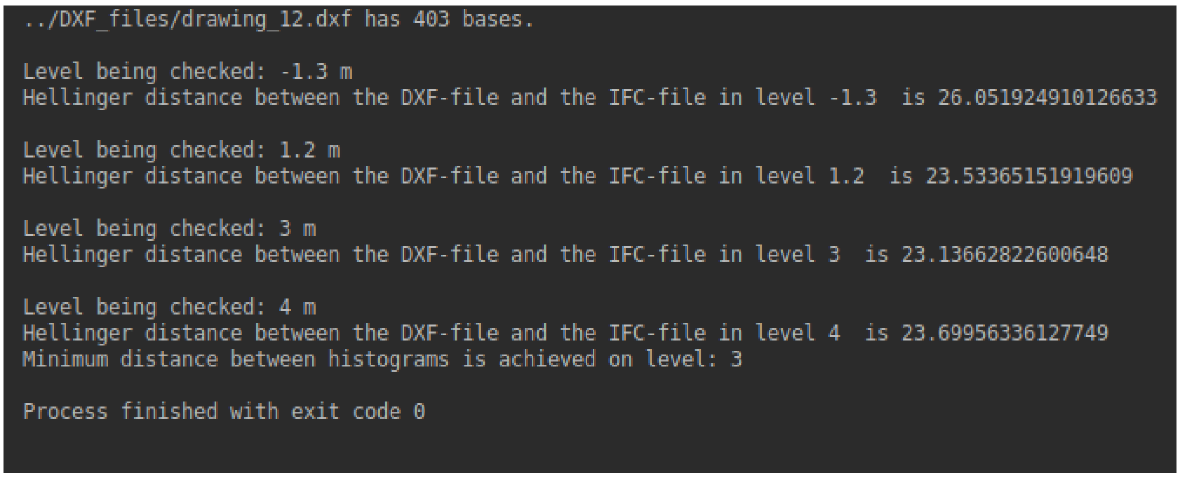

(b) Histogram based on IFC drawing

Figure 5.7: Histograms based on the DXF drawing and the IFC cross-section on level 1.2, plots of the drawings are in figure 5.6

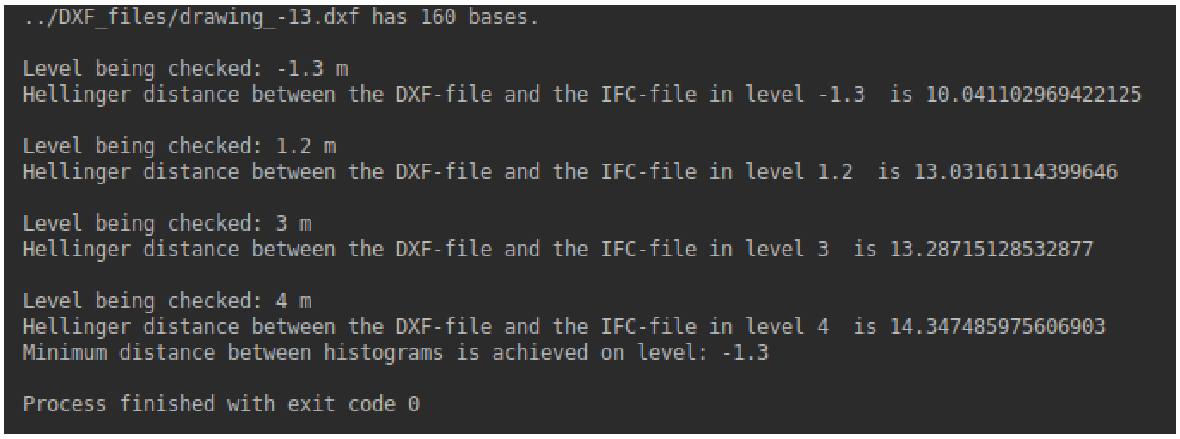
Furthermore, the descriptor does not seem to support partial matching. Meaning that it is not able to match parts of an image to another image. It will always compare the two whole images which creates problems when trying to match an IFC cross-section to a DXF drawing.
The cross-section takes only the parts of the building which exactly intersect with the cutting level into account. A staircase or terrace will not be displayed, partially for the staircase and completely for the terrace as figure 5.6 shows.
While extracting a DXF-file these attributes are completely forwarded. This creates a clear difference between the two, originally identical, drawings.
Therefore, the histograms issued from these drawings will not match perfectly even if the rest of the process works correctly as shown by figure 5.7.
The chances of a wrong assignment of the corresponding level increases because of this as 5.8 shows.



```
../DXF_files/drawing_12.dxf has 403 bases.

Level being checked: -1.3 m
Hellinger distance between the DXF-file and the IFC-file in level -1.3  is 26.051924910126633

Level being checked: 1.2 m
Hellinger distance between the DXF-file and the IFC-file in level 1.2  is 23.53365151919609

Level being checked: 3 m
Hellinger distance between the DXF-file and the IFC-file in level 3  is 23.13662822600648

Level being checked: 4 m
Hellinger distance between the DXF-file and the IFC-file in level 4  is 23.69956336127749
Minimum distance between histograms is achieved on level: 3

Process finished with exit code 0
```

Figure 5.8: Progress report on level 1.2 with terrace, the wrong level is selected.

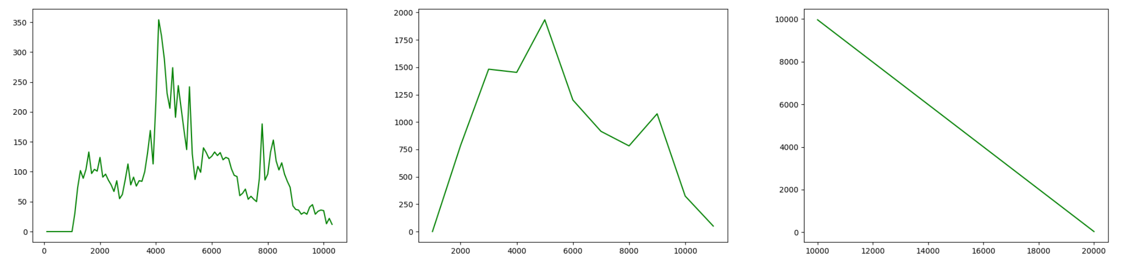
When the differences are not as dramatic as a missing terrace, for example a staircase which is not completely shown, the program still is able to find the proper drawing as figure 5.9 shows.



```
../DXF_files/drawing_-13.dxf has 160 bases.

Level being checked: -1.3 m
Hellinger distance between the DXF-file and the IFC-file in level -1.3  is 10.041102969422125

Level being checked: 1.2 m
Hellinger distance between the DXF-file and the IFC-file in level 1.2  is 13.03161114399646

Level being checked: 3 m
Hellinger distance between the DXF-file and the IFC-file in level 3  is 13.28715128532877

Level being checked: 4 m
Hellinger distance between the DXF-file and the IFC-file in level 4  is 14.347485975606903
Minimum distance between histograms is achieved on level: -1.3

Process finished with exit code 0
```

Figure 5.9: Progress report on level -1.3 without a terrace, the correct level is selected.

4. The second variable which influences the histogram output is the size of the shells. All previous examples have been made with a shell size of 1000 millimeters.
The histogram gets more detailed when the shell size is small and less detailed when it is big as shown in figure 5.10. When the histogram is built with too large steps, the histogram gets unusable as 5.10c demonstrates.
The runtime augments the degree of detail as show by figure 5.11.

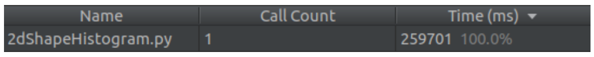


(a) Shell size 100 millimeters **(b)** Shell size 1000 millimeters **(c)** Shell size 10000 millimeters

Figure 5.10: Histograms based on the DXF drawing on level -1.3 built with different shell sizes.

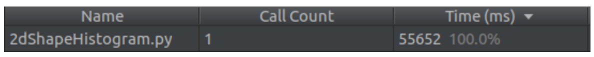
The runtime augments accordingly to the degree of detail as show by figure 5.11 as it is multiplied by a factor **5** when the shell steps are narrowed down from 1000 to 100 millimeters.

Furthermore, the stability of a histogram as a reliable comparison tool is weakened by a too small shell size. The points being randomly chosen, their distribution is likewise, resulting in an ever changing histogram with each time the program is run an additional time, as figure 5.12 displays.



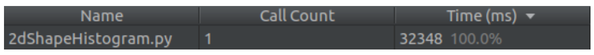
Name	Call Count	Time (ms) ▾
2dShapeHistogram.py	1	259701 100.0%

(a) Shell size 100 millimeters



Name	Call Count	Time (ms) ▾
2dShapeHistogram.py	1	55652 100.0%

(b) Shell size 1000 millimeters



Name	Call Count	Time (ms) ▾
2dShapeHistogram.py	1	32348 100.0%

(c) Shell size 10000 millimeters

Figure 5.11: Runtime of histograms based on the DXF drawing on level -1.3 built with different shell sizes.

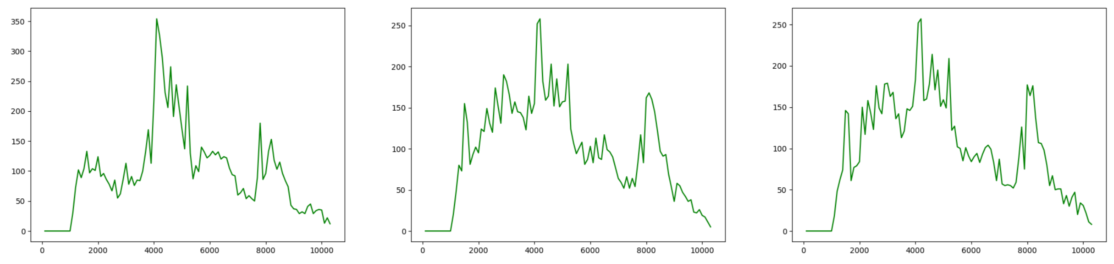


Figure 5.12: Histograms based on the DXF drawing on level -1.3 built with shell size 100 millimeter.

5. The third parameter to influence the result is the location of the source point, from which the distances are measured. $[0, 0]$, as it seemed to be the most obvious one, was compared to $[1000, 1000]$, $[5000, 5000]$, $[10000, 10000]$ and proved to be the right choice as histograms built with the same number of sampling points and shell sizes but with one of the other source point proved to give less precise, or sometimes even wrong, results as the difference between the levels was not as clear.

The results of the test can be found in the appendix.

6. The fourth parameter to affect the results is the method used to measure the distance between the histograms. (Pu & Ramani, 2006) proposed the **Minkowski** distance due to its simplicity, but neither **L1**, nor **L2**, seemed to give results which would not necessairly be correct with each time the program is running through. The **Hellinger** distance however seemed to fit to the way this program was implemented, as it can be seen in the test results shown in the appendix.

5.1 Results

To find out which configuration of these three variables is providing the most accurate results, a test has been performed which results are displayed in the appendix.
The test consist of running three for loops encapsulated into each other. Each loop going through an array filled with sensible input for each variable (sample points, shell size, source point location).

Listing 5.1: For-loop run over the whole program for testing purposes

```
number_of_sampling_p = [1000, 2500, 5000, 7500, 10000]
shell_size = [500, 750, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000,
              9000, 10000]
source_point = [[0, 0], [1000, 1000], [5000, 5000], [10000, 10000]]

for a in number_of_sampling_p:
        for b in shell_size:
                for c in source_point:
```

The result was: **Best results are achieved on [10000, 1000, [0, 0]]**

Meaning that the most efficient setup, for this implementation, for the 2D shape histogram of a construction drawing to be built is to have 10^4 sample points with a shell distance of 1000 millimeters to select their dispersion and $[0, 0]$ as a source point.

The whole testing results can be found in the attached digital data of this thesis.

Chapter 6

Conclusion

In this thesis, the second step of the 3D-to-2D registration problem (similarity measurement and matching) has been addressed in the context of construction drawings and BIM.
The method tested was developed by (Pu & Ramani, 2006) for the matching of sketches to 2D drawings and is called **2D shape histograms**. In this case, it has been implemented to perform the matching process between a 3D BIM model and a 2D construction drawing derived from the same model.

To simplify the process, the comparison in this thesis has been performed exclusively from 2D to 2D. The 3D model was cut into cross-sections at the level were construction drawings usually are derived from.
The difference between a derived construction drawing and an IFC cross-section is that with the deriving process the floor slab is taken into account and transmitted as well as the other interior features of the specific floor. With a cross-section the only parts that are transmitted from 3D to 2D are the parts of the building which are in direct contact with the cross-section, the floor slab is disregarded.
This is not a problem for any facility built with the outer walls being flush with the floor slab as the contour does not change. Yet a building which has a balcony or an terrace planned, the contour is significantly altered when this information is disregarded.

This is where the descriptor encountered severe problems with some of the drawings as, (Pu & Ramani, 2006) had already determined this in their paper, the 2D shape histogram method is good at differentiating 2D drawings with similar contours and different interior structure (Pu & Ramani, 2006), but as it does not support partial matching, as soon as the contour changes too much the method becomes unreliable.

To solve this issue different approaches can be thought of.
First of all, a combination, with a second method, such as the 2.5D spherical harmonics (Pu & Ramani, 2006) is better at differentiating drawings with obvious structure shape distinctions, could be implemented.
A second possible approach would be to combine two cross-sections extracted from the BIM model, one at floor level and the other at the height on which DXF drawings are usually extracted, to create a drawing closer to the derived construction drawing. This could help bypass the issue of the current method not being able to perform partial matching.

In conclusion, the method is only working adequately for drawings with similar contours, which can be an issue when generalizing the process and needs further addressing. The random sampling process works flawlessly, but the point query should be refined as the runtimes reached in the paper of (Pu & Ramani, 2006) were far superior to the ones obtained in this thesis. The point query seems to be the critical phase for optimization of the runtime itself.

Appendix A

Appendix

A.1 Attached Data

The attached data contains:

- The written part of this thesis as PDF document
- The full python code of the 2D shape histogram
- The full testing results as a PDF document

A.2 Details of the testing results

A.2.1 Test for source point

Test is run with 10000 sampling points with a shellsize of 1000 and over different source points:

```
number of sampling points is: 10000
shell size is: 1000
source_point is : [0, 0]

Level being checked: −1.3 m
Hellinger distance between the DXF−file and the IFC−file in level −1.3
is 9.760922311164874
Level being checked: 1.2 m
Hellinger distance between the DXF−file and the IFC−file in level 1.2
is 13.39337025526842
Level being checked: 3 m
```

```
Hellinger distance between the DXF−file and the IFC−file in level 3
is 12.854796692627927
Level being checked: 4 m
Hellinger distance between the DXF−file and the IFC−file in level 4
is 13.773125716522468
Minimum distance between histograms is achieved on level: −1.3
3.5795085769747312


number of sampling points is: 10000
shell size is: 1000
source_point is : [1000, 1000]

Level being checked: −1.3 m
Hellinger distance between the DXF−file and the IFC−file in level −1.3
is 11.317321295669085
Level being checked: 1.2 m
Hellinger distance between the DXF−file and the IFC−file in level 1.2
is 10.750396562024795
Level being checked: 3 m
Hellinger distance between the DXF−file and the IFC−file in level 3
is 11.951533135100533
Level being checked: 4 m
Hellinger distance between the DXF−file and the IFC−file in level 4
is 13.697869568096973
Minimum distance between histograms is achieved on level: 1.2


number of sampling points is: 10000
shell size is: 1000
source_point is : [5000, 5000]

Level being checked: −1.3 m
Hellinger distance between the DXF−file and the IFC−file in level −1.3
is 12.710445724281039
Level being checked: 1.2 m
Hellinger distance between the DXF−file and the IFC−file in level 1.2
is 13.188517999354175
Level being checked: 3 m
Hellinger distance between the DXF−file and the IFC−file in level 3
is 13.72358162380639
Level being checked: 4 m
Hellinger distance between the DXF−file and the IFC−file in level 4
is 13.173924789913817
Minimum distance between histograms is achieved on level: −1.3
```

```
1.8447319390686665

number of sampling points is: 10000
shell size is: 1000
source_point is : [10000, 10000]

Level being checked: −1.3 m
Hellinger distance between the DXF−file and the IFC−file in level −1.3
is 10.938707010266395
Level being checked: 1.2 m
Hellinger distance between the DXF−file and the IFC−file in level 1.2
is 13.478103833294869
Level being checked: 3 m
Hellinger distance between the DXF−file and the IFC−file in level 3
is 11.084028904956071
Level being checked: 4 m
Hellinger distance between the DXF−file and the IFC−file in level 4
is 11.714439402567102
Minimum distance between histograms is achieved on level: −1.3
1.7683943496958412

Best results are achieved on [10000, 1000, [0, 0]] with 3.5795085769747312
```

A.2.2 Testing results for the method chosen

Test is run with 10000 smapling points und a shellsize of 1000 over the source points:

```
Level being checked: −1.3 m
L1 distance between DFX−file and IFC−file in level −1.3  is 2362
L2 distance between DFX−file and IFC−file in level −1.3  is 567926.0
Hellinger distance between the DXF−file and the IFC−file in level −1.3
is 10.10026917265048

Level being checked: 1.2 m
L1 distance between DFX−file and IFC−file in level 1.2  is 1658
L2 distance between DFX−file and IFC−file in level 1.2  is 245152.0
Hellinger distance between the DXF−file and the IFC−file in level 1.2
is 12.549325159483635

Level being checked: 3 m
L1 distance between DFX−file and IFC−file in level 3  is 1678
L2 distance between DFX−file and IFC−file in level 3  is 219444.0
Hellinger distance between the DXF−file and the IFC−file in level 3
is 12.98061210197332
```

```
Level being checked: 4 m
L1 distance between DFX-file and IFC-file in level 4  is 1944
L2 distance between DFX-file and IFC-file in level 4  is 323237.0
Hellinger distance between the DXF-file and the IFC-file in level 4
is 13.04797870673757
```

Bibliography

Ankerst, M., Kastenmüller, G., Kriegel, H.-P. & Seidl, T. (1999). 3D Shape Histograms for Similarity Search and Classification in Spatial Databases. In: *SSD*.

Autodesk, I. (2019). Was ist Software für 3D-Rendering? [Online; accessed 2 April , 2019].

Avbelj, J., Iwaszczuk, D., Müller, R., Reinartz, P. & Stilla, U. (2014). Coregistration refinement of hyperspectral images and DSM : An object-based approach using spectral information.

Azhar, S., Nadeem, A., Mok, J. Y. & Leung, B. H. (2008). Building Information Modeling (BIM): A new paradigm for visual interactive modeling and simulation for construction projects. In: *Proc., First International Conference on Construction in Developing Countries*, Volume 1, S. 435–446.

BIM4INFRA2020, A. (2017). Umsetzung des Stufenplans "digitales planen und bauen". [Online; accessed 23 March , 2019].

Borrmann, A., König, M., Koch, C. & Beetz, J. (2018, 09). *Building Information Modeling: Why? What? How?: Technology Foundations and Industry Practice*, S. 1–24.

Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D. & Jacobs, D. (2003). A Search Engine for 3D Models. *ACM Transactions on Graphics* 22(1), S. 83–105.

Hardin, B. & McCool, D. (2015). *BIM and construction management: proven tools, methods, and workflows.* John Wiley & Sons.

J. Oosterhoff, W. R. v. Z. (2011). A Note on Contiguity and Hellinger Distance. In: *Selected Works of Willem van Zwet*, S. 63 – 72. Springer, New York, NY.

Jain, A. K. & Feng, J. (2009). Latent Palmprint Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, S. 1032–1047.

Jung, J., Sohn, G., Bang, K., Wichmann, A., Armenakis, C. & Kada, M. (2016). Matching aerial images to 3D building models using context-based geometric hashing. *Sensors* 16(6), S. 932.

Kim, H., Anderson, K., Lee, S. & Hildreth, J. (2013). Generating construction schedules through automatic data extraction using open BIM (building information modeling) technology. *Automation in Construction* 35, S. 285–295.

Lenihan, R. (2015). Navisworks Clash Detection 101 – Part 1.

Love, D. & Barton, J. (2001). Drawing retrieval using an automated coding. In: *Proceedings of the 11th international conference on flexible automation and intelligent manufacturing*, S. 158–166.

Mokhtarian, F., Abbasi, S. & Kittler, J. (1997). Efficient and robust retrieval by shape content through curvature scale space. In: *Image Databases and Multi-Media Search*, S. 51–58. World Scientific.

Pu, J. & Ramani, K. (2006). On visual similarity based 2D drawing retrieval. *Computer-Aided Design* 38(3), S. 249–259.

Schinler, D. & Nelson, E. (2008). BIM and the structural engineering community. *STRUCTURE* 10.

SciPy-Community, T. (2019). Spatial algorithms and data structures (scipy.spatial.cKDTree). [Online; accessed 5 March , 2019].

Tangelder, H. & Veltkamp, R. (2004, 01). A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications 39* 39, S. 441–471.

Trzeciak, M. (2018). Towards Registration of Construction Drawings to Building Information Models. In: *Proc. of the 30th Forum Bauinformatik*, Weimar, Germany.

Trzeciak, M. (2019). deep_linkage.

Zhang, D. & Lu, G. (2002). Shape-based image retrieval using generic Fourier descriptor. *Signal Processing: Image Communication* 17(10), S. 825–848.

Declaration of Originality

With this statement I declare, that I have independently completed this Bachelor's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

München, 08. April 2019

Clément Thiele

[redacted]