UNIVERSITY OF LEEDS

Faculty of Engineering
School of Civil Engineering

Postgraduate Research Thesis:

Automating Visual Inspection, Documentation, and Assessment of Masonry Structures Using Computer Vision and Deep-Learning Techniques

Name: Dimitrios Loverdos
ID Number: 201056880

Email (Personal): jimloverdos@gmail.com
Email (University): cn16dl@leeds.ac.uk

Project Supervisor: Prof. Vasilis Sarhosis

Date of registration: 01/09/2019

Date of submission: 30/06/2023

Final revision: 07/10/2023

Acknowledgements

I'd like to express my gratitude, starting with my supervisor, Prof. Vasilis Sarhosis. I'm particularly thankful for his unwavering support, both professionally and personally, as well as his patience. He challenged me in ways I never thought possible. His extensive knowledge and strong ethical values make him an ideal supervisor.

I'd also like to extend my appreciation to my family for their continuous encouragement along my challenging yet rewarding path in research and academia. My heartfelt thanks go out to my mother and father, Marina Tsikalaki and Ioannis Loverdos, who have always believed in me and provided the means to pursue my dreams.

Additionally, I'm grateful to my close friends who supported my choices and stood by my side during difficult moments throughout my postgraduate journey. Especially, Christina Konstantinidou, Konstantinos Pappaioanou, Lefteris Koutsoloukas, and Sotiria Kiriakidou. We all rely on connections to help us navigate through challenging circumstances.

Lastly, I would like to express my thanks to the University of Leeds, my colleagues, and the individuals I interacted with over the years. Each person had something valuable to offer, whether it was knowledge or support in various forms. I genuinely appreciate all of you, as none of this would have been possible without your assistance and camaraderie.

Abstract

Masonry, valued for its durability and sustainability, is a common building material. However, many masonry structures have surpassed 100-year span, and show significant signs of deterioration. Therefore, inspecting and assessing these structures is crucial for their continued function and longevity. This research aims to streamline traditional inspection and assessment processes through automation. The primary contribution of this work is the development of advanced deep-learning and computer-vision techniques. These techniques enable precise detection, segmentation, and documentation of intricate masonry micro-geometry using digital data sources like images, point clouds, and reality-mesh objects. Additionally, this research involves creating precise numerical models for masonry assessment and studying how geometric accuracy affects numerical analysis. These objectives have been successfully achieved across various publications, encompassing both 2D and 3D environments.

More specifically, the initial phase of the research focused on automating the generation of geometric and numerical models of masonry structures from photographs, utilizing computer-vision methods. This process included generating CAD drawings for documentation and DEM (Discrete Element Method) models for structural assessment. The validity of this approach was confirmed by comparing it to idealized and precise numerical models of different structures. Subsequently, improvements were made by incorporating deep learning for the semantic segmentation of masonry micro-geometry into the existing workflow. This resulted in a more reliable method for detecting masonry features. This step also involved integrating an existing defect-detection model and developing a new block-detection model based on convolutional neural networks (CNNs). The enhanced workflow was further validated by comparing manually and automatically generated geometry using DEM. Finally, the research was extended to a 3D environment, where a realistic 3D model was used to generate a classified point cloud of any masonry structure. Feature detection in this 3D context benefited from CNN models for blocks and cracks, with additional classifications (mortar and other elements) estimated using image-processing techniques.

In conclusion, the developed workflows simplify a significant portion of the manual procedures involved in visual inspection, assessment, and even computer graphics generation. In most cases, the generated outputs meet the accuracy standards required for commercial use, assuming clear digital input with visible material changes. However, opportunities for improvement remain, primarily in refining detection techniques to enhance accuracy, addressing the mortar-mesh effect in numerical analysis, and achieving a solid 3D reconstruction of identified geometry to generate 3D numerical models for assessment.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

General Abbreviations	
AI	Artificial Intelligence
BCE	Binary Cross Entropy
BIM	Building Information Modelling
CAD	Computer Aided Design
CSV	Comma-Separated Values File
CNN	Convolutional Neural Network
CPU	Central processing unit
DEM	Discrete Element Method
DL	Deep Learning
DSLR	Digital Single-Lens Reflex Camera
DXF	Drawing Exchange Format
F1L	F1 Loss
FCN	Fully Convolutional Network
FEA	Finite Element Analysis
FEM	Finite Element Method
FPN	Feature Pyramid Network
GPU	Graphics processing unit
GUI	Graphical User Interface
hBIM	Historic Building Information Modelling
LiDAR	Laser Imaging, Detection, and Ranging
ML	Machine Learning
PG	Photogrammetry
RAM	Random Access Memory
RANSAC	Random Sampling Consensus
RMSProp or RMSP	Root Mean Square Propagation
ROI	Region of Interest
SGD	Stochastic Gradient Descent
WCE	Weighted Cross Entropy
Research Specific Abbreviations	
Note: *The included number indicates research publication of first appearance. I.E., (1) is publication #1, chapter 2.*	
BL (1)	Block List
CL (1)	Contour List
CCL (1)	Changed Contour List
DT (3)	Distance Transform
EPL (1)	End-Point List
GBM (1)	Generated Background Mask

GMM (1)	Generated Mortar Mask
GPM (1)	Generated Perimeter Mask
HMT (3)	H-Minima Transform
IDM (1)	Imported Damage Mask
ID (1)	Identification (list of numbers)
IL (1)	Interface List
IMM (1)	Imported Mortar Mask
IP (1)	Interface Point
IPL (1)	Interface Point List
LL (1)	Line List
LM (3)	Local Maxima
PWS (1)	Padded Watershed Array
SM (2)	Segmentation Models (Python Package)
TBL (1)	Temporary Block List
Th or Thld (1)	Threshold
WS (1)	Watershed

1. Introduction

Masonry structures are the main target of the research presented in this thesis. Although one of the oldest construction methods, it is still presently used due to its reliability and sustainability. More specifically, it is estimated that at least 40% of bridge stock in the UK is composed of masonry (Eaton, Edwards and Crapper, 2014). Where a large portion of them is beyond the 100-year mark with significant signs of deterioration, which affects their performance (McKibbins *et al.*, 2006). In many cases, they are still in use (i.e., masonry arch bridges used as railway pathways). Their structural reliability has been affected by the change of load application, environmental impact, climate-change, etc. Furthermore, a lot of those structures are listed as part of our cultural heritage and there are large efforts by the government to repair and preserve those invaluable assets (UNESCO Institute for Statistics, 2021; Brown, 2024). Additionally, preservation and renovation follows the UK strategy for greener alternatives (BEIS, 2021), in addition to the economic advantages compared to alternatives (i.e., demolish and rebuild).

Frequent inspection assists on the preservation of existing structures since it can identify defects and formulate a suitable strategy for their preservation (Sowden, 1990; Phares *et al.*, 2004; Eaton, Edwards and Crapper, 2014). However, traditional methods of visual inspection rely highly on manual recordings and human interaction. Thus, the inspection of engineering assets is an expensive and time-consuming process, especially for difficult-to-reach locations or structures with moderate-to-high traffic. More importantly, visual inspection is subjective to the individual experience of the conductor. Research has demonstrated that at least 22% of the average condition ratings, of routine highway-bridge inspection, is incorrect (Phares *et al.*, 2004). Nonetheless, the absence or improper assessment of critical engineering assets can lead to substantial economic expenses and, in some cases, even loss of human life (specifically, in the event of sudden failure).

Another important part of the evaluation of the current condition of existing structures is the manual assessment using numerical-modelling techniques (i.e., to assess the maximum load and estimate the crack-propagation). However, masonry is a complex element, with non-linear behaviour, since it is composed of multiple materials (mortar, stone, bricks, etc.). There are mainly two modelling-strategies for the assessment of masonry structures (Lourenço, 1996, 2013; Asteris *et al.*, 2015; D'Altri *et al.*, 2020).

The first is macro-modelling, where the structure or element is considered a continuum material. In which case, the formation of cracks may be shown where the maximum stresses are recorded. However, their accuracy is limited since it ignores the effect of composite nature of masonry and the masonry pattern of brickwork/stonework (Alessandri *et al.*, 2015; Iannuzzo *et al.*, 2018; Segura *et al.*, 2021). A macro-modelling alternative to continuum material is the discontinuum macro-modelling, where the continuum object is separated in multiple elements with the same material-properties (Caliò, Marletta and Pantò, 2012; Angelillo, 2019). That allows the investigation of crack propagation in terms of separation, but still ignoring the composite nature and actual pattern of masonry.

A more detailed approach is micro-modelling, which considers masonry as an assemblage of mortar-joints and masonry-units. Similarly, to the macro-modelling approach, this is also separated mainly into 2 different sub-classes. The simplest is the simplified micro-modelling, which considers the mortar-joints as a zero-thickness interface (Sarhosis and Sheng, 2014; Sarhosis *et al.*, 2014; Sarhosis, Garrity and Sheng, 2015; Forgács, Sarhosis and Bagi, 2017, 2018; Zhang, Macorini and Izzuddin, 2018; Erdogmus *et al.*, 2020). The advantage of this method is that the computational effort required is smaller, although less accurate than the alternative. Where the alternative is the detailed micro-modelling approach, which includes the simulation of the mortar with realistic size and different

material-characteristics than the masonry-units (Sarhosis and Lemos, 2018). Other research also includes the discretization of blocks, mortar, and even backfill material as a more detailed procedure (Sarhosis and Lemos, 2018; Sarhosis, Forgács and Lemos, 2019). This allows internal separation of those materials. Nonetheless, detailed-micro-modelling introduces high complexity to the geometry and numerical analysis (i.e., increases time of geometry-generation and computational-effort significantly). Thus, it is usually avoided, for the analysis of larger elements/structures, in favor of the simplified micro-modelling approach. Thus, the micro-modelling approach followed depends on a multitude of factors such as the accuracy required, available time, computational resources, etc. Even so, both micro-modelling approaches provide high simulation accuracy, compared to macro-modelling approaches.

Lastly, another important aspect of the numerical modelling of masonry is the effect of the geometric accuracy that is transferred to the numerical model. Such as potential defects, change of masonry pattern, actual mortar thickness in individual locations, actual size of individual bricks, etc. Recent research has demonstrated the effect of geometric accuracy on analysis in terms of size and shape of masonry materials (Erdogmus *et al.*, 2019; Ferrante *et al.*, 2021; Kassotakis *et al.*, 2021). However, that level of accuracy is difficult to achieve using traditional means. Thus, most researchers and engineers focus on the idealized geometry due to the effort required to produce high-accuracy geometry.

Photogrammetry and laser-scanning (LiDAR) offer a solution for the remote visualization and documentation of existing structures (Historic England, 2017, 2018). For example, photogrammetry can be used to obtain reliable information of an object by processing image-data. First conceived by "*Albrecht Meydenbauer*" for the indirect measurement of facades from photographs (Albertz, 2007). A Modern application of photogrammetry allows to obtain the 3-dimensional information from images by a technique called structure-from-motion (SfM). Where a sequence of images is used to provide depth information by tracking matched-features across different images. Photogrammetry is an inexpensive method of obtaining 3d geometry and colour of an object in space (since it only requires a camera for data capture). LiDAR is another method that yields results similar to photogrammetry, employing a laser-scanner to capture a 3D point-cloud of the scanned area. LiDAR measures the distance of a point in space with the emission of a laser-light and by measuring the time of reflection. Modern LiDAR instruments can emit more than 150,000 pulses per second for fast acquisition of georeferenced points. Depending on the model, the points recorded can include colour as well. Lastly, LiDAR has terrestrial ("*Terrestrial Scanning*" or "*Ground Based LiDAR*") and airborne applications (Schmid *et al.*, 2012).

Although the capabilities of the aforementioned methods for the documentation of existing structures are enormous, they lack automation for the extraction of geometric features to assess their condition and structural behaviour (Morer, de Arteaga and Ortueta, 2013). The research in Napolitano and Glisic (2019) shows a simple method of identifying the location of bricks with the assistance of manually placed markers. In Sithole (2008) are able to detect individual bricks of laser-scans of masonry walls using a semi-automatic approach. Where the accuracy of the detection relies on the depth-difference between mortar and brick and is best used on single plane elements. The work in Shen et al. (2018) shows a novel approach of identifying different bricks from a point-cloud of an unorganised pile of bricks. Another example is shown in Cabaleiro et al. (2017), where it shows a potential way of identifying cracks in timber beams. Recent research focuses on generating BIM and H-BIM (Historical-BIM) models (Volk, Stengel and Schultmann, 2014), with the aim to provide a useable model for common commercial packages (Andriasyan *et al.*, 2020). Recent research even takes advantage of artificial intelligence to identify the different elements of the structure (Bassier and Vergauwen, 2020).

Further research even tries to generate a useable numerical model of the scanned structures for assessment (Barazzetti *et al.*, 2015; Korumaz *et al.*, 2017; Bassier *et al.*, 2019; Rolin *et al.*, 2019; Funari

et al., 2021; Pepi *et al.*, 2021); using the Finite Element Method (FEM). However, those methods consider the macro-modelling approach, which may not have sufficient accuracy for masonry structures. A different use of 3d point-clouds, for the generation of numerical models, is voxelisation (Hinks *et al.*, 2013; Kassotakis and Sarhosis, 2021). Where the 3d-point-cloud is converted to a collection of solid cubes that replicate loosely the geometry. This allows the investigation of crack propagation since the voxels are separable. However, voxelised models have reduced accuracy to more complex alternatives (i.e., micro-modelling), since they do not consider the masonry-pattern or other details on the geometry (i.e., existing defects).

The human eye can easily detect edges and shapes on an image without much effort. On the other hand, a computer can only perceive an image as an array of different numbers. For that reason, the use of complex algorithms is required to be able to acquire additional information from image-data. The family of those algorithms is typically referred to as image-processing.

Feature detection refers to the ability to detect certain characteristics on an image by scanning an area (usually 3x3), using a dedicated algorithm. The most renown algorithm of that purpose is the canny edge detection (Canny, 1986). The technology behind edge detection is still evolving with more algorithms promising better results (Martin, Fowlkes and Malik, 2003, 2004; Arbeláez *et al.*, 2011; Bora, 2017). Even with the aid of machine-learning but with the disadvantage that the algorithm requires sufficient training to manually-annotated data (Martin, Fowlkes and Malik, 2003, 2004). However, the simplicity and performance of the canny edge detection are still appreciated and used to this day. A different approach is presented in Brackenbury and Dejong (2018), which investigates the mortar detection of masonry elements. The image is filtered using an edge-detection algorithm based on the Sobel kernel to locate possible edges followed by Hough-transform to identify the lines of the bed-joints. However, since their approach is using the Hough-transform, the detection of features is limited to straight-lines. A similar example is provided in Oses, Dornaika and Moujahid (2014), where the authors detect line segments on masonry walls, using existing image-processing techniques. After delineation, the result is enhanced using artificial-intelligence. Although applied on 3d point-cloud, the research in Valero, Bosché and Forster (2018) describes a reliable method to detect bricks on masonry-walls using image-processing. They used the 2D Mexican-hat-wavelet (MHW) to detect features on the depth-map (2d array). Excess noise is removed under the assumption that every brick is enclosed within its convex-hull. Thus, the main limitation of the proposed approach is that it cannot define properly concave elements. Another approach that does not use typical photographs is shown in Cluni et al. (2015), where the authors use thermographic images to identify the location of bricks and mortar with the aim to evaluate the structural response for modal analysis.

However, the detected edges cannot be used to define the location of the individual elements of the structure in that state. Segmentation is a method to divide an image into smaller groups (i.e., bricks, mortar, cracks, openings), which are easier to manipulate and measure. A solution to convert detected features to segmented areas is by the use of a technique called watershed-transform. The process is considering the image as a topological map with the intensity value of the pixel representing the height. Peaks and valleys are the local maxima and local minima of the image respectively. There are two main approaches to watershed segmentation, “watershed by flooding” and “watershed by rain-falling”. “Watershed by flooding” is locating the valleys of the image and continues by flooding the surrounding areas gradually. The flooding stops when the peak of the surrounding ridge is reached or when two flooded areas meet. In the case of two flooded areas meeting, a dam of zero thickness is erected that separates the regions. The “watershed by rain-falling” works differently. The main difference is that the watershed is simulating water-falling where the water dropping along the surface of the image flows towards the valley following the steepest path (Kornilov and Safonov, 2018). Over the years a great number of different watershed algorithms have been proposed. The version of the algorithm most commonly used is provided in Beucher, S.; Meyer (1993), which is based on the “watershed by flooding” method initially proposed by Vincent and Soille (1991).

There are many algorithms that can be used to convert the information acquired from the segmentation into manageable data. One example is border-following, which can be used to acquire the perimeter of objects in binary-images (Suzuki and Abe, 1985). In this case, the perimeter is acquired as a collection of vertices. Border-following can be paired with line-generalisation to reduce the number of vertices on the polyline and retain the most significant to describe its geometry. A common generalisation method is the "Ramer-Douglas-Peucker" algorithm, which is based on a vertical-threshold to filter the points of the polyline (Ramer, 1972; Douglas and Peucker, 1973).

The main issue with the use of simple image-processing operations, for feature-detection, is that they are highly affected by the change in illumination, colour, texture, and resolution. In all except ideal cases, their performance is not appropriate for complete-detection. An alternative approach proposed is the use of machine-learning (M.L.), where an algorithm can be trained to identify the desired locations of micro-geometry of masonry.

A subclass of M.L., Deep Learning (D.L.) is trying to replicate the natural ability of the human-brain to learn. The disadvantage of this method is that the algorithm requires a large amount of labelled data for adequate performance (Bagińska, Srokosz and Srokosz, 2019). There are different types of D.L architectures. Convolutional-Neural-Networks (CNN) are often used for image classification and segmentation. CNN architectures are typically composed of multiple layers of convolutions (acting as feature detectors), followed by fully connected layers of neurons (for classification). However, for semantic-segmentation usually FCN (Fully-Convolution-Network) is used instead of CNN (Long, Shelhamer and Darrell, 2015). The difference is that FCN replaces the fully-connected-layers (neurons) with convolutional layers. The advantage of FCN over CNN is that they allow the use of any image-size as input. This is due to the fact that the fully connected layers require a specific number of inputs, while convolutions do not have this limitation. Some of the commonly used FCN architectures include U-Net, DeepLabV3+, LinkNET, FPN, and many more (Chen *et al.*, 2015, 2018; Ronneberger, Fischer and Brox, 2015; Lin *et al.*, 2017; Chaurasia and Culurciello, 2018).

Often a CNN or FCN architecture is paired with a backbone network that acts as feature extractor. It is placed prior to the main architecture, with the aim to improve classification and segmentation accuracy. Examples of common backbone architectures include VGG, ResNet, Inception, etc. Another common approach to increase the accuracy of the model is transfer-learning, where a model is pretrained to a different dataset with the aim to learn to detect complex-features from the image. Then, the model is trained as normally to the actual dataset. Transfer-learning can be used in both CNN and FCN architectures and has shown to boost the performance of models trained to smaller datasets (Hussain, Bird and Faria, 2019; Dais *et al.*, 2021).

Machine learning is often used in engineering projects. Regarding masonry, one common application of M.L. is to assist with the defect detection of heritage structures (Valero *et al.*, 2019). Especially in the case of CNN/FCN for either region classification (Ali, 2019; Wang *et al.*, 2019), patch classification (Chaiyasarn *et al.*, 2018; Brackenbury, Brilakis and Dejong, 2019), or semantic segmentation (Kalfarisi, Wu and Soh, 2020; Dais *et al.*, 2021), directly on the input-image. Where region classification is based on a region proposal system and classifies the selected regions, patch classification classifies small patches on the image individually, and semantic segmentation classifies every pixel on the image. There are also efforts to transfer the detected defects to 3D environment for improved visualisation (Kalfarisi, Wu and Soh, 2020). Although not as common as defect detection, recent research demonstrates the potential of deep-learning for the use of brick/stone detection on masonry structures (Ibrahim, Nagy and Benedek, 2019; Ergün Hatir and İnce, 2021; Loverdos and Sarhosis, 2022a). Deep-learning may consider region-classification (assuming rectangle units parallel to the ground-plane; (Ergün Hatir and İnce, 2021)), or semantic-segmentation (Ibrahim, Nagy and Benedek, 2019; Loverdos and Sarhosis, 2022a). Nonetheless, the complete identification of the micro-geometry of masonry is important if the aim is complete visualisation and evaluation of the structural condition.

1.1. Research-Gap Analysis

Nowadays, data gathering is simpler than ever due to recent advances in the technology of remote sensing (LiDAR, Photogrammetry, smartphones, thermographic cameras, drones, etc.). However, those data are usually processed manually to generate i.e., the numerical model (suitable for DEM) or the classification of the different elements for use in BIM programs (i.e., walls, floors, etc). The research-gap, identified in the literature, is provided below:

- The inspection and assessment of existing structures is traditionally a visual and manual process (i.e., hammer tapering, measuring of deflection, documentation of defects), that comes with large monetary costs, requires extensive time, and relies on expertise personnel. Furthermore, certain locations may prove challenging to investigate (i.e., vital infrastructure, tall structures, etc.).
- There is no work in the literature to generate the "as is" geometry of masonry structures for documentation or analysis. Although there is work showing the application of binary-structures for the generation of numerical-models, their application is limited to specific shapes (i.e., rectangle walls; (Tiberti and Milani, 2019)). Additionally, although there are studies that convert 3D-scans to numerical models, those are limited to macro-modelling techniques using FEM for analysis (Funari *et al.*, 2021) or DEM for voxelised structures (Kassotakis and Sarhosis, 2021).
- Image-processing techniques proved inadequate to identify the micro-geometry of masonry in most cases (Oses, Dornaika and Moujahid, 2014) due to low visual-quality of pictures, blurry edges between materials, change in illumination/colour/texture, etc. Furthermore, those techniques are incapable of identifying the composition of a structure (i.e., concrete, steel, masonry, openings, background, etc.).
- The detection of defects in all type of constructions, including masonry, is a common research-project. However, the identification of their geometric properties is usually neglected. Even though, the evaluation of defects has substantial commercial applications, i.e., by assisting with the manual-process of inspection and decision-making regarding restoration and maintenance of heritage structures.
- Lastly, there is very limited work in transferring the available classifications of structures (using for instance CNN) to 3D environment for documentation (Kalfarisi, Wu and Soh, 2020). Additionally, the process of classification on 3D environment usually requires specialised data that is hardly available on demand (3D point-clouds), their generation requires large effort for classification (i.e., manual classification of structural-elements in 3D point-clouds) and technology that may not be available due to budget concerns (i.e., laser-scanners).

1.2. Aim and Objectives

The research in this document represents the effort to take advantage of visual-data to generate the geometric/numerical-model of masonry structures (automatically or semi-automatically), including the identification and measurement of detected-defects. Those are aimed at assisting engineers with the visual-inspection, documentation, and rehabilitation of heritage structures of masonry. The main objectives of the research are highlighted below.

1. **[Publication #1]:** Develop a workflow for the generation of 2D geometric/numerical-models of masonry from image-data, including the definition of damaged areas.
2. **[Publication #2]:** Define a method to identify the masonry-units from image-data, more reliably than simple image-processing operations.
3. **[Publication #2]:** Evaluate the use of deep-learning for the detection of masonry-units from image-data. Identify the most suitable network-architecture for the purpose of the study and define the hyperparameters to maximise validation-accuracy.

4. **[Publication #2]:** Combine block-detection and defect-detection to a single image-output for complete visualisation of the structural condition.
5. **[Publication #3 & 4]:** Combine the detection of the micro-geometry of masonry using CNN/FCN, with the workflow to generate the geometric/numerical-model of masonry structures.
6. **[Publication #3 & 4]:** Define the remaining aspects of the masonry-geometry (i.e., openings, background, other structural elements) using the CNN/FCN output combined with image-processing.
7. **[Publication #3 & 4]:** Define a precise method for the evaluation of the geometric-properties of cracks in 2D environment.
8. **[Publication #5]:** Develop a workflow to transfer the classification of the micro-geometry of masonry from 2D to 3D models (reality-mesh).
9. **[Publication #5]:** Define a precise method for the evaluation of the geometric properties of cracks in 3D environment.

1.3. Contributions

The overall research is a novel approach to the visualization and assessment of masonry structures, which offers a number of contributions to the scientific community. The complete list of contributions and knowledge generated is the following:

1. A novel workflow developed to acquire the 2D digital-representation of the geometry of a masonry structure, from image-data. The workflow also includes the automatic generation of numerical models in DEM.
2. Evaluation of the validity of the "as is" geometry for numerical analysis (using DEM), compared to the commonly used "idealized" geometry.
3. Generation of sufficient dataset for training/validation of artificial intelligence models for semantic segmentation of masonry brick elements.
4. Evaluation of different CNN architectures, hyperparameters, loss functions, and optimizers for the semantic segmentation of masonry brick elements.
5. Identification of the optimal combination of CNN architecture, hyperparameters, loss function, optimizer, etc., for the detection of brick elements.
6. A novel workflow for the generation of a classified point-cloud from photorealistic 3d models of masonry structures.
7. A novel workflow for the measurement of the geometric properties of detected defects, in both 2D and 3D environments.

1.4. Organization of the Thesis and Publications

The main body of the thesis includes the 5 peer-reviewed journal-publications, finalised during the postgraduate research period of 2019-2023. Those are provided in order of date of publication, where the main objective of each publication is to progress the initial research project (by improving the automation and main-capabilities of the developed algorithms). In more detail, the complete structure of the report is provided below:

1.4.1. Chapter #1: Introduction

Includes the combined literature review of all publications, in logical order to describe the research progression. Furthermore, it includes the research-gap analysis, aim, objectives, and organisation of the thesis.

1.4.2. Chapter #2: Publication #1 (Loverdos et al., 2021a)

The 1st publication demonstrates a novel approach to convert binary-images of masonry structures to CAD drawings. Both blocks and cracks are detected using image-processing methods (i.e., canny-edge-detection and image-thresholding). The generated-geometry is transferred to a numerical-model, by the conversion of DXF-lines to line-commands, for use with the commercial program UDEC. The main disadvantage of the developed workflow is that it relies heavily on the quality of the binary-image generated using image-processing. It has been perceived that a large number of images of masonry cannot be converted to binary-images with adequate quality for feature-extraction (solved in publication #2). Additionally, each processed image requires a significant amount of time to generate since it requires the modification of the image-processing parameters. Another issue with the work presented is that the mortar-mesh, used to investigate crack propagation, was placed manually (solved in publication #3). Thus, some mortar-mesh-elements had concave shape, which could cause overestimation of collapse-loads (due to interlocking).

1.4.3. Chapter #3: Publication #2 (Loverdos and Sarhosis, 2022a)

The 2nd publication evaluates the use of CNN for the automatic detection of masonry-units. Multiple network-architectures have been tested with varied configuration of the hyperparameters involved, to maximise accuracy. The result is combined with a crack-detection model developed by Dais et al. (2021), for visualisation. The aim of combining block and crack-detection using CNN is to provide a robust method for the detection of the micro-geometry of masonry, to replace the image-processing operations in the 1st publication.

1.4.4. Chapter #4: Publication #3 (Loverdos and Sarhosis, 2023b)

The 3rd publication combines the work presented in the 1st and 2nd publications and provides further improvements to the algorithm in terms of automation. It defines a complete workflow for the generation of geometric/numerical models of masonry structures from image-data. This includes the future-detection of the micro-geometry of masonry using CNN (from 2nd publication), an updated method for the feature-extraction of shapes as polylines (from 1st publication), automatic generation of convex-mesh elements to investigate crack-propagation (from 1st publication), and an accurate method for the measurement of detected cracks.

1.4.5. Chapter #5: Publication #4 (Loverdos and Sarhosis, 2023c)

The work presented in the 4th publication demonstrates the conversion of the algorithms, developed in the 3rd publication (Loverdos and Sarhosis, 2023b), to a self-sufficient program for industrial use. This includes the development of a complete graphical-user-interface (GUI), pop-up windows for the registration of parameter-values, etc. The publication is mainly aimed to developers, rather than engineers, that are interested in the applied-code and presentation of the program instead of the algorithmic approach.

1.4.6. Chapter #6: Publication #5 (Loverdos and Sarhosis, 2023d)

The work presented in the 5th publication transfers part of the previous research in a 3D environment. It demonstrates a novel methodology for the generation of a classified point-cloud, from an existing 3D model of a masonry-structure (reality-mesh). The initial classification is applied on off-screen renders, captured automatically or semi-automatically, from the reality-mesh itself. The classification is applied using existing CNN models (Dais *et al.*, 2021; Loverdos and Sarhosis, 2022a), on the 2D renders instead of the point cloud. Furthermore, it describes the methodology to convert the detected-cracks from points to mesh-elements, to measure their geometric properties.

1.4.7. Chapter #7: Discussion

The work presented in the supplementary chapter, is the overall discussion of the results of the presented thesis. More specifically, the discussion chapter includes the application of artificial intelligence, development of 2D workflow, and development of 3D workflow. Lastly, it includes the

validation of the final-version of the 2D workflow, developed and improved during the 4 first publications. The validation is considering the comparison of the “idealised” and “as is” geometry, for numerical analysis using DEM.

1.4.8. Chapter #8: Conclusions

The final chapter provides the conclusions of the overall research. Those include the summary, possible applications, advantages and disadvantages, future work, and possible improvements to the workflows proposed.

1.4.9. Appendix

Includes all publications, which are 5 peer reviewed journal-articles as 1st author (Loverdos *et al.*, 2021a; Loverdos and Sarhosis, 2022a, 2023c, 2023b, 2023d), 2 peer-reviewed journal-article as 2nd author (Ferrante *et al.*, 2021; Vandenabeele *et al.*, 2023), a conference article as 1st author (Loverdos and Sarhosis, 2023a), and lastly a conference article as 3rd author (Muhit *et al.*, 2023). Those are provided separated, as individual uploads. For a complete list of publications, use the following link.

ResearchGate Link: https://www.researchgate.net/profile/Dimitrios-Loverdos/research

2. Paper #1: An innovative image processing-based framework for the numerical modelling of cracked masonry structures.

Dimitrios Loverdos[a], Vasilis Sarhosis[a,*], Efstathios Adamopoulos[b], Anastasios Drougkas[a]

[a] *University of Leeds, School of Civil Engineering, Woodhouse Ln, Leeds LS2 9DY, United Kingdom*
[b] *University of Turin, Department of Computer Sciences, Corso Svizzera 185, Turin 10149, Italy*

Abstract

A vital aspect when modelling the mechanical behaviour of existing masonry structures is the accuracy in which the geometry of the real structure is transferred in the numerical model. Commonly, the geometry of masonry is captured with traditional techniques (e.g., visual inspection and manual surveying methods), which are labour intensive and error-prone. Over the last ten years, advances in photogrammetry and image processing have started to change the building industry since it is possible to capture rapidly and remotely digital records of objects and features. Although limited work exists in detecting distinct features from masonry structures, up to now there is no automated procedure leading from image-based recording to their numerical modelling. To address this, an innovative framework, based on image-processing, has been developed that automatically extracts geometrical features from masonry structures (i.e., masonry units, mortar, existing cracks, and pathologies) and generate the geometry for their advanced numerical modelling. The proposed watershed-based algorithm initially deconstructs the features of the segmentation, then reconstructs them in the form of shared vertices and edges, and finally converts them to scalable polylines. The polylines extracted are simplified using a contour generalisation procedure. The geometry of the masonry elements is further modified to facilitate the transition to a numerical modelling environment. The proposed framework is tested by comparing the numerical analysis results of an undamaged and a damaged masonry structure, using models generated through manual and the proposed algorithmic approaches. Although the methodology is demonstrated here for use in discrete element modelling, it can be applied to other computational approaches based on the simplified and detailed micro-modelling approach for evaluating the structural behaviour of masonry structures.

Keywords: masonry, image processing, watershed transform segmentation, feature extraction, numerical modelling, DEM.

***Corresponding author:** Dr V. Sarhosis, University of Leeds, email: v.sarhosis@leeds.ac.uk

2.1. Introduction

Assessing the structural performance of ageing masonry structures is a difficult task. Over the last three decades, significant efforts have been devoted to developing numerical models to represent the complex and non-linear behaviour of existing unreinforced masonry structures subjected to external loads. Such models range from considering masonry as a continuum (macro-models) to the more detailed ones that consider masonry as an assemblage of units and mortar joints (micro-models or meso-scale models), see (Lourenço, 1996). Since old and deteriorated masonry is typically characterised by low bond strength (Sarhosis and Sheng, 2014), cracking is often a result of the masonry units' de-bonding from the mortar joints. Given the importance of the masonry unit-to-mortar interface on the structural behaviour of aged masonry structures, micro-modelling approaches (i.e., those based on Discrete Element Method; in which the mortar is described as zero-thickness interfaces between the masonry units) are better suited for simulating their serviceability and load carrying capacity (Sarhosis and Sheng, 2014; Sarhosis, Garrity and Sheng, 2015). A vital aspect when modelling masonry structures, based on the micro-modelling approach, is the accuracy in which the geometry and material performance characteristics are transferred in the numerical model (Forgács, Sarhosis and Bagi, 2017, 2018). Even though current numerical modelling strategies for masonry are focusing primarily on idealised geometry (Asteris *et al.*, 2015), examples in the literature (e.g., (Erdogmus *et al.*, 2019)) demonstrate that a more representative visualisation of the masonry leads to more accurate results.

Some efforts are being made by the scientific community to accurately capture the geometrical characteristics of masonry structures using traditional techniques (e.g., on-site inspection and manual surveying methods). However, such methods have been found to be labour intensive and error-prone(Zhang, Macorini and Izzuddin, 2018). Over the last ten years, advances in laser scanning and photogrammetry have started to drastically change the building industry since similar techniques are able to capture rapidly and remotely digital records of building elements and features in three-dimensional (3D) point-cloud and ortho-image format (Morer, de Arteaga and Ortueta, 2013; Altuntas, Hezer and Kırlı, 2017; Napolitano and Glisic, 2019). However, current approaches for extracting geometrical features (i.e., size and positioning of masonry units, location, and size of cracks, etc.) from imagery, lack automation, while protocols for the systematic generation of meso-scale models for assessing the structural behaviour of masonry structures are absent. Thus, even today, the feature extraction of masonry units' geometry is done manually using either computer-aided design (CAD), image, or point-cloud-based approaches (Morer, de Arteaga and Ortueta, 2013; Napolitano and Glisic, 2019). Point-based voxelisation of point-cloud data offers a possible solution by allowing the generation of discretised models (Hinks *et al.*, 2013). Even so, voxelisation methods do not consider the effect of the masonry-unit geometry or physical defects on the numerical model.

Image-processing approaches can offer various solutions to the problem of creating simplified records of masonry structures suitable for meso-scale modelling via the use of feature-detection (Canny, 1986; Martin, Fowlkes and Malik, 2003; Arbeláez *et al.*, 2011; Bora, 2017) and segmentation techniques (Beucher, S.; Meyer, 1993; Arbeláez *et al.*, 2011; Kornilov and Safonov, 2018). Research in automated detection and segmentation of masonry elements has drawn much attention by the scientific community (Sithole, 2008; Brackenbury and Dejong, 2018). Additionally, research in defect localisation using artificial intelligence also offers alternative methods to identify the extent of damage present on masonry structures, with high-level of automation (Chaiyasarn *et al.*, 2018; Valero *et al.*, 2019; Dais *et al.*, 2021). Feature-detection of masonry elements presents a challenging task due to the anisotropic radiometric characteristics of the imagery or ortho-imagery involved. Raw images or photogrammetric derivatives, depending on the method of acquisition, may be of low quality for the automated detection of features. That includes the on-image sharpness of the edges of masonry units and defects, efficient contrast between masonry units and interfaces, and the effect of surface degradation on capturing the necessary radiometric data.

However, recent studies have demonstrated the efficient application of image-enhancing algorithmic implementations, with the purpose to improve the radiometric quality of digital records of masonry, facilitating the extraction of geometric features of their structural elements (Valero, Bosché and Forster, 2018; Ibrahim, Nagy and Benedek, 2019). Advanced solutions have also considered the use of infrared thermography as a primary sensing technique for the interpretation of the structure of masonry (Cluni *et al.*, 2015). Those recent developments establish the use of image-based applications as a viable solution for the mechanical evaluation of masonry elements. A similar notion is presented in (Tiberti and Milani, 2019, 2020a, 2020b), where it contemplates the use of binarised-images for the automatic construction of a voxel/pixel heterogenous pattern for the limit-analysis of irregular masonry. However, despite the rationale of identifying the structural composition characteristics of masonry in a cost and time-effective way, the practical use of feature and defect-detection is rarely used for the automated generation of discreet numerical models.

To resolve the commonly discussed topic of masonry evaluation using image-obtained data, this study proposes an automated methodological approach for numerical model generation using the output of image-processing applications. The main objective is the geometric feature-extraction from masonry structures (e.g., masonry units, mortar, and damage pathologies) using an innovative watershed segmentation approach. The methodology proposed in this document is part of a holistic framework that aims to automate fully the generation of masonry models from point-cloud data (PCD) and imagery data (Fig. 2.1). Although the approach is demonstrated here for use in discrete element modelling, it can be applied to other computational approaches for evaluating the structural behaviour of masonry structures.

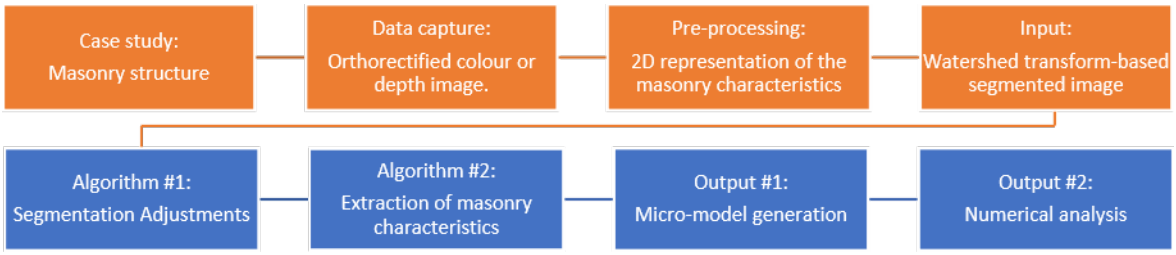


Fig. 2.1: Suggested workflow of the overall framework. The work presented in this document is shown in blue (Second row).

2.2. Segmentation Adjustments

The purpose of this section is to describe the refinement procedure followed to correct spatially the characteristics of masonry segmentation (Fig. 2.2), which can run as input for the accurate description of the masonry geometry in the structural analysis model. The procedure commences with a watershed segmentation-derived input and considers the actual geometric characteristics of both mortars and cracks in masonry to correct segmentation issues caused by geometric irregularities (Fig. 2.2: Steps 3 and 4).

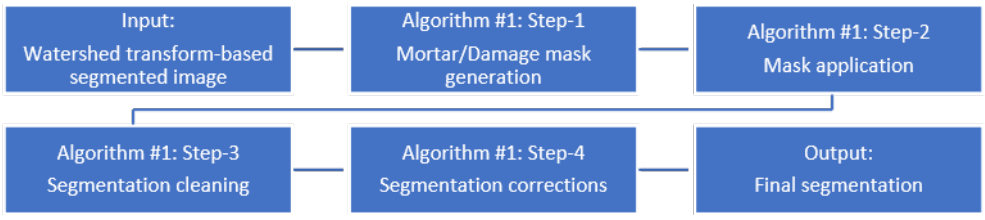


Fig. 2.2: Workflow of the algorithm responsible for the segmentation modifications (Algorithm #1).

Segmentation is often used in image processing to reduce the amount of available data on an image from pixels to regions. The segmentation technique considered is the marker-based watershed-transform, due to its innate ability to produce closed-regions. The methodology proposed is aimed to be used in combination with algorithms that can produce good binarisation. However, if the contrast

between building blocks and their interfaces on the source imagery is adequate, typical image processing techniques can be applied (Fig. 2.3(d)). Additionally, If the input includes background information, it should be removed during the pre-processing stage. Moreover, a background of uniform colour can be used to limit the segmentation by generating the background-mask (Fig. 2.3(c)). After an appropriate binarised image is applied, a morphological operation can be used to detect the local minima (Fig. 2.3(e)). In which case, H-minima transform is often used to remove false local-minima and prevent over-segmentation (Fig. 2.3(f)). The local-minima of a modified distance-transform provides the markers of the watershed (Fig. 2.3(g)). Figure 3 shows a typical procedure of watershed-segmentation.

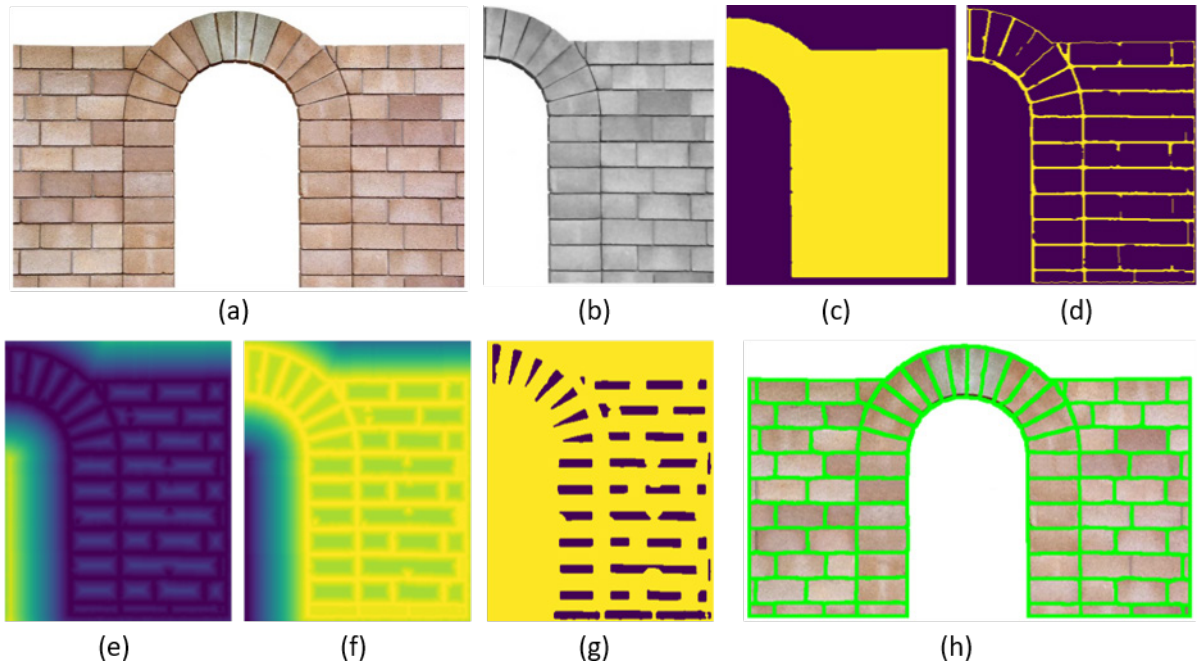


Fig. 2.3: Marker-based watershed segmentation, (a): Image source; (b): Bilateral blurring on greyscale to reduce noise and retain edges; (c): Background mask by global-thresholding; (d): Canny edge-detector applied on the filtered image (after erosion/dilation); (e): Inverse distance-transform for the markers; (f): H-Minima transform to remove false minima; (g): Local-minima for watershed-markers (after dilation/erosion); (h): Segmentation lines.

2.2.1. Mortar and Damage Mask Generation

Considering the labelling convention of the watershed segmentation, the proposed algorithm generates a mortar-mask based on the numerical values of the watershed array, where the background label is marked with zero values, while the segmented areas are positive. The input for the mask generation process is the initial watershed array, padded by 1-pixel with zero values in all four directions. That allows to scan the image using a 2x2 Region of Interest (ROI). The size of 2x2 ROI corresponds to the minimum size required to detect an area where multiple labels are present. The following are the steps to define the network of mortar interfaces:

1. *Generated Mortar Mask (GMM)*: The mortar mask is generated using a 2x2 ROI to scan the image for inner interfaces (i.e., where the 2x2 ROI has two or more unique values, and all values are larger than zero) (Fig. 2.4 (b)).
2. *Imported Mortar Mask (IMM)*: Optionally, the original rasterised output of the feature-detection, can be included to provide with minor corrections to the GMM (Fig. 2.4(c)). However, the rasterised image will contain perimetral edges that should be removed.
   2.1. *Generated Perimeter Mask (GPM)*: When the original raster image is used, the perimeter mask is generated (i.e. where the 2x2 ROI has two or more unique values and contains at least one zero) which aims to remove the perimetral edges from the IMM (Fig. 2.4(d)).

3. *Generated Background Mask (GBM)*: Excessive mortar caused by the dilation of GMM and IMM is removed by the background mask (i.e. where the padded watershed array has zero values), the GBM is applied to the final segmentation (Fig. 2.4(e)).

The padding of all masks contains values of ones, except the GBM that contains zeros (*Fig. 2.4*). This allows the background to reduce the segmentation, even if it envelopes the entire image. The initial line-thickness of each mask that uses a 2x2 ROI is at least 2-pixels, since the entire ROI is transferred to the mask. The 2-pixel thickness is required due to the segmentations being connected, and a mortar of 1-pixel thickness would reduce the size of a masonry unit in one side. The size of each mask is controlled by erosion/dilation, which effectively adjusts their effect. The GMM is adjusted manually to represent the average mortar thickness. The GBM and IMM are generally used as generated without modifications to their thickness. However, the GPM is given an excessive value to remove the perimeter of the imported-mortar-mask, given that it does not override succeeding mortar layers. The option to adjust each mask using morphological erosion and dilation allows the fine-tuning of the final result if it is required.

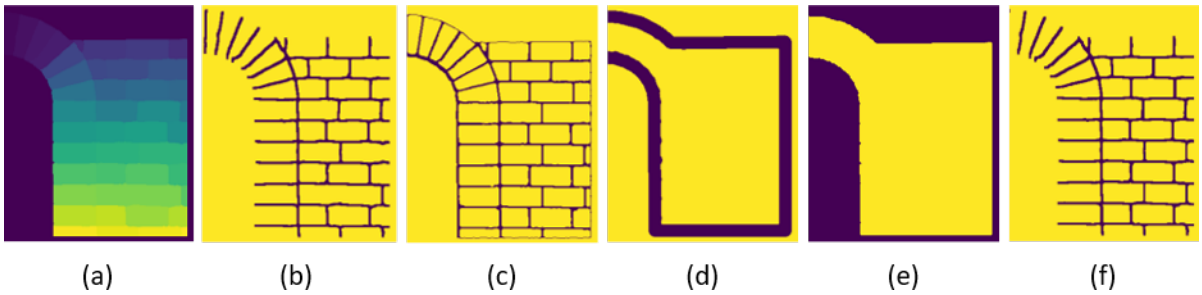


Fig. 2.4: Raster masks; (a): Initial watershed segmentation; (b): Generated-mortar (GMM); (c): Imported-mortar (IMM); (d): Generated-perimeter (GPM); (e): Generated-background (GBM); (f): Final mortar (FMM= IMM+GMM).

If the damage (i.e., cracking) in masonry is provided as a raster image (IDM, Fig. 2.5(d)), it can be applied to the final segmentation before the background mask. However, the use of external masks should be adjusted or avoided; if they are inaccurate (i.e., excessive noise, false detection of damage, etc.). If not, they may create inappropriate discontinuities on the material (Fig. 2.5(e)) and may cause incorrect estimation of collapse loads during the numerical analysis. Finally, a unique label is applied by each mask, on the final segmentation, used by the feature extraction method to identify different locations (i.e. $mortar = -1$; $damage\ in\ mortar = -2$; $damage\ in\ masonry\ unit = -3$). Assuming that the accurate mortar (GMM, IMM) and damage (IDM) location is provided, the different damage states will indicate different damage types (i.e., crack on mortar, crack on brick, loss of material due to spalling or excessive cracking, etc). If not, they will only indicate the prior label before the damage is assigned to the affected location (Fig. 2.5(f)).

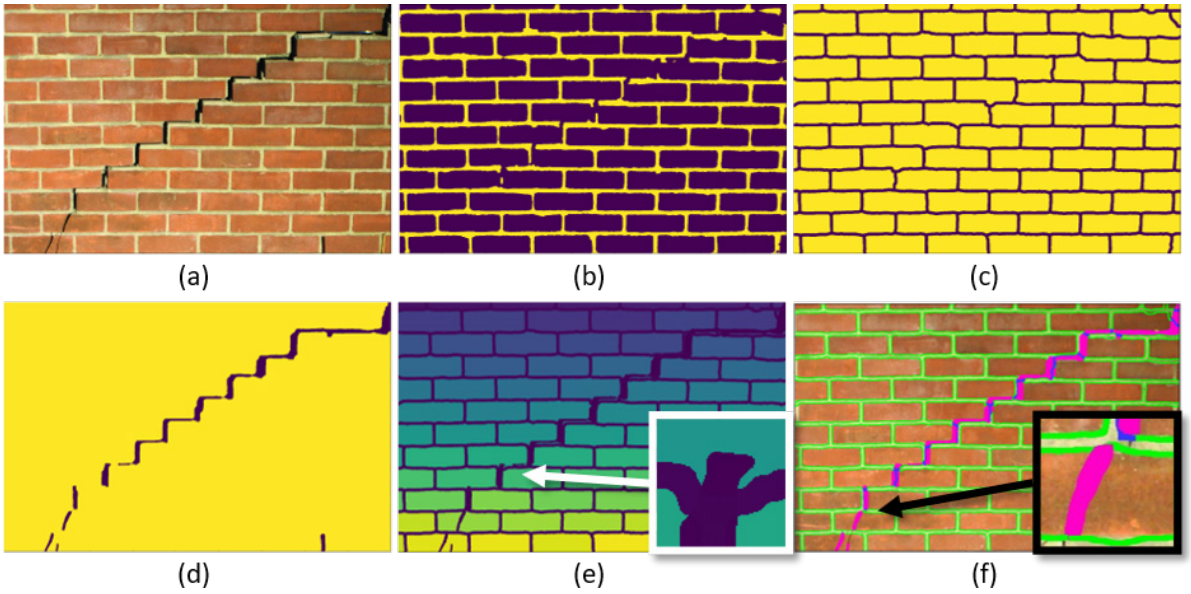


Fig. 2.5: Watershed segmentation demonstrating the name convention (GMM & IDM); (a): Source image; (b): Initial rasterised image; (c): GMM; (d): IDM; (e): Watershed segmentation with damage; (f): Watershed lines with damage states (Blue: Damage on mortar pixels, Magenta: Damage on brick pixels).

2.2.2. Segmentation Cleaning and Correction

The application of external masks, on the watershed array, may cause the isolation of individual pixels or separation of a segmentation into multiple objects (i.e., Fig. 2.5(d)). Their existence must be corrected before the feature extraction as it may cause issues with the definition of each block.

The first step, towards the correction of the segmentation, is the elimination of isolated pixels (Fig. 2.6). Pixels that do not have a 4-connectivity (i.e., no diagonal connectivity) with a label are considered "isolated". An "isolated" pixel may cause the erroneous description of a masonry units' perimeter. For that reason, they are replaced with the most common label of its neighbour pixels (Fig. 2.6(c) & Fig. 2.6(d)). All values of the 3x3 ROI are considered, but the pixel is replaced only by a label that has 4-connectivity with it.

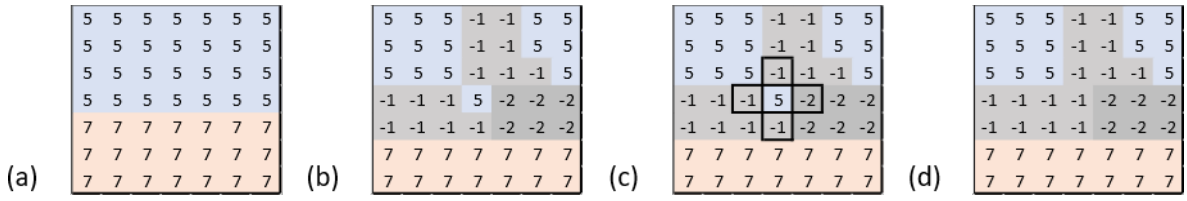


Fig. 2.6: Segmentation-cleaning; (a): Initial segmentation; (b): After mask application; (c): Detection of isolated pixels; (d): Replacement of isolated pixels with the most common value of the 3x3 ROI (limited to 4-connectivity labels).

The second and final step, for the correction of the segmentation, is the separation of duplicate objects (Fig. 2.7). If two or more segmentations have the same label, they are considered duplicates. Duplicate objects may cause conflicts during the feature-extraction. For that reason, duplicate objects are provided with a new label. Initially, each zero/positive label is isolated on a new-array (Fig. 2.7(b)) with a size equal to the original watershed padded by 1-pixel of zero values. The new-array contains zeros and ones, where one is the segmentation label examined. The watershed segmentation is then applied using as a mask and source the new-array and the inverse new-array to acquire the local-minima for the markers (Fig. 2.7(c)). After the first segmentation, any subsequent labels are assigned a new value equal to the existing maximum plus one (Fig. 2.7(d)). The command used for the markers should consider only pixels with 4-connectivity to separate segmentations that are not connected vertically or horizontally. Doing so also solves the issue where two isolated pixels are located side-by-side and thus not detected by the segmentation-cleaning process demonstrated above (Fig. 2.6). Only

zero and positive labels are verified during this step. Furthermore, any modified label is stored in the Changed-Contour-List (CCL), with its prior label to retain the previous state/type of the segmentation (i.e., Blocks, Background, etc.). The structure of the Changed-Contour-List is provided below:

$$CCL_n = [New\ Label, Old\ Label] \quad (1)$$

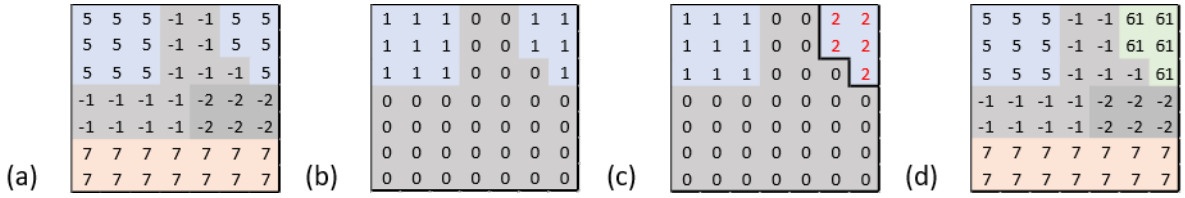


Fig. 2.7: Segmentation-corrections; (a): Masked segmentation; (b): Isolation of label #5 in a new array; (c): Watershed segmentation and detection of additional labels; (d): Re-labelling of additional segmentation.

2.3. Feature Extraction

The second part of the proposed framework is the extraction of a network of lines and nodes, which will represent the simplified geometry of the interfaces. The final output is a collection of polylines that will be used for the numerical model generation. The input of this section is the modified segmentation acquired previously (Algorithm #2, see Fig. 2.1). It initiates by scanning the watershed array to extract the coordinates in-between segmentations using a novel approach (Fig. 2.8: Step-1). The correct order of the extracted pixels is provided by using a border-following algorithm (Fig. 2.8: Step-2). Additionally, it uses a generalisation algorithm to reduce the number of vertices that describe an interface (Fig. 2.8: Step-3). Furthermore, it includes geometric operations to adjust the location of selected vertices and produce more accurate results (Fig. 2.8: Step-4).

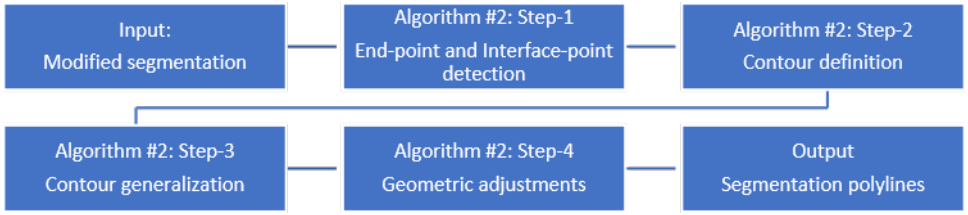


Fig. 2.8: Workflow of the algorithm responsible for the feature extraction (Algorithm #2).

2.3.1. Point Detection and Contour Definition

The input of the point-detection algorithm is the watershed array padded by 1-pixel of zero values in all directions (i.e., from 100x100 to 102x102, (Fig. 2.9(a)), to allow scanning using a 2x2 ROI. Moreover, the geometric definition of damage is excluded at this stage; since the same labels have been assigned to multiple segmentations and may cause false detection of the perimetrical characteristics of masonry units. The simplest solution is to apply the point detection on a padded watershed array (PWS), where all damage labels are replaced with mortar (i.e., $\mathrm{PWS}(< -1) = -1$). Initially, the watershed array is scanned using a 2x2 ROI under the following conditions:

1. *Interface-Point*: If the 2x2 ROI contains two or more unique values, it is considered an interface-point (Fig. 2.9(a)). In which case, it is saved on the interface-point-list (IPL), with its ID and location.
2. *End-Point*: If the 2x2 ROI contains three or more unique values, it is considered an end-point (Fig. 2.9(c)). In which case it is saved on the end-point-list (EPL), with its ID and location.

The ID is the ordered list of unique values from the ROI. The location saved for each point detected is the top-left location of the 2x2 ROI. Each collection of points of a unique ID (of two components), makes an interface with known end-points, and their location is saved on the interface list (IL). End-points have three or more ID values and are stored to all appropriate interfaces (where they have two

common values). Practically, the point-detection does not consider the location of pixels, but the gridlines instead. This method was selected to avoid the separation between segmentations. For that reason, the coordinates include an additional unit in the two directions (i.e., from 100x100 to 101x101, Fig. 2.9(b)). The structure of the lists is provided below:

$$ROI = PWS[\,i, i+1 : j, j+1\,] => Loc = [x, y] = [j, i] \quad (2)$$

$$IPL_n = [ID, Loc] \text{ (For 2+ unique labels)} \quad (3)$$

$$EPL_n = [ID, Loc] \text{ (For 3+ unique labels)} \quad (4)$$

$$IL_n = [ID, End\ Points, Interface\ Points] \quad (5)$$

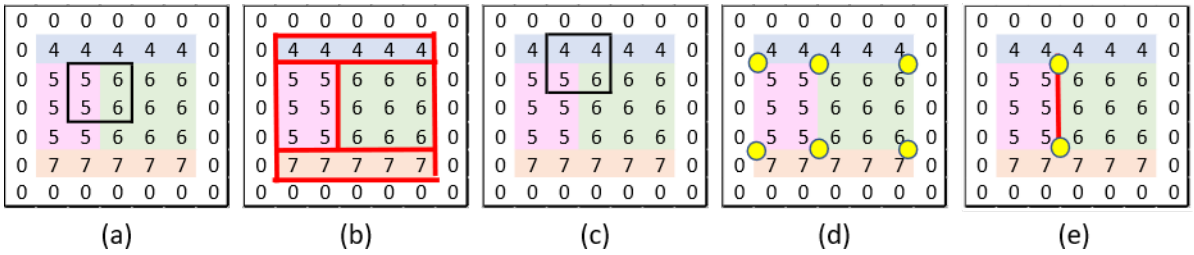


Fig. 2.9: Point detection on the padded array; (a): interface-point with 2 unique labels; (b): All detectable interface-points; (c): interface-point and end-point with 3 unique labels; (d): All end-points; (e): interface with ID= [5,6] (gridline).

At this stage, all nodes that define the geometry of the interfaces have been identified. However, their order is still unknown. Their sequence is calculated by applying the border-following algorithm developed by (Suzuki and Abe, 1985). The algorithm is applied to a new 2D-array for each unique ID with all its points marked (Fig. 2.10(b)). Moreover, only the parent-contours are stored (outer perimeter of a closed object), using the algorithm's hierarchy. As mentioned previously, the algorithm returns the perimeter that contains duplicate points if it is an open shape (Fig. 2.10(c)). Additionally, it does not return the last point of closed shapes and thus, complicates the determination of closed objects. The aforementioned are addressed by the following framework:

1. Create an array of zeros where ones mark the examined interface.
2. Apply the border following algorithm on the new array and extract the outer-contours. In rare cases, multiple contours may be extracted with the same interface ID (i.e., when the mortar label is in contact with a specific block at two separated locations).
   - 2.1. Scan each contour detected to find if it contains any end-points of the same interface. Each end-point is only considered once.
     - 2.1.1. If less than two end-points are detected, consider the object a closed shape.
     - 2.1.2. If two are detected, store only the first range between the end-points.
     - 2.1.3. If three or more are detected, find the first location of every end-point and store all ranges between end-points separately. Duplicate end-points are not considered.
   - 2.2. If the contour is considered a closed shape, and the first and last points are not equal, append the first point to the end of the xy-array.
   - 2.3. Finally, store each range separately to the contour-list (CL), including the interface ID.

The structure of the contour-list is the following:

$$CL_n = [ID, End\text{-}Points, xy\text{-}Array] \quad (6)$$

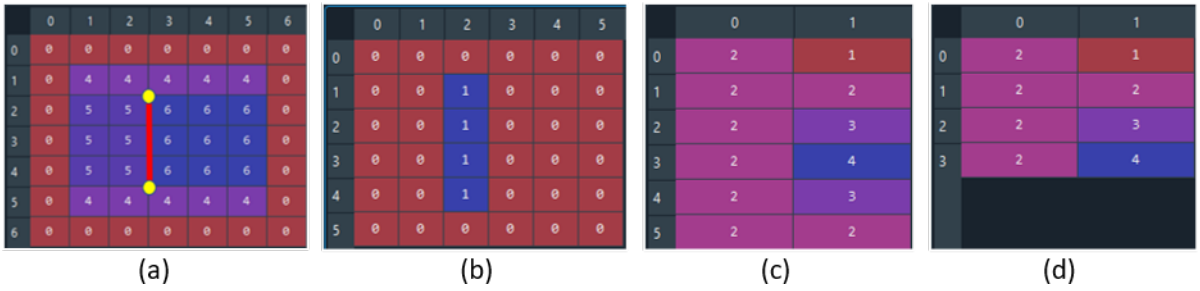


Fig. 2.10: Ordering of interface points; (a): Marked interface with ID= [5,6], end-point #1= [2,1], and end-point #2= [2,4]; (b): New array of examined interface (counting gridlines); (c): Border-following output (perimeter); (d): Modified output (polyline).

2.3.2. Contour Generalisation

Contour generalisation is the process of reducing the number of vertices that define a contour. This process will dramatically reduce the time-cost of the structural analysis. Additionally, it will provide general corrections to the shape by smoothing the interface. It is also the reason why the border-following algorithm was not used to extract the shape of each block, but was used instead on each interface. Using the border-following algorithm on the blocks, the generalisation of contours would produce multiple generalisations for the same interface (since an interface is common between two segmentations). Moreover, a line generalisation algorithm is developed, based on the "*Ramer–Douglas–Peucker*" algorithm (Ramer, 1972; Douglas and Peucker, 1973), with conditions specialised to the needs of the research.

Considering an original contour $C = [p_0, \dots, p_n]$, with n subset of vertices p, a generalised contour can be defined as $C' = [\dots, p'_{k-1}, p'_k, \dots] \subseteq C$, with the minimum number of elements that satisfies the condition $f(S_k) \leq t$. Each sequential pair of vertices of C', divides C into sections $S_k = \{p'_{k-1}, \dots, p_i, \dots, p'_k\} \in C$. The examined vertex $p_i \in S_k$ is tested with regards to the straight-line segment $\overline{p'_{k-1}p'_k}$. If the condition is not satisfied, the vertex p_i with the highest vertical distance (vd), is stored in C'. The (simplified) original condition in (Ramer, 1972), is the following:

$$f(S_k) = \max\big(vd(S_k)\big) \leq t \tag{7}$$

where $vd(S_k)$ is the vertical distance of all elements in S_k and the line segment $\overline{p'_{k-1}p'_k}$, and t is the constant-threshold value. Compared to the original, the following changes were implemented:

Transformation: Before the first iteration and only if the contour is closed (i.e. $p_0 = p_n$), the contour is reformed to start/end from the point $p_i \in C$, with the largest distance from the initial point p_0 (similar to Eq. (8)).

Condition #1: For the first iteration and only if the contour is closed (i.e., $p_0 = p_n$), add the point $p_i \in S_k$ with the largest distance from the initial point p_0, to the generalised contour. Where p_i is defined as:

$$f(i) = f(S_k) = \max\big(ld(S_k)\big) = \max\left(ld1(S_k), ld2(S_k)\right) \tag{8}$$

Condition #2: If the previous condition did not activate, then the second condition is considered. Add the point $p_i \in S_k$ with the largest distance from either the first or last point of the line segment, if it satisfies either: The vertical and length, or the horizontal and length thresholds. Using the minimum of the horizontal and length values of the pair. Where p_i is defined as:

$$f(i) = f(S_k) = \max\big(ld(S_k)\big) = \max\left(ld1(S_k), ld2(S_k)\right) \tag{9}$$

Where the condition is disregarded if: $vd(i) \leq t_v \; or \; \min\big(ld1(i), ld2(i)\big) \leq t_l$ (10)

and: $\min\big(hd1(i), hd2(i)\big) \leq t_h \; or \; \min\big(ld1(i), ld2(i)\big) \leq t_l$ (11)

Condition #3: If the previous condition did not activate, then the third condition is considered. Add the point $p_i \in S_k$ with the largest vertical distance from the line segment, if it satisfies both: the vertical and length thresholds. Using the minimum of the length values of the pair. Where p_i is defined as:

$$f(i) = f(S_k) = \max(vd(S_k)) \quad (12)$$

Where the condition is disregarded if: $vd(i) \leq t_v \; or \; \min\big(ld1(i), ld2(i)\big) \leq t_l$ (13)

Threshold: The threshold values are: Vertical (t_v), Horizontal (t_h), and Length (t_l). If the value of any threshold is below one, then it is considered the ratio of the threshold over the length of the line segment. Else, if it is above one, it is the actual threshold.

$$\text{If } t < 1\text{: } t = t \cdot length\big(\overline{p'_{k-1}p'_k}\big)\text{; else: } t = t \quad (14)$$

Where f(i) is a simplification of: $f(i) = f\big([p'_{k-1}, p_i, p'_k]\big)$. Each p_i point considered creates two horizontal (hd) and two length (ld) values (Fig. 2.11(a)), but only the minimum of each pair is tested (Eq. (10), Eq. (11), Eq. (13)). The "*Transformation"* ensures that the first point of a closed shape is essential to describe the general shape (i.e. Fig. 2.12(a1) as opposed to Fig. 2.12(b1)). While "*Condition #2"* (Eq. (11)), ensures that a vertex with a vertical point outside the range of the line segment and small vertical distance, is considered with the horizontal threshold instead (Fig. 2.11(a) & Fig. 2.11(c)). Additionally, the length threshold (Eq. (10), Eq. (11), Eq. (13)) ensures that both line segments created have adequate length. Modifying the threshold values (t_v, t_h, t_l) allows to control the level of detail of the final output by reducing the number of vertices accordingly. Lastly, the threshold ratio for values below one, ensures that the preferred accuracy is preserved by dynamically adjusting the applied threshold (Eq. (14)). Every identified polyline is stored in the line-list (LL). The structure of the line-list is provided below:

$$LL_n = [ID, Original\ Contour, Generalised\ Polyline] \quad (15)$$

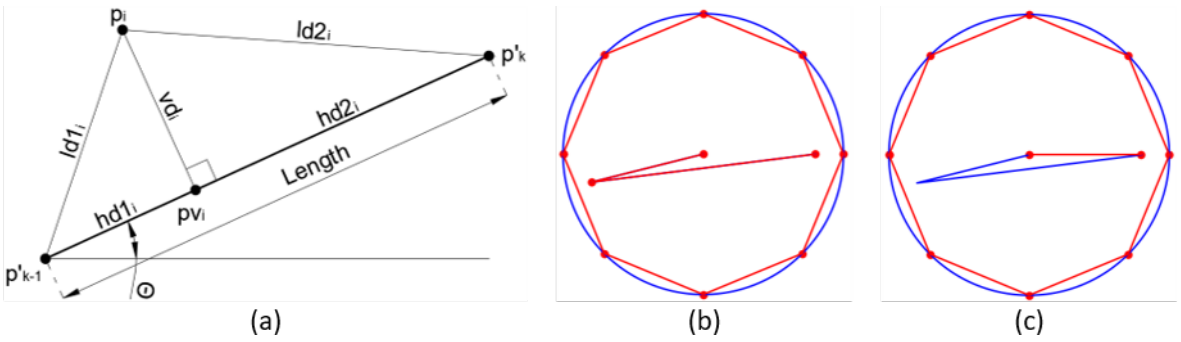


Fig. 2.11: (a): Variables of proposed algorithm; (b) Proposed algorithm (with: "th"); (c): Original algorithm (without: "th").

The resulting generalisations of the original "*Ramer–Douglas–Peucker"* (RDP) and the proposed algorithm (PROP) are compared at two levels of detail (Fig. 2.12). The level of detail is measured in the number of line segments (Lines) of the total output. Group [a] demonstrates the original algorithm using a static threshold (RDP), group [b] the proposed algorithm using static threshold (PROP-S), and [c] the proposed algorithm using a dynamic threshold (PROP-D). The results displayed below include only the geometry with mortar, since without mortar, almost every main vertex is pre-determined by the end-points between the interfaces. Regarding the results, the RDP algorithm detects a false point as initial (p'_0) on top of the closed contour in (i.e. [a1]), which is eliminated by the proposed algorithm using the "*Transformation*" (i.e. [b1, c1]). When the dynamic threshold with high ratio values is used (not shown below), the proposed algorithm may omit a second vertex to describe the small curvature near the corners better. Additionally, if a very small threshold-ratio is used (Eq. (14)), it may create an excessive number of vertices near the edges, where the threshold becomes smaller due to limited distance. Finally, all cases could be used for the development of the geometry for numerical analysis

except [a2, b2], where it is possible that their highly uneven and irregular interface could cause overestimation of collapse loads due to local hinging phenomena.

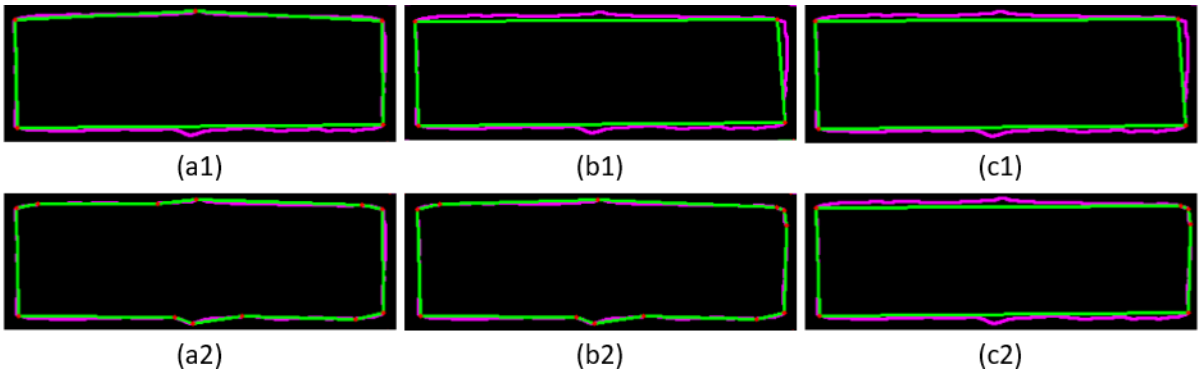


Fig. 2.12: Comparison of generalisation (Purple: Contour, Green: Polyline); (a1): RDP (Lines= 443); (b1): PROP-S (Lines= 438); (c1): PROP-D (Lines= 431); (a2): RDP (Lines= 1049); (b2): PROP-S (Lines= 1048); (c2): PROP-D (Lines= 1075).

2.3.3. Geometric Adjustments

An issue arising from segmenting the masonry imagery via a watershed-based transform is that the location where two segmentations meet is not placed at the centre of gravity of the mortar area of the binarised image (Fig. 2.13(a1)). This is because the segmentation is growing further from the marker provided. Thus, any pixel located in the bright section will be labelled from the closest marker, creating a triangular shape (Fig. 2.13(a2)). This issue is amended partially when using the distance transform of the raster image as a source for the segmentation (Fig. 2.13(b)). Doing so creates a boundary located in the middle of the bright section, forcing the assignment of half the mortar thickness to each block. Another solution suggested, only when the final model includes mortar, is to use a large erosion value on the GMM and then dilation to restore its average size, covering the small cavities near the end-points (Fig. 2.13(c)). However, that will also introduce a small curvature to the edges of the block. Nonetheless, the erosion/dilation solution is often unnecessary since the generalisation of the contour will correct this, assuming that the threshold is not excessively low. However, it is required to have implemented the transformation of the previous section; otherwise, the top-most point will be included, since it is the first point detected by the border following algorithm (Fig. 2.12(a1)).

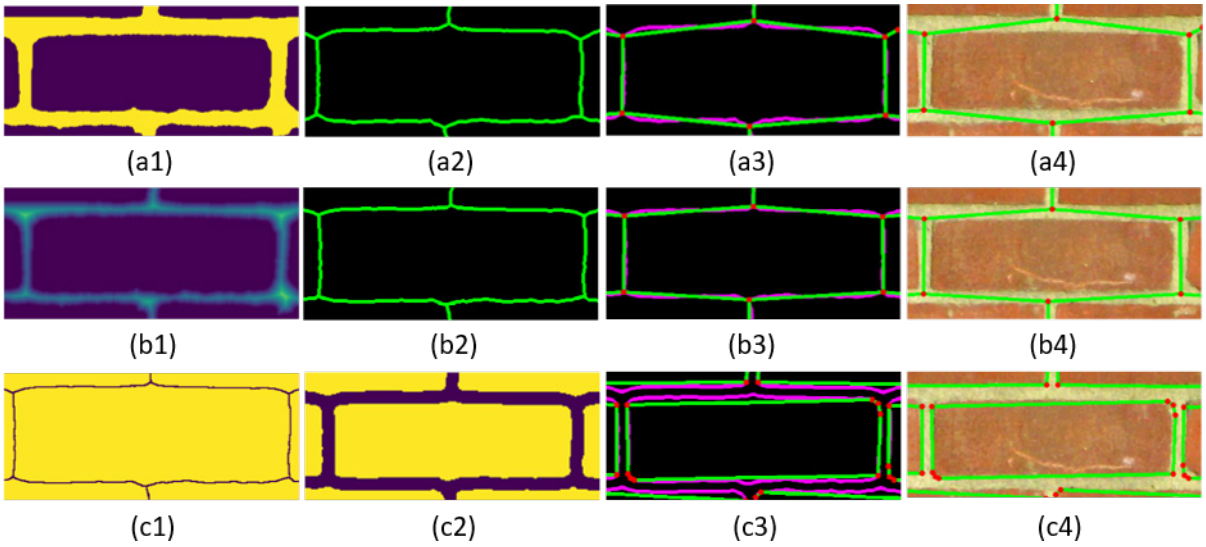


Fig. 2.13: Simple corrections to segmentation: (a): Contours and polylines using the original rasterised image; (b): Contours and polylines using the distance transform as a source; (c): Correcting generated mortar-mask using erosion/dilation.

An additional processing step is introduced that adjusts only the affected vertices when the mortar is not modelled (Fig. 2.15(a)). The issue develops where two segmentations are connected, and two of the three interface lines form a near ~180° angle. Thus, the algorithm must target end-points that have three ID values (three unique values in the 2x2 ROI), and all are larger than zero. The location where the ID of end-points have zero or negative values are not affected since they do not have a second marker. Any end-point that has precisely three unique labels (i.e., end-points connected to three polylines), and all its values are larger than zero (i.e., not mortar, damage, or background) is considered a possible candidate. Knowing the location and the unique ID, makes possible to identify the three polylines that are connected to the end-point. For every polyline detected, the line-segment is formed between the end-point and the previous on the polyline's xy-array. That may be either the first two or the last two points. If the angular difference between any combination of two line-segments is ~180°, then the end-point is relocated so that the angular difference between the two lines is exactly 180°. However, the global angle of the remaining line must remain the same to retain its shape (Fig. 2.14: $\overline{BP}$). The extension of the line that does not form ~180° splits the geometry into two triangles (Fig. 2.14: $\overline{BP'}$), where the law of sines can be used to determine its length. Then, geometric functions are used to calculate the coordinates of the adjusted end-point (Fig. 2.14: P'). A threshold value (t_a) is used to determine the tolerance of the angular difference. Furthermore, three iterations of the algorithm are required to correct all vertices. The formal description is provided below:

1. Create a copy of the "End-Point-List" and "Line-List" (i.e., EPL2 and LL2), to avoid modification of the original lists. Important: This may be required later if a closed-block is incorrectly defined.
2. Create the first iterative loop to repeat the process three times.
3. Create the second iterative loop to scan through the copied end-Point-list" (EPL2).
4. If the ID of the end-point contains exactly three labels, and all are higher than zero, consider the end-point (P) a possible candidate (i.e., $ID = [Lbl_1, Lbl_2, Lbl_3]$; and: $min(ID) > 0$).
   4.1. Identify the location index of the three polylines in the line-list (LL), by verifying that the first or last point of the polyline is equal to the location of the end-point saved on EPL2. The ID label of the end-point may also be used to locate the index.
   4.2. Create the line-segments by taking either the first two, or the last two points of the generalised line, based on the location of the end-point (i.e. if: $xy_P = xy_0$, then: $Line = [xy_0, xy_1]$; else if: $xy_P = xy_n$, then: $Line = [xy_n, xy_{n-1}]$).
   4.3. Calculate the angular difference between all combinations of the three line-segments to identify if two of the line segments form ~180° angle (i.e., $Ad_1 = \angle APC$; $Ad_2 = \angle CPB$; $Ad_3 = \angle BPA$). If the optional condition is used (Eq. (17)), calculate the global angle of the two line-segments that form ~180°, excluding the end-point (i.e., $A_1 = \Theta_{AC}$).
   4.4. If the conditions are satisfied (Eq. (16), Eq. (17), Eq. (18)), relocate the end-point (P), such that the lines that form ~180° become parallel (i.e. $\overline{AP'} \parallel \overline{P'C}$, with $P' \in \overline{AC}$), while retaining the global angle of the remaining line (i.e. $\Theta_{BP} = \Theta_{BP'}$).
   4.5. Update the value of the modified end-point to every list used (i.e., EPL2, LL2)
5. Repeat the process for all end-points during three iterations.

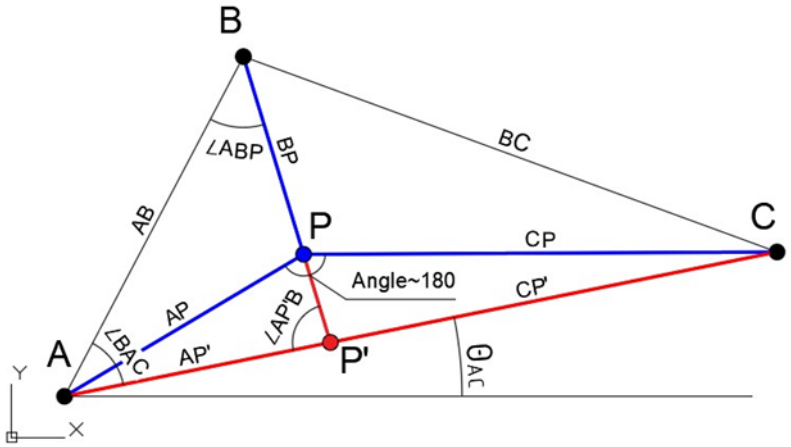


Fig. 2.14: Variables used in the geometric corrections (P: End-point, Blue: Old lines, Red: Adjusted lines).

The main-condition is to verify if any combination of angular difference is ~180° (Eq. (16)). An optional-condition to automatically avoid rubble or arch-lines is to target cases where both ~180° and ~90° angular-difference are detected (Eq. (17)). An alternative optional-condition, that has the same purpose, is to verify that the global-angle of the adjusted line that forms near ~180° local-angle, is either: ~0° or multiples of ~90°; before applying the geometric corrections (Eq. (18)). All proposed conditions of the formal description (Sub-list: [4.4]), are provided below:

Condition #1 (Main): $[180° - t_a \leq Ad_i \leq 180° + t_a]$ (16)

Condition #2 (Optional): $[90° - t_a \leq Ad_j \leq 90° + t_a]; i \neq j;$ (17)

Condition #3 (Optional): $[A_t - t_a \leq A_i \leq A_t + t_a]; A_t = [0° \vee 90° \vee \ldots \vee 360°];$ (18)

Applying the geometric corrections to the interface between structural units allows for a more representative visualisation of masonry characteristics (Fig. 2.15). Regarding the examples demonstrated below, only the main condition was used (Eq. (16)). The arch-lines (Fig. 2.15(e)) were automatically excluded from the corrections since one of the three labels is zero (i.e., background). However, if the arch would contain more layers, an additional condition would be necessary to avoid the geometric adjustments on the inner interfaces (i.e. Eq. (17), Eq. (18)).

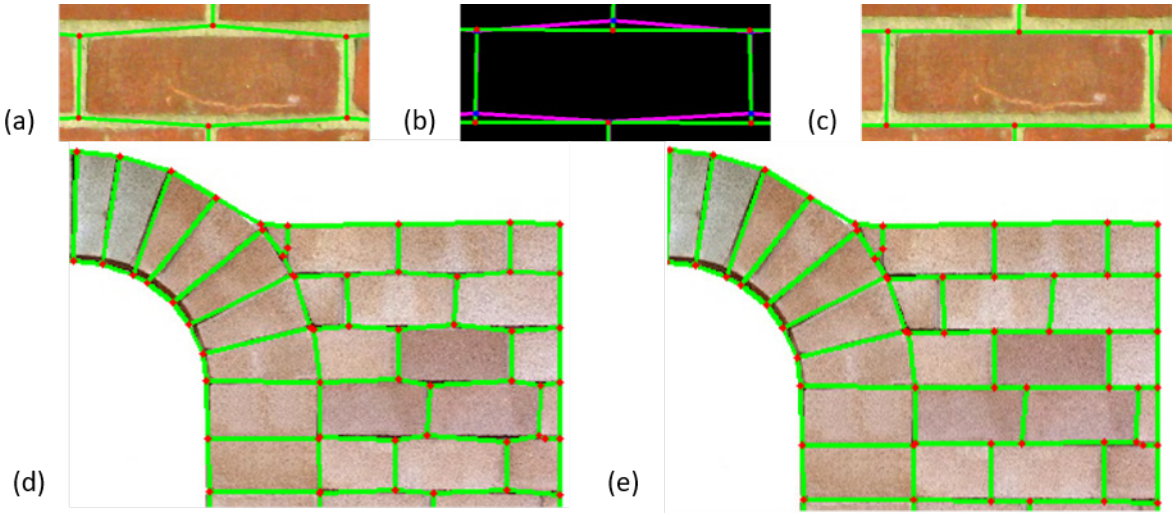


Fig. 2.15: Correcting geometrical inaccuracies ($t_a = 20°$): (a): Initial generalised lines; (b): Original and modified lines comparison; (c): Adjusted generalised lines; (d): Original lines on arched-door; (e): Adjusted lines on arched-door.

2.3.4. Producing Closed-Shapes

Specific numerical analysis software require closed-objects to define a shape (i.e., ABAQUS, 3DEC, LS-DIANA, etc.). For that purpose, it is necessary to generate the closed-block from the open-polylines. An interface ID contains two labels that are equal to the two objects that are in contact, where the label is the value of any segmentation. Thus, all polylines are compared, and those that contain a specific label are assigned to the block of the same value (i.e., Interface $ID = [2, 3]$ is assigned to blocks #2 and #3). The collection of the polylines is added to a temporary-block-list (TBL) with its label.

The polylines are then combined by comparing the initial and final vertices of each entry in the collection (using the first entry as a temporary combined-polyline). The process is repeated until no connection is found with the temporary combined-polyline. Every entry used is removed from the polyline-collection to ensure that it is not used repetitively. If the combined-polyline is closed (i.e., equal first and last vertices), it is stored to the final block-list (BL) with its original label (Eq. (1)) to retain its type. Alternatively, if no connection is found and the combined polyline remains open, it is rejected. Furthermore, the process must repeat again for the same collection if there are remaining entries, which is possible when multiple segmentations have the same label. However, this will not be the case if the "*Segmentation Corrections*" have been applied. The original contour (the line with all points included), can also be stored to allow the use of the preferred accuracy for CAD model generation. Negative labels (damage and mortar) are excluded from this process, but the zero label is included to describe the overall perimeter. The structure of the block-lists is provided below:

$$TBL = [Label, Contour\ Collection, Polyline\ Collection] \quad (19)$$

$$\text{If } Label \geq 0\text{: } BL_n = [Old\ Label, Closed\ Contour, Closed\ Polyline] \quad (20)$$

Moreover, if the generalisation and generation of closed shapes for the damage is required (i.e. for CAD design), each negatively-labelled segmentation must be assigned a unique label during the "*Segmentation Corrections*" (Eq. (1)), which aims to avoid conflicts with the contour definition.

Although a rare occurrence, a block label may fail to regenerate or provide the correct shape when an external mask is supplied. This is because interfaces of equal ID value are marked on the same array to apply the border-following algorithm. When 1-pixel segmentation separates two interfaces of equal ID, they are drawn connected since the method proposed considers the gridlines rather than the pixels. In this case, the algorithm may return incorrect ranges by omitting an end-point that is blocked by connected pixels (case #1) or return a single perimeter of the combined shape instead of two separated interfaces (case #2). The aforementioned refers only to individual outputs of the border-following algorithm that contain three or more end-points and is considered an open-shape. Additionally, it affects only watershed-segmentations that were modified using imported-masks (i.e., mortar, damage).

The simplest solution to this is to increase the size of the watershed segmentation two-fold just before the point-detection algorithm, which will effectively increase segmentations of 1-pixel thickness to 2-pixel. In this way, the unification of individual sections that are separated by 1-pixel is avoided. Alternatively, a programmable solution is also possible by forcing the algorithm to follow only the outer-perimeter of an affected interface. This is accomplished by applying the border-following algorithm on the gridlines of a segmentation instead, which will provide the outer perimeter of the affected block. The perimeter can be divided into interfaces by comparing the ID values of each point it contains. All contours/blocks of the affected cases must be removed and replaced with the adjusted items. However, the contour-generalisation and geometric adjustments require repetition for every affected item and interfaces/blocks in contact with the affected case, since modified interfaces may be common to two different segmentations. The programmable solution should be applied after combining interfaces into closed-blocks to ensure that all detection methods were used. Affected contours/blocks can be identified by:

1) *Detection method #1*: When an individual and open contour-output, of the border-following algorithm, contains duplicate inner-points.
2) *Detection method #2*: When the combined closed-shape includes duplicate inner-points (excluding the first since it is a closed shape).
3) *Detection method #3*: When failing to combine the interfaces of a segmentation into a closed-shape (i.e., unequal first and last vertices).

2.3.5. Data Scaling

Before any further adjustments are made, the block and line list (BL and LL) are scaled to the preferred size by application of a scale factor to xy-coordinates of each element (Eq. (21)).

$$\begin{aligned} x' &= x \times scale \\ y' &= (ymax - y) \times scale \\ ymax &= \max(y) \end{aligned} \tag{21}$$

The final geometry may include small objects that are inappropriate for use in CAD geometry or numerical model generation. Furthermore, the application of the line-generalisation algorithm may cause the generation of blocks with zero area due to inadequate space between essential vertices. Thus, the area verification of each object is required to provide proper input for the numerical analysis. The area of a closed polygon can be calculated using the Surveyor's Formula provided below (Eq. (22):

$$Area = \frac{1}{2} \times \left| \sum_{i=1}^{n-1} (x_i \times y_{i+1}) + (x_n \times y_1) - \sum_{i=1}^{n-1} (x_{i+1} \times y_i) - (x_1 \times y_n) \right| \tag{22}$$

Acquiring the area of either the scaled closed-contour or the closed-polyline allows the removal of small or zero-area objects from the block-list (BL) (Fig. 2.16). This can extend to lines by removing entries from the line-list (LL), that do not have at least one ID label equal to the labels of the remaining blocks (i.e., if both labels 10 and 11 do not exist in the adjusted block-list, then the polyline with $ID = [10,11]$ is removed). Furthermore, the samples below exclude damage from the "*point detection*" section to avoid common issues (pg. 28). However, the damage will be included during the numerical analysis.

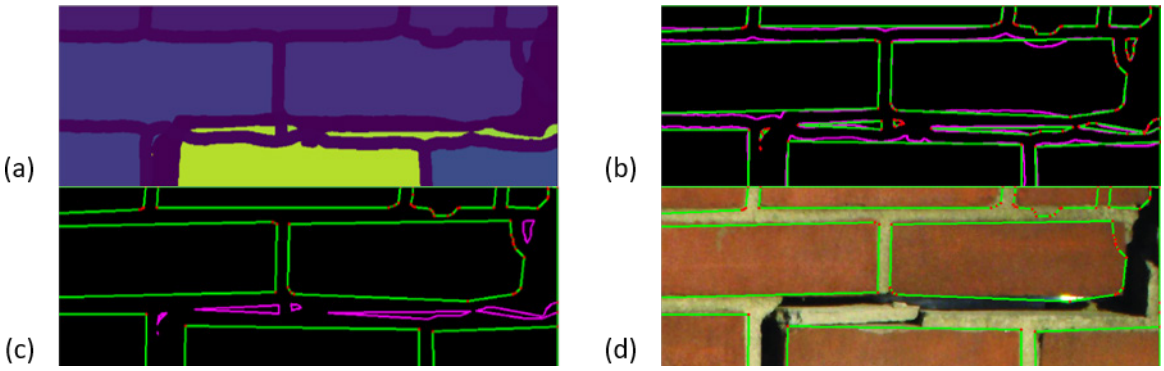


Fig. 2.16: Removing small objects; (a): Initial segmentation; (b): Original contours and generalised lines; (c): Removed small objects (in purple); (d): Remaining objects on source image.

2.4. Numerical Model Generation

The last part of the proposed framework is the numerical model generation, which includes mortar and damage depending on the user's preferences (Fig. 2.17: Output). The proposed framework can be used to simulate masonry with the simplified-micro-modelling (or meso-scale) and detailed-micro-modelling approach. For the development of the numerical simulations, the commercial software UDEC, developed by Itasca, has been used (Itasca, 2019). The formulation of the method was

proposed initially by (Cundall, 1971) to study jointed rock, modelled as an assemblage of rigid blocks. Later this approach was extended to other engineering fields requiring a detailed study of the contact between blocks or particles such as soil and other granular materials (Ghaboussi and Barbosa, 1990). More recently, the approach was applied successfully to model historic masonry structures in which the collapse modes were typically governed by mechanisms in which the blocks' deformability plays little to no role at all.

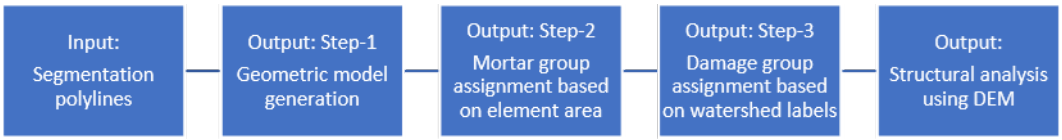


Fig. 2.17: Workflow of the numerical model generation (Output)

2.4.1. Geometric Model Generation

The methodology used to develop the polylines from the previous sections has been used here to generate the model geometry in AutoCAD or directly into a structural analysis software (Fig. 2.18). The python library used to create the AutoCAD model was "*pyautocad*". The 3D model was extruded using a standard value for the depth (Fig. 2.18(b)). The 2D mesh was created in AutoCAD using the "*hatch*" command (Fig. 2.18(c)) and includes both mortar and damage. For the models developed using the detailed micro-modelling approach, the mesh generation was made externally when the mortar was modelled. Optimally, the mesh could be produced programmatically when the geometry is aimed for UDEC, to avoid the use of AutoCAD entirely. For cases where the model was initially generated in AutoCAD, the python library "*dxfgrabber*" was used to read DXF files. Lastly, each line was imported in UDEC using FISH (programming language embedded in ITASCA software) to generate the masonry units (Fig. 2.18(d)).

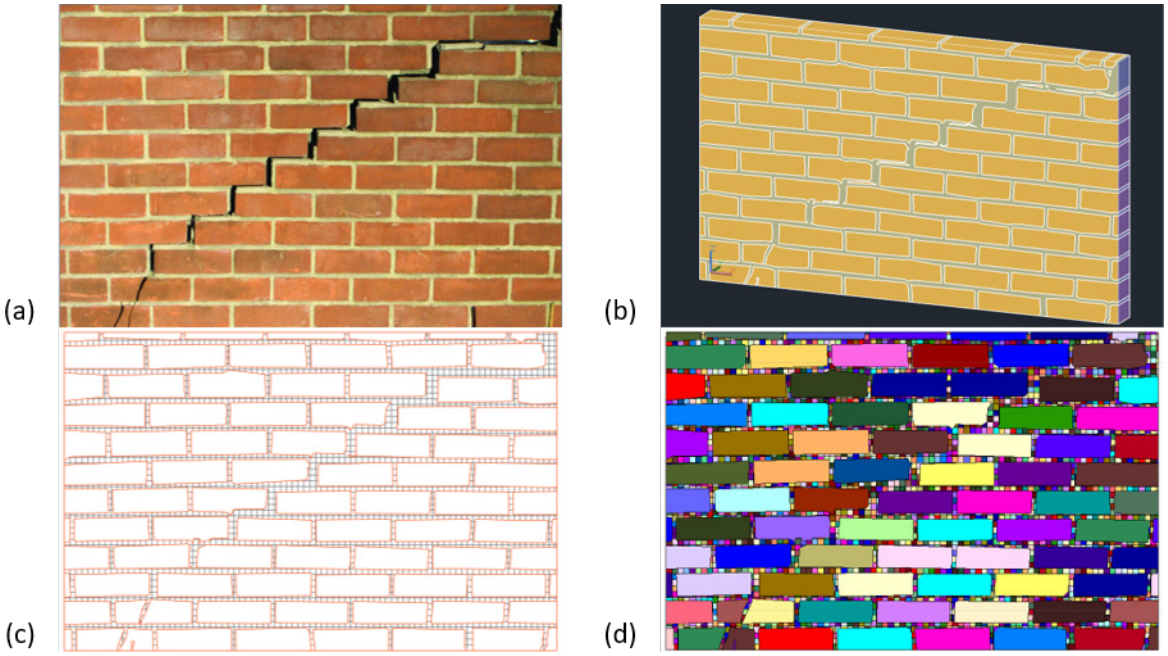


Fig. 2.18: Geometric model generation; (a): Image source; (b): AutoCAD 3D-model using blocks; (c): AutoCAD 2D-model using lines (detailed micro-modelling); (d): UDEC 2D-model using lines (detailed-micro-modelling).

2.4.2. Mortar and Damage Group Assignment

In the numerical model, the mortar was assigned by calculating the area of each element. So, if an element is smaller than a predefined value, then it is assigned to the "Mortar" group (Fig. 2.19(a)). The predefined value is equal to the square of the largest (vertical or horizontal) side of the mesh element, see (Eq. (23)).

If $Area \leq (length)^2$, then assign the element to the "*Mortar*" group (23)

Moreover, the mortar may be cracked, which needs to be reflected in the "*Mortar"* group. If this is the case, then the masked watershed segmentation (without padding) is used to extract the coordinates. Whenever a pixel with a damage label is detected, a FISH command is generated that re-assigns the "*Mortar*" element that contains the specified coordinates to the "*Damage*" group, see (Fig. 2.19(b)). It is essential to mention that the coordinates obtained require adjustment, since the interface-contours measure gridlines instead of pixels:

If $WS(i,j) = -2$ or $WS(i,j) = -3$: Extract adjusted xy coordinates (24)

$$x = (j + 0.5) \times scale \quad (25)$$

$$y = \big(ymax - (i + 0.5)\big) \times scale$$

$$ymax = \max(y)$$

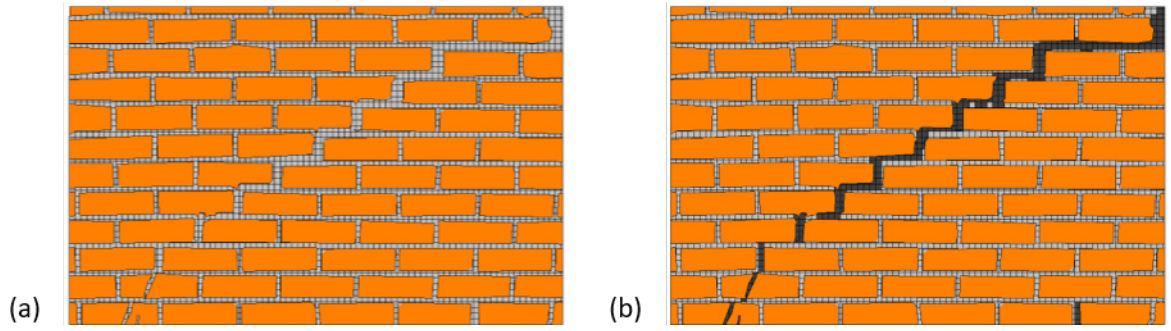


Fig. 2.19: Assigning mortar and damage (Orange: Brick, Grey: Mortar, Black: Damage); (a): Mortar comparing the element area; (b): Damage per pixel at each mortar element (71030 entries – 315s).

The method mentioned earlier used to assign the damage, requires an excessive amount of time due to the large number of pixels detected (i.e., 315s for 71,030 entries on a laptop with i7-9750h, Fig. 2.19(b)). For that reason, the accuracy of pixel extraction is limited to a preferred area (i.e., $Acc = 5px$). Thus, rejecting new entries if they are in proximity to an already extracted location, which reduces dramatically the computational time required (i.e., 18s for 4,386 entries, Fig. 2.20(a)). The method used to verify the distance is by creating a test-array of equal size to the original image where all coordinates extracted are marked by the preferred area of double the size of the accuracy (i.e., $TArray[i-5:i+5, j-5:j+5] = 0$). Creating multiple test-arrays for each damage state allows to verify and assign multiple groups (Fig. 2.20(b)), which in turn can be used to assign different material and joint characteristics or to remove completely specified damage-groups. The assignment accuracy of multiple groups depends highly on the mesh size, the precision of the imported damage/mortar masks, and the applied order of the damage to the model (i.e., Fig. 2.20(b)), where the mortar-damage is given priority). For the tested accuracy, multiple damage assignments are not recommended since the mesh size is not sufficiently small, and due to inadequate accuracy of the imported (not generated) masks. The second method proposed is based on both; a range value targeting the centroid (Eq. (26) & Eq. (27)) and the block that contains the specified coordinates (Eq. (25)), of mortar-elements only. The range of values used, in the numerical model, are provided below:

$$x1 = (j + 0.5 - Acc) \times scale \quad (26)$$

$$x2 = (j + 0.5 + Acc) \times scale$$

$$y1 = \big(ymax - (i + 0.5 - Acc)\big) \times scale \quad (27)$$

$$y2 = (ymax - (i + 0.5 + Acc)) \times scale$$

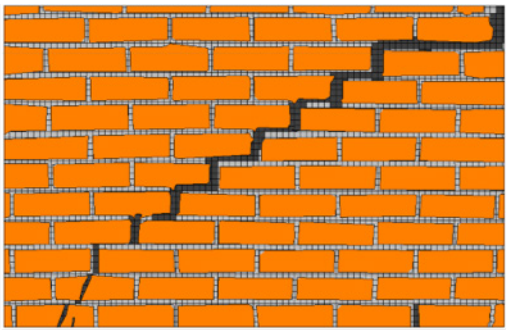
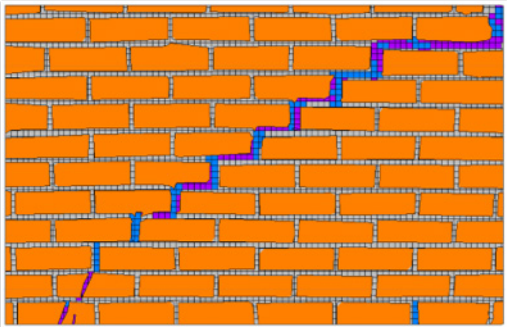
(a)  (b) 

Fig. 2.20: Assigning damage using range (5-pixel range, Blue: Mortar damage, Purple: Block damage); (a): General damage (4386 entries, 18s); (b): Multiple assignments giving priority to mortar-damage (4628 entries, 18s).

2.5. Numerical Analysis of existing masonry walls

The geometry obtained using the proposed algorithm is compared with the idealised model to verify the proposed methodology's potential to be used in automated model generation. This section does not aim to predict the behaviour of a real structure. Instead, it tests the structure for similar behaviour, assuming a similar geometry is provided, while retaining the mechanical properties equal between all models. The numerical analysis also assumes an accurate model calibration, by following the typical modelling procedure and by using representative material characteristics, using information acquired from the literature. Moreover, the following generalisation values are suggested for the general definition of the geometric model aimed for numerical analysis:

$$Vertical\ Ratio = Horizontal\ Ratio = 0.1 \quad (28)$$

$$Min.Length = Mortar\ Thickness + 1 \text{ (in pixel size)} \quad (29)$$

The first model selected as a case study is the masonry arched-door (Fig. 2.21). The specific geometry was chosen as it introduces additional complexity to the analysis by including an arch and opening to the model. In this case, the mortar is represented as a zero-thickness interface since the simplified micro-modelling approach is typical for the numerical analysis of masonry structures. The vertical and horizontal generalisation-ratio used, to acquire the geometry, is equal to 0.1 with a 3-pixel minimum distance. Regarding the analysis, only deformable blocks are considered (although failure is expected to occur in the mortar joints rather than in the masonry units). The source of the action affecting the structure is a horizontal velocity applied on a block pattern, see (Fig. 2.21(c) & Fig. 2.21(d)).

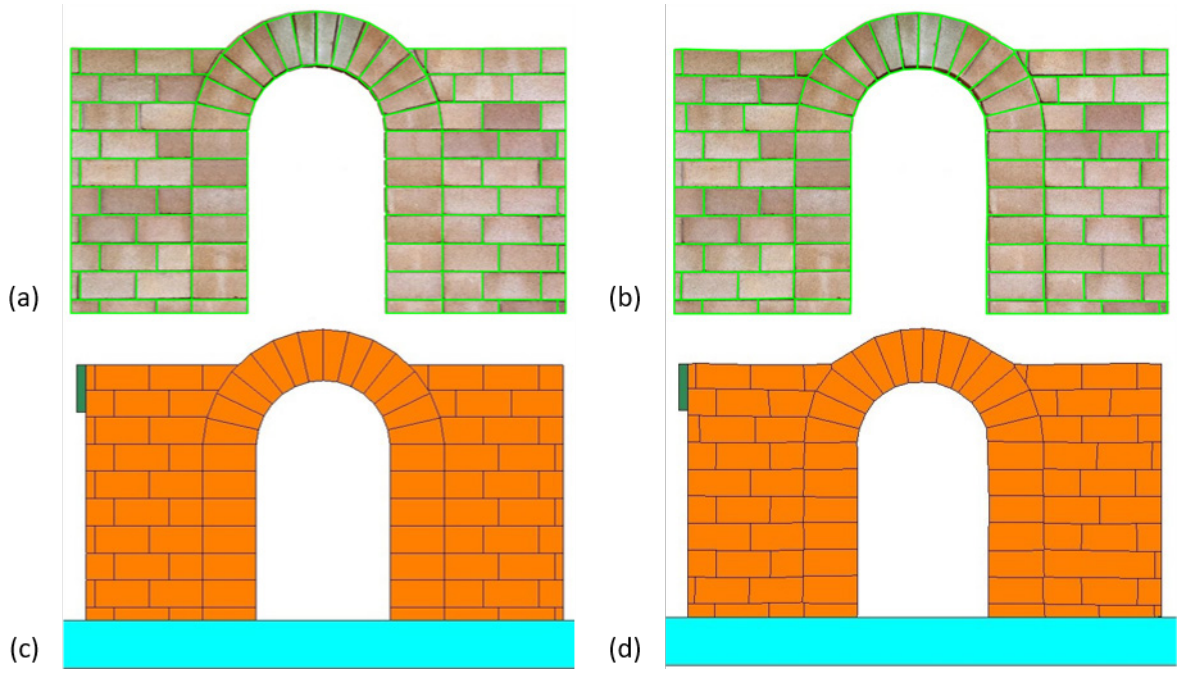


Fig. 2.21: Model geometry of Arched-Door; (a): Idealised model; (b): Generated model; (c): Idealised UDEC groups; (d): Generated UDEC groups.

Table 2.1: Macro properties of the brick elements- Block properties: Sandstone (Karagianni et al*., 2010).*

Density (kg/m^3)	Young's modulus (N/m^2)	Poisson's ratio
2350	$26364x10^6$	0.2

Table 2.2: Joint properties of the zero-thickness interfaces - Join contact properties of mortar (Sarhosis and Lemos, 2018).

Normal stiffness (N/m^3)	Shear stiffness (N/m^3)	Friction (deg)	Cohesive strength (N/m^2)	Tensile strength (N/m^2)	Dilation (deg)
4×10^{11}	2×10^{11}	38	0.6×10^6	0.6×10^6	4

The maximum loading of the idealised and generated model is 1,240 kN and 1,360 kN respectively (+9.7 %, Fig. 2.22(a)), assuming a 500 mm depth. The maximum-load difference is possibly caused by the interlocking/hinging mechanism between the top-left block-row and the arch in the generated model, not present in the idealised case. A more accurate binarised-source would provide overall better results. However, the failure-pattern at 10mm displacement is similar, with only minor differences (Fig. 2.22(b) & Fig. 2.22(c)), although the geometry between the two models is not identical (i.e. merged blocks, additional edges, etc.).

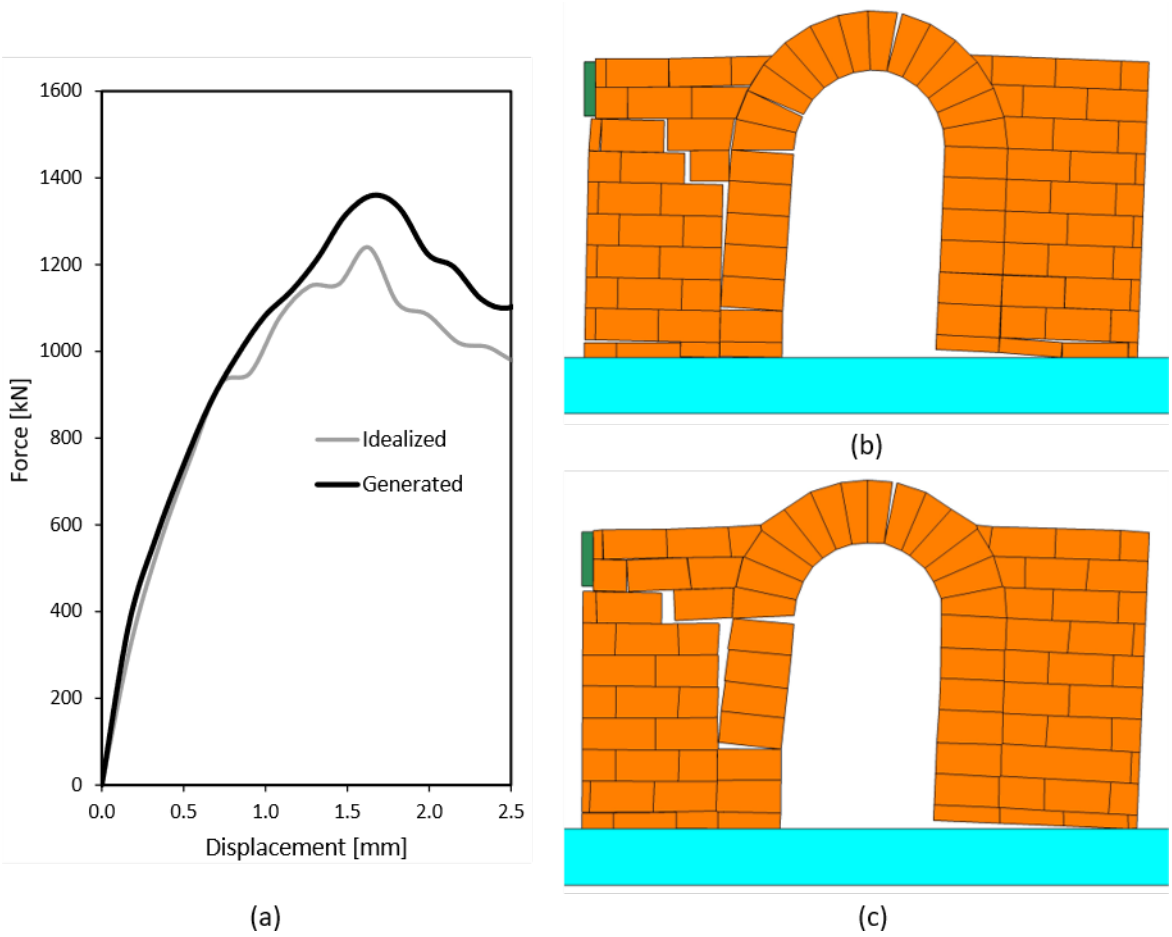


Fig. 2.22: (a): Force-Displacement graph of Arched-Door under horizontal loading; (b): Idealised model after 10mm horizontal displacement; (c): Generated model after 10mm horizontal displacement

The second model for comparison is the damaged brick-wall, including the physical modelling of mortar (Fig. 2.23(a) & Fig. 2.23(b)). The pixels corresponding to damaged areas, detected from the image-processing of the original image, were directly inserted on the idealised model. This allows the comparison of the extracted geometry without considering the success rate of the defect detection. Furthermore, the damaged locations were removed to imitate the separation of material on the original image. The vertical and horizontal generalisation-ratio used, to simplify the geometry, is equal to 0.1 with a 5-pixel minimum distance. The analysis considers only deformable blocks.

Moreover, the mechanical properties of the joints are the same as in the previous case. However, the brick/mortar units have different properties, which are provided below. In this particular case, the mortar is consisted of individual triangles of 20 mm sides to reduce the computational cost for the analysis. More importantly, the triangular mesh was selected since it permits diagonal separation. In the future, segmentation of the mortar can be done using Voronoi elements or alternative shapes. The source of the action affecting the structure is a horizontal velocity applied on a block pattern, see (Fig. 2.23(c) & Fig. 2.23(d)).

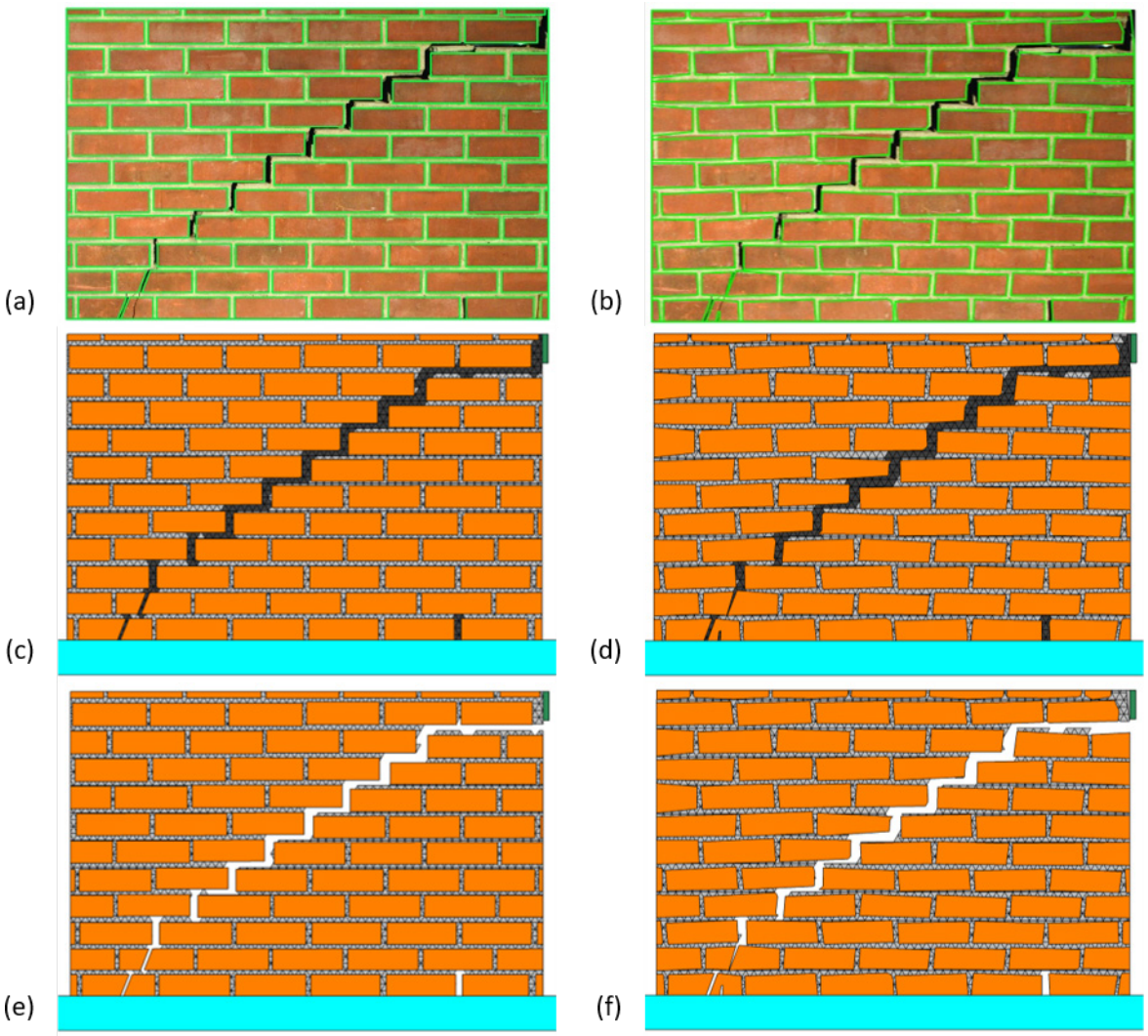


Fig. 2.23: Model geometry of Damaged-Wall; (a): Idealised model; (b): Generated model; (c): Idealised UDEC-groups; (d): Generated UDEC-groups; (e): Removed material of Idealised model; (f): Removed material of generated model.

Table 2.3: Macro properties of the brick elements (Sarhosis and Lemos, 2018).

Brick elements			Mortar elements		
Density (kg/m^3)	Young's modulus (N/m^2)	Poisson's ratio	Density (kg/m^3)	Young's modulus (N/m^2)	Poisson's ratio
1900	$19700x10^6$	0.2	1200	$2974x10^6$	0.2

The maximum loading of the idealised and generated model is 23.15 kN and 22.45 kN respectively (-3.0 %, Fig. 2.24(a)), assuming a 102.5 mm depth. Moreover, the failure-pattern at 10mm displacement is similar, following the pre-existing damage and causing separation at the top-right and bottom-left corners (Fig. 2.24(b) & Fig. 2.24(c)). However, the shape of the mesh has not been optimised for numerical analysis and may interfere with the results due to local hinging phenomena. The investigation of the mesh type and size is outside the scope of this research (investigated partially in chapter 4). Nonetheless, the expected behaviour is observed in both cases (i.e., separation at damaged locations).

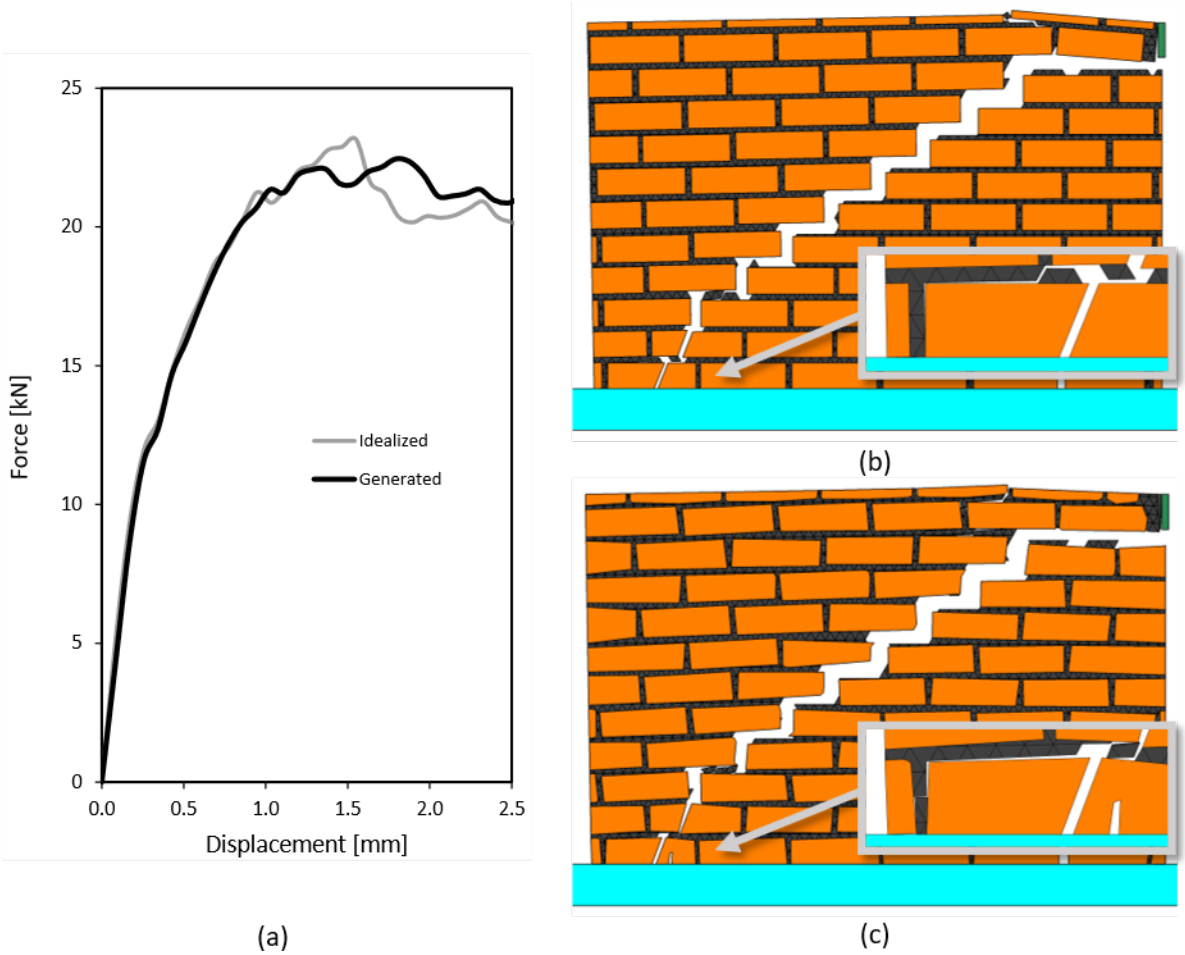


Fig. 2.24: (a): Force-Displacement graph of Damaged-Wall under horizontal loading; (b): Idealised model after 10mm horizontal displacement; (c): Generated model after 10mm horizontal displacement.

2.6. Conclusions

The proposed methodology proves to be a time-efficient and robust system of acquiring the geometric shape of a masonry structure for numerical analysis, especially in the case of complex structures where a significant effort is required to create the numerical model. Any type of masonry construction is supported, as long as an adequate source is provided (i.e., ashlar, rubble, dry-joint, mortared-joint, etc.). Another possible application of the proposed methodology is the automated assessment of numerical analysis results (by comparing the displacements of the elements with the coordinates of the objects acquired during the feature-extraction). Especially during evaluation of an inverse analysis or assessment of proposed models aimed to predict crack propagation (Tiberti and Milani, 2020b, 2020a). The algorithm may also be used to generate precise CAD designs of real structures in an efficient and timely manner. Additionally, it provides a use-case for state-of-the-art research in feature-detection and segmentation. However, a reliable method of feature-detection is required for optimal results (i.e., use semantic-segmentation instead of edge-detection/thresholding).

From the analysis of results, the efficiency of the methodology depends highly on the accuracy of the geometry extracted and the number of lines used to describe the same geometry. The generated model has the potential to provide more accurate results, assuming that it resembles the shape of the structure in more detail than the idealised geometry. Nevertheless, the user must ensure that no interlocking between the blocks is present, where it is not anticipated, due to unnecessary complexity. This is resolved by the proposed generalisation-algorithm, which reduces the number of edges of the blocks in the model. For general use, the values proposed in the "*Numerical Analysis*" section (Eq. (28)

& Eq. (29)) demonstrate adequate results, which are taking advantage of the dynamic-adjustment to automate the assignment (Eq. (14)). If necessary, a larger generalisation-ratio can be used to simplify the model further. However, the physical modelling of mortar assists on that regard since there is no direct interaction between the masonry blocks.

The main limitation of the methodology is the accuracy of the original binarised-image used to produce the watershed segmentation (resolved in chapter 3). The proposed approach is aimed to be used in combination with state-of-the-art image-processing techniques to improve the overall precision of the feature-extraction. Additionally, the methodology is limited to 2D model generation. Although, simple 3D models can be generated by assuming a standard depth value across a single plane (i.e., Fig. 2.18(b)).

Regarding the discrete element analysis, the mortar mesh has not been optimised for the current use-case. Further investigation is required to optimise the mesh type and the size of each element. A different mesh type (i.e., Voronoi, (Sarhosis and Lemos, 2018)) and equal or smaller size than the mortar-thickness may prove more efficient in highly stressed areas. However, it may also increase the computational effort required. Moreover, the joint characteristics consider a single global assignment, where the mortar-to-mortar interface is equal to the block-to-mortar (only valid for the damaged-wall example). A more accurate representation of the joints could provide a more realistic outcome. Lastly, the material, joint, and damage characteristics require further investigation and should be calibrated to experimental data for a realistic depiction of the analysis results.

Acknowledgements

This work was partially funded by European Union's Framework Programme for Research and Innovation Horizon 2020, under the H2020-Marie–Skłodowska Curie Actions-COFUND scheme (Grant Agreement ID: 754511), and from the banking foundation Compagnia di San Paolo.

3. Paper #2: Automatic image-based brick segmentation and crack detection of masonry walls using machine learning.

Dimitrios Loverdos[a], Vasilis Sarhosis[a,*]

[a] School of Civil Engineering, University of Leeds, Woodhouse Ln, Leeds LS2 9DY, United Kingdom

Abstract

This paper aims to enhance automation in brick segmentation and crack detection for masonry walls using image-based techniques and machine learning. Initially, a large dataset of hand-labelled images of different in colour, texture, and size of brickwork masonry walls has been developed. Next, various deep learning networks, including U-Net, DeepLabV3+, U-Net (SM), LinkNet (SM), and FPN (SM), were employed, and their quality was evaluated. Furthermore, the paper explores the ability to generate geometric models of masonry structures and evaluates the geometric properties of detected cracks. Additional metrics were also developed to compare the CNN-output with other image-processing algorithms. From the analysis of results, it was shown that the use of machine learning, for brick segmentation, provides better outcome than typical image-processing applications. This implementation of deep-learning for crack detection and localisation of bricks in masonry walls highlights the great potential of new technologies for documentation of masonry fabric.

Keywords: masonry, image processing, documentation, watershed, segmentation, deep learning, CNN.

***Corresponding author:** Dr. V. Sarhosis, School of Civil Engineering, University of Leeds, LS2 9JT, Leeds, UK email: v.sarhosis@leeds.ac.uk

3.1. Introduction

Masonry is one of the oldest building materials. It is composed of individual masonry units (bricks, blocks, ashlars, irregular stones, etc.) jointed together with or without mortar. Masonry structures represent the highest building stock worldwide. A large portion of masonry structures are "Listed Buildings" and form part of our "Cultural Heritage" (McKibbins *et al.*, 2006). According to UN Sustainable Development Goal 11, Target 11.4, there is a need to "repair and maintenance" rather than "demolish and rebuild" our structures. At present, when inspecting existing masonry structures, damage pathologies (such as cracking and spalling) are typically recorded using traditional techniques like visual inspection and manual surveying methods (Eaton, Edwards and Crapper, 2014)). However, traditional methods are labour intensive, subjective, and error prone (Sowden, 1990; Phares *et al.*, 2004). Additionally, for the documentation of masonry structures, the size and location of masonry units and mortar is of particular interest for the engineers and architects, since such information can be used for the development of high-fidelity computational models for their structural analysis and assessment (Lourenço, 1996, 2013; D'Altri *et al.*, 2020; Kassotakis *et al.*, 2021).

In the last ten years, advances in laser-scanning and photogrammetry have started to drastically change the building industry since such techniques are able to capture rapidly and remotely digital objects and features in images and points' cloud format. Past research demonstrated that computer-vision and image-processing can be used to create detailed digital records of masonry structures (Kassotakis and Sarhosis, 2021) using feature detection (Canny, 1986; Martin, Fowlkes and Malik, 2003; Arbeláez *et al.*, 2011; Bora, 2017) and segmentation algorithms (Beucher, S.; Meyer, 1993; Arbeláez *et al.*, 2011; Kornilov and Safonov, 2018). Applications of Image-processing can be used to quantify deformations on masonry (i.e., when coupled with the installation of simple markers on the structure on key-locations to identify their position (Bal *et al.*, 2021; Stockdale, Yuan and Milani, 2022)). Furthermore, feature-extraction has the potential to generate the "as is" numerical model of masonry for detailed analysis (Tiberti and Milani, 2019, 2020a; Kassotakis *et al.*, 2021; Loverdos *et al.*, 2021a). Those applications of image-processing offer a low-cost and reliable alternative to more traditional methods. Although, feature detection and segmentation has already been applied to identify the shape and location of masonry units from images (Sithole, 2008; Oses, Dornaika and Moujahid, 2014; Cluni *et al.*, 2015; Brackenbury and Dejong, 2018; Valero, Bosché and Forster, 2018), their application proves challenging due to digital noise produced by the change in illumination, colour, and texture presented within the digital images.

An alternative approach for image segmentation of masonry units and mortar is with the use of ML (Machine-Learning), such as Deep-Learning (DL; subset of M.L.) (Garcia-Garcia *et al.*, 2017; Spencer, Hoskere and Narazaki, 2019). Some typical examples of DL include:

- *Classic-Networks:* Multilayer architecture of fully-connected-layers of neurons, which are typically used in data classification and predictions.
- *Convolutional-Neural-Networks (CNN):* Typically used for image classification and segmentation.
- *Fully-Concolutional-Networks (FCN):* Typically used for image-segmentation, often combined with a CNN backbone.

The most efficient image-segmentation architectures consider the use of FCN (Chen *et al.*, 2015, 2018; Long, Shelhamer and Darrell, 2015; Ronneberger, Fischer and Brox, 2015; Lin *et al.*, 2017; Chaurasia and Culurciello, 2018), since they allow any image resolution as input by replacing the final fully-connected-layers of a CNN with convolution layers (Long, Shelhamer and Darrell, 2015). CNN and FCN architectures require a large labelled or annotated dataset, trained in representative sample to provide adequate results. However, they have the potential to detect complex features by training the model to a large variety of different cases. Furthermore, for smaller datasets, DL approaches can use a technique called Transfer-Learning which involves pre-training a CNN model on a different and larger dataset with the purpose to learn to detect complex features. This has been shown to provide

a reduction to the required computational effort and a boost to overall performance of the model for smaller datasets (Hussain, Bird and Faria, 2019). Transfer learning can be applied to any CNN and FCN (Fully Convolutional Networks) architecture. Multiple architectures have been used to demonstrate the ability of DL in the semantic segmentation of images.

There are different architectures that could be used for the segmentation of masonry units. U-Net is an FCN architecture that was developed initially for biomedical image-segmentation (Ronneberger, Fischer and Brox, 2015). It is based on a contracting followed by an expansive path, that initially decreases and then increases the input-size. Due to its performance, U-Net is considered the benchmark in image-segmentation and has been used extensively especially in relatively small datasets. DeepLabV3+ is another state-of-the-art FCN architecture for general use semantic segmentation (Chen *et al.*, 2015, 2018). Additional features on DeepLabV3+ compared to previous iterations and simpler architectures aim to deliver a faster and more accurate network. FPN (Feature-Pyramid-Network) is an FCN network for object detection (Lin *et al.*, 2017). It is a feature extractor that follows a bottom-up followed by a top-down path with the addition of lateral connections between the two to merge feature maps of equal spatial size. LinkNet is a light FCN network developed for pixel-wise segmentation optimised for efficiency (Chaurasia and Culurciello, 2018). LinkNet is a lightweight and fast FCN architecture able to be used for real time applications (i.e., video streaming).

ML applications have already seen use in structural engineering due to their immerse potential to assist with visual inspection and monitoring applications (Spencer, Hoskere and Narazaki, 2019). In cases of damage detection, ML provides the means to identify, locate, and asses detected deterioration on structural elements. Valero et al. (2019) extracted statistical data from a 3D point-cloud and used them to train a ML algorithm for the detection and classification of chromatic (i.e., discoloration) and geometric defects on ashlar masonry (using logistic regression with multi-class classification). However, most typical applications of defect detection include the use of DL due to its architecture, which allows the detection of complex features on unstructured data. Chaiyasarn et al., 2018, investigated the use of patch classification using CNN algorithm coupled with a classifier to identify small patches that contained damaged location of historical structures. Their work was continued by Ali (2019) , where Faster-R-CNN was used for the detection of bounding-boxes that contain locations of damaged bricks on masonry structures. The same year, Wang et al. (2019) used a Faster-R-CNN model based on ResNet101 for the real time detection and classification of bounding-boxes that include defected areas on historic masonry buildings. A workflow that utilises mobile-phones for the direct capture and processing of image-data was also proposed by them. Brackenbury et al. (2019) discussed the use of GoogleNet-Inception-V3 algorithm and the use of transfer learning for the classification and segmentation of mortar and defects in masonry components. Each defect (i.e., cracking, spalling, or vegetation) was classified separately. Kalfarisi et al. (2020) used Mask-RCNN and FRCNN-FED to detect bounding boxes that contain cracks on structures and performed pixel-wise segmentation within the detected areas. Furthermore, they transferred the segmented locations to a 3D reality-mesh object, generated using photogrammetry. Recently, Dais et al. (2021) tested different CNN algorithms for the detection of cracks on masonry images. Both patch-classification (with 95.3% accuracy) and semantic-segmentation (with 79.6% F1-score) on pre-trained networks were investigated.

Although most research focuses in defect-detection, the detection of masonry-units and mortar is essential to achieve a comprehensive visualization of the condition of masonry structures. Ibrahim et al. (2019) proposed the use of U-Net for the segmentation of mortar in masonry structures with different bonding pattern (i.e., including rubble). Additionally, they used watershed-transform for the segmentation of each brick unit. Ergün Hatir & İnce (2021) proposed the use of Mask-R-CNN for the classification and segmentation of masonry units in historic stone masonry buildings. Each stone detected was classified to a different lithology based on their detected features (i.e., colour, texture, etc).

Based on the information above, while there has been some progress in crack and mortar segmentation, no previous research has explored the integration of brick-segmentation with crack-detection in masonry structures. The objective of this study is to combine brick-segmentation and defect-detection techniques to automatically offer comprehensive, real-time information for documenting, visually inspecting, and evaluating existing masonry structures. In this research, brick-segmentation and defect-detection were acquired using different state-of-the-art FCN architectures including U-Net, DeepLabV3+, U-Net (SM), LinkNet (SM), and FPN (SM). The work presented here provides an automatic workflow for the assessment of masonry structures from digital images.

3.2. Development of the database for training and evaluation

Initially, a database was created that includes various images of brick masonry walls of regular pattern (ignoring rubble masonry). Some images were obtained from the internet while others were captured using different sources (i.e., DSLR camera, smartphones) of varied resolution. To improve generalisation, the dataset included masonry walls with cracks, with windows and doors, with varied in colour masonry units as well as with varied illumination and capture-angle. A sample of the raw database used for training and evaluation is shown in Fig. 3.1.

Fig. 3.1: Sample of the raw database used for training-evaluation.

In total 107 images of masonry structures were fully annotated, including multi-class annotations of masonry blocks, openings, lintels, other/random objects, and background (Fig. 3.2). Each class was annotated to a different binarized image where black was the background and white was the annotated element. The software used for annotation was the desktop version of "*SuperAnnotate V.1.1.0*". This specific software was selected since it allowed vector annotations which permits a simplified annotation of each block, ignoring unnecessary details. It was found that simpler shapes could allow easier transferability to CAD environment.

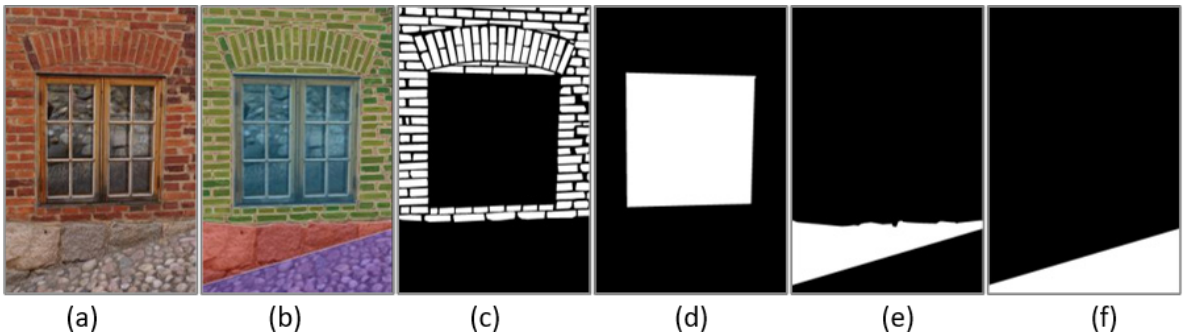


Fig. 3.2: Annotation of ground-truth data; a) Original image; b) SuperAnnotate vector classification; c) Bricks; d) Openings; e) Structural; f) Background.

Each image resolution was normalized based on the resolution of the image-slice passed through the network. This was to allow each slice to contain several blocks, but not allow the average block-size to be larger than the image-slice. Doing so, the accuracy of the model has increased, since each image-slice had similar-sized blocks that improved the detection rate. The normalised resolution of each image was evaluated and compared with the normalised resolution of the image-slice (image-part passed through the network). Thus, the image is allowed a specific range of resolution. Although, using a specific resolution of block elements (i.e., pixels contained within a block; by capturing pictures with specific resolution, angle, and distance), would potentially increase the accuracy. However, that would reduce generalisation of the final model and complicate its use (i.e., would require the user to capture images from specific distance/angle). Equations 1 to 5 were used to adjust each image. If the image was within the limits of Eq. (32), it retained its resolution. However, if the image was outside the limits of Eq. (3), it was adjusted based on Eq. (34).

$$ImDim = \sqrt{x_{image} * y_{image}} \quad (30)$$

$$OutDim = \sqrt{x_{slice} * y_{slice}} \quad (31)$$

$$MaxLimit = OutDim * MaxRatio \;\&\; MinLimit = OutDim * MinRatio \quad (32)$$

$$Scale = Limit/ImDim \quad (33)$$

$$(x_{final}, y_{final}) = (x_{image}, y_{image}) * Scale \quad (34)$$

,where, $ImDim$ is the average image-resolution, $OutDim$ is the average image-slice resolution, $MaxLimit/MinLimit$ is the maximum over the minimum limit that is allowed to disregard further adjustments, and x_{final}/y_{final} is the final resolution per image-axis.

Furthermore, the only variables provided were the $MaxRatio/MinRatio$ that are the Maximum over the Minimum preferred ratio to adjust the images based on the size of the image-slices. Those values were adjusted manually until all the image slices contained a satisfactory number of block elements for the specific database. For the current database, the values used were the following:

$$MinRatio = 2 \text{ , and } MaxRatio = 4 \quad (35)$$

The size of each image slice was equal to 224 × 224 × 3 (i.e., $OutDim = 224$). Each image was padded to adjust its resolution to multiples of the slice-resolution per axis (i.e., 224 pixels), to retain the original aspect-ratio when the image is sliced to smaller parts. The padding value was equal to 255, which created a white border around most of the edges. Then, the white padding was used as filtered locations of post-processed images (i.e., images were background and openings have been replaced with white) and would be instantly disregarded from the CNN output. This provided a total of 2,814 image-slices and were used as training and validation (i.e., approx. 25% of them were used for validation), see (Fig. 3.3). Other slice-resolutions were also tested. The smaller resolutions provided more accurate models. This was due to the increased number of training and validation data.

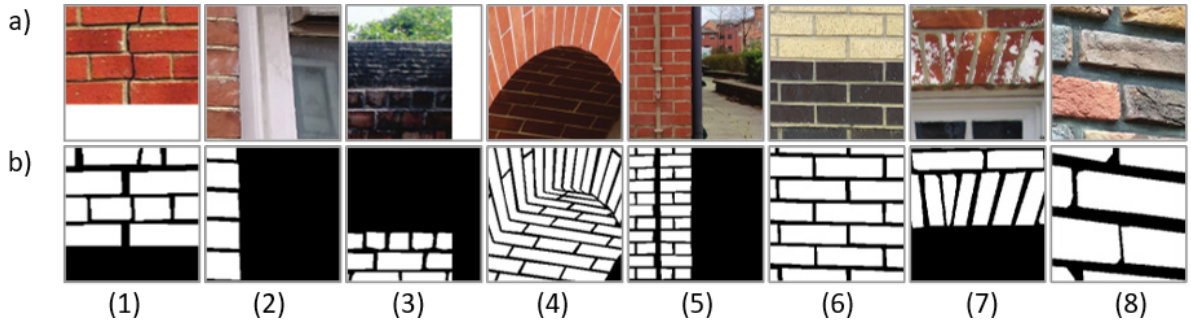


Fig. 3.3: Sample of the slices used to train and evaluate the model; Top: Original image slice; Bottom: Annotated blocks.

3.3. Convolutional Neural Networks

In this research, the networks evaluated were the U-Net, DeepLabV3+ (Fig. 3.4), U-Net (SM), LinkNet (SM), and FPN (SM). The latter three (i.e., the *SM*'s) were generated through the python package called "*Segmentation Models*", which includes: ready-to-use semantic-segmentation models, multiple backbones of renown architectures, and pretrained models for transfer learning. The training procedure involved only the use of the brick class since the dataset of other classes was not considered to be large enough.

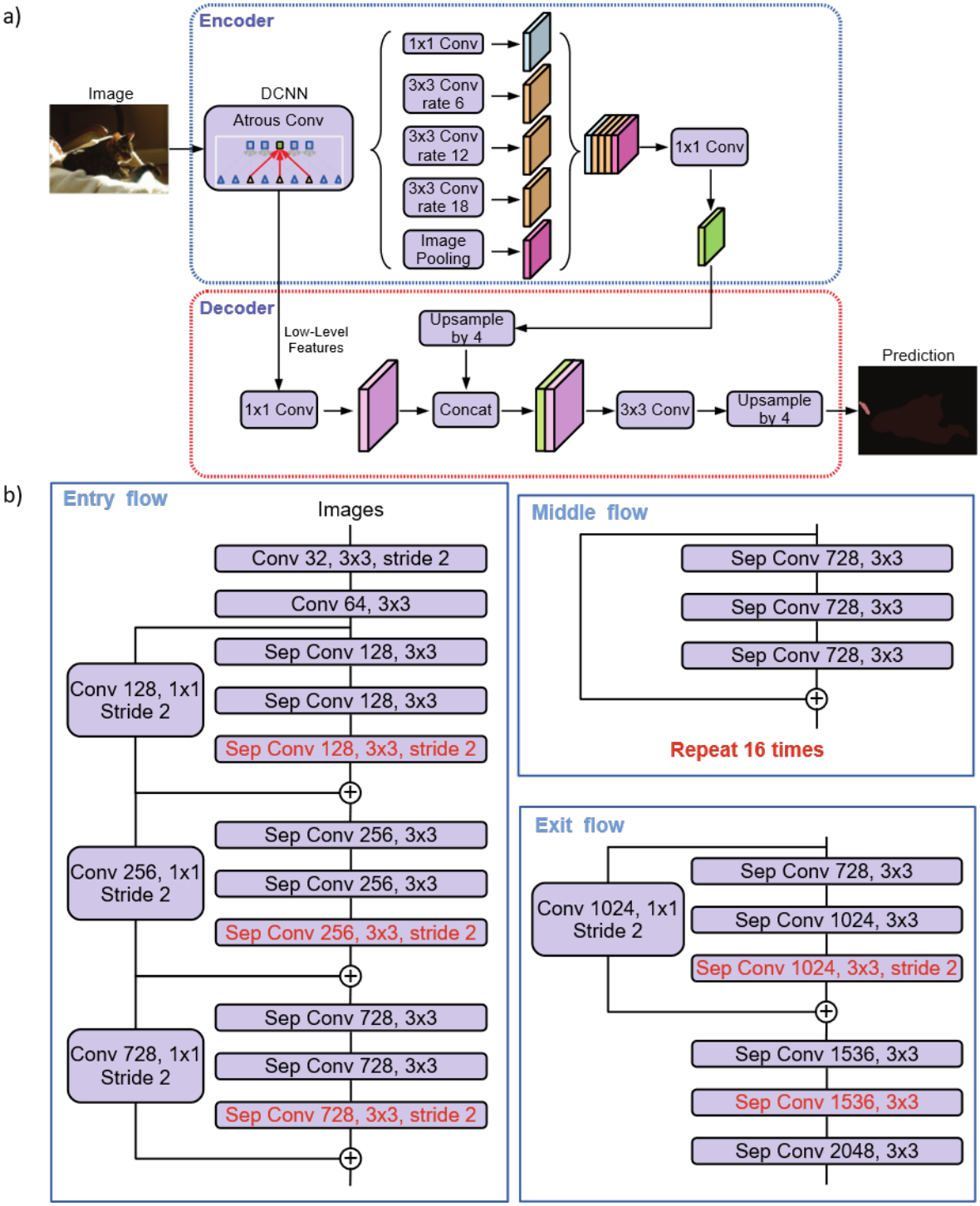


Fig. 3.4: Architecture of the highest performance model (Chen et al.*, 2018); a) DeepLabV3+ architecture; b) Modified XCEPTION backbone.*

Multiple tests were conducted to identify the best combination of backbone, pretrained dataset, loss function, optimiser, and parameters (see Table 3.1), that would provide the most efficient model. The first test included a combination of different parameters except loss, optimiser, and activation function (Table 3.1). Every architecture was tested with the "*Adam*" optimiser and "*Weighted-Cross-Entropy*" loss function. The learning-rate of the first test-sequence was equal to 5E-4 with a decay of 5E-6 over 100 epochs.

Table 3.1: Testing different parameters for each provided architecture (bold indicates best of each section).

Architecture	Model	Backbone	Val. Acc.	Val. F1	Val. Precision	Val. Recall	Val. Loss
-	-	-	%	%	%	%	-
U-Net	#1	-	95.62	95	0.92	98.26	0.37
U-Net	#2	-	95.4	94.71	92.35	97.45	0.54
U-Net	#3	-	95.51	94.76	92.18	97.8	0.42
U-Net	#4	-	95.86	95.19	93.01	97.68	0.48
U-Net	#5	-	93.39	92.37	88.03	97.51	0.58
U-Net	#6	-	96.02	95.4	93.05	97.99	0.41
U-Net	#7	-	95.88	95.21	93.77	96.86	0.63
U-Net (SM)	#1	VGG16	95.1	94.3	92.1	96.78	0.65
U-Net (SM)	#2	VGG19	95.17	94.32	91.96	97.01	0.59
U-Net (SM)	#3	InceptionV3	95.37	94.6	94.06	95.38	0.97
U-Net (SM)	#4	Inception-ResNetV2	89.07	72.46	72.9	75.83	1.23
U-Net (SM)	#5	MobileNet	95.83	95.19	93.79	96.78	0.69
U-Net (SM)	#6	ResNet50	94.65	93.88	91.06	97.09	0.59
U-Net (SM)	#7	SeresNet101	95.13	94.4	92.45	96.61	4.58
U-Net (SM)	#8	SeresNet152	94.77	94.09	90.69	97.96	4.21
U-Net (SM)	#9	ResNet152	94.74	93.85	91.2	96.96	0.64
U-Net (SM)	#10	MobileNet	95.98	95.24	93.64	97.02	0.61
U-Net (SM)	#11	MobileNet	95.94	95.26	93.41	97.31	0.56
U-Net (SM)	#12	MobileNet	95.77	95.09	93.13	97.25	0.59
LinkNet (SM)	#1	MobileNet	96.13	95.52	94.26	96.92	0.64
LinkNet (SM)	#2	MobileNet	95.91	95.31	93.4	97.41	0.55
LinkNet (SM)	#3	MobileNet	95.92	95.35	93.3	97.59	0.54
FPN (SM)	#1	MobileNet	95.99	95.42	93.32	97.71	0.59
FPN (SM)	#2	MobileNet	95.89	95.3	92.9	97.93	0.48
FPN (SM)	#3	MobileNet	96.07	95.53	93.45	97.77	0.53
FPN (SM)	#4	MobileNet	95.99	95.39	93.61	97.35	0.58
DLV3+	#1	Xception	93.12	87.02	91.47	85.33	4.51
DLV3+	#2	Xception	96.26	95.6	95.12	96.24	0.81
DLV3+	#3	Xception	96.27	95.66	94.81	96.66	0.75
DLV3+	#4	MobileNetV2	95.85	95.26	93.45	97.22	0.5
DLV3+	#5	Xception	95.4	94.71	92.35	97.45	0.54

All architectures provided similar results (between 95.98% to 96.27% validation-accuracy), with DeepLabV3+ (#3) having the highest accuracy (96.27%). For each architecture, the parameters used for the optimal model were:

a) U-Net (#6): Optimized using 64 output filters, 0.0005 L2-Regularization, 0.25 dropout, Batch-Normalization, and "*glorot-uniform*" initializer.
b) U-Net-SM (#10): 0.0005 L2-Regularization, "*MobileNet*" backbone, "*ImageNet*" Encoder-Weights, "*Sigmoid*" activation function, and (256, 128, 64, 32, 16) decoder filters.
c) U-Net-SM (#11) is like U-Net-SM (#10) except that (512, 256, 128, 64, 32) decoder filters were used.
d) LinkNet-SM (#1): 0.0005 L2-Regularization, "*MobileNet*" backbone, "*ImageNet*" Encoder-Weights, "*Sigmoid*" activation function, and (1024, 512, 256, 128, 64) decoder filters.
e) FPN-SM (#3): 0.0005 L2-Regularization, "*MobileNet*" backbone, "*ImageNet*" Encoder-Weights, "*Sigmoid*" activation function, 512 filters and 0.25 dropout.
f) DeepLabV3+ (#3): 16 OS (feature-extractor output ratio), "*Xception*" backbone, "*Pascal-Voc*" pretrained weights, and "*Sigmoid*" activation.

3.4. Loss Function

The loss function was used to minimise the error during training and define the weights to reduce the loss during the next evaluation. The loss functions tested were Focal-Loss (FL), Weighted-Cross-Entropy (WCE), F1-Loss (F1L), and Binary-Cross-Entropy (BCE). All cases used the "Adam" optimizer with learning-rate of 1E-4 and decay equal to 1E-6.

Table 3.2: Test of loss functions (bold indicates best model of each section)

				Best of three of each model/loss combination				
Architecture	Model	Loss	Epoch	Val Accuracy	Val F1	Val Precision	Val Recall	Val Loss
-	-	-	-	%	%	%	%	-
U-Net	#6	FL	98	96.18	95.3	97.12	93.72	3241.99
U-Net	#6	WCE	78	95.44	94.69	92.77	96.86	0.66
U-Net	#6	F1L	97	96.09	95.22	95.6	95.09	0.05
U-Net	#6	BCE	92	96.27	95.57	96.12	95.14	0.13
U-Net (SM)	#10	FL	89	96.41	95.52	96.82	94.5	6022.23
U-Net (SM)	#10	WCE	74	95.99	95.28	94.36	96.35	0.97
U-Net (SM)	#10	F1L	74	96.08	95.32	95.98	94.81	0.1
U-Net (SM)	#10	BCE	99	96.52	95.79	96.5	95.21	0.18
LinkNet (SM)	#1	FL	99	96.06	95.11	97.17	93.29	4998.93
LinkNet (SM)	#1	WCE	80	96.07	95.25	94.3	96.41	0.92
LinkNet (SM)	#1	F1L	87	96.49	95.8	96.15	95.56	0.08
LinkNet (SM)	#1	BCE	95	96.54	95.83	96.24	95.49	0.17
FPN (SM)	#3	FL	93	96.36	95.55	96.74	94.53	5473.35
FPN (SM)	#3	WCE	87	96.21	95.5	93.95	97.2	0.78
FPN (SM)	#3	F1L	100	96.52	95.8	96.78	94.92	0.08
FPN (SM)	#3	BCE	85	96.58	95.88	96.36	95.49	0.19
DLV3+	#3	FL	59	96.31	95.55	97.08	94.17	3632.67
DLV3+	#3	WCE	83	96.09	95.47	93.94	97.15	0.57
DLV3+	#3	F1L	69	96.45	95.77	96.53	95.13	0.04
DLV3+	#3	BCE	97	96.65	96.03	96.53	95.62	0.15

Table 3.2 presents the best of three of each architecture/loss combinations. After evaluating all cases, the most efficient loss-function (highest validation-accuracy) was the BCE. However, it has been noticed that the highest validation-precision was typically acquired when using FL and the highest validation-recall when using WCE. Nonetheless, the target metric was the validation-accuracy. Thus, the optimal loss function was taken equal to the BCE. Furthermore, to exclude any error in the evaluation procedure of the loss-functions, a combination of different optimisers per loss-function was tested. However, the evaluation of the optimiser was only undertaken for the DeepLabV3+ architecture since it had the highest validation-accuracy for both tests. The remaining parameters were equal between the second and third tests. The two optimisers used herein were: a) Adam, Stochastic Gradient Descent (SGD); and b) RMSprop (RMSP).

Table 3.3: Test of optimiser/loss combination (Bold indicates best of each section)

				Best of three of each optimiser/loss combination				
Architecture	Optimizer	Loss	Epoch	Val Accuracy	Val F1	Val Precision	Val Recall	Val Loss
-	-	-	-	%	%	%	%	-
DLV3+	SGD**	FL	62	56.72	0	0	0	350036.3
DLV3+	SGD	FL	62	56.72	0	0	0	350036.3
DLV3+	SGD**	WCE	72	86.14	84.6	79.51	92.45	1.95
DLV3+	SGD	WCE	96	75.16	73.23	66.15	84.95	4.75
DLV3+	SGD**	F1L	96	76.52	74.38	68.73	84.36	0.26
DLV3+	SGD	F1L	100	72.47	67.84	63.72	74.88	0.33
DLV3+	SGD**	BCE	25	83.59	79.86	81.19	81.05	6.08
DLV3+	SGD	BCE	100	73.65	67.11	67.45	70.25	4.58
DLV3+	RMSP*	FL	55	94.93	93.75	96.22	91.67	5921.18
DLV3+	RMSP	FL	44	96.4	95.75	96.51	95.1	7827.02
DLV3+	RMSP*	WCE	46	94.09	93.1	92.58	94	1.48
DLV3+	RMSP	WCE	88	96.2	95.5	94.75	96.39	1.2
DLV3+	RMSP*	F1L	91	95.87	94.92	95.49	94.72	0.05
DLV3+	RMSP	F1L	56	96.72	96.09	96.81	95.47	0.04
DLV3+	RMSP*	BCE	76	95.84	94.9	95.36	94.79	0.26
DLV3+	RMSP	BCE	96	96.57	95.92	96.19	95.75	0.23
DLV3+	Adam	FL	59	96.31	95.55	97.08	94.17	3632.67
DLV3+	Adam	WCE	83	96.09	95.47	93.94	97.15	0.57
DLV3+	Adam	F1L	69	96.45	95.77	96.53	95.13	0.04
DLV3+	Adam	BCE	97	96.65	96.03	96.53	95.62	0.15

* Momentum = 0.9

** Momentum = 0.9, Nesterov = True

Default (No Stars): Momentum = 0, Nesterov = False

From Table 3.3, each optimiser provided the most efficient model with different loss function. So, the use of BCE as the optimal loss function was not universal. Using the SGD optimiser, the most efficient loss function was WCE (86.14% validation accuracy). With RMSP, the optimal loss function was F1L (96.72% validation accuracy). Using Adam, the highest score was obtained through BCE (96.65% validation accuracy). Also, from Table 3.3, it is concluded that the combinations (Optimiser/Loss) with the highest accuracy were the Adam-BCE and RMSP-F1L and had very similar accuracy. So, both have

been considered for the development of the final model. In contrast, the SGD optimiser was disregarded due to the low accuracy score along all loss functions.

3.5. Final Model

The optimal learning-rate used to adjust the final model and decide on the utilisation of F1L and BCE loss functions. RMSP was selected as the target optimiser, since it obtained the highest score, see (Table 3.3). Different learning-rate values were tested over 200 epochs. The decay used was equal to the Learning-Rate over the Max-Epoch. All models used the DLV3+ architecture with the Xception backbone, pretrained to the Pascal-VOC dataset (Table 3.4).

Table 3.4: Testing different learning-rate values for the selection of the final model (bold indicates best of each section).

				Best of three (DLV3+)				
Optimizer	Loss	Learning Rate	Epoch	Val Accuracy	Val F1	Val Precision	Val Recall	Val Loss
-	-	-	-	%	%	%	%	-
RMSP	F1L	1.00E-04	79	96.73	96.16	96.54	95.82	0.04
RMSP	F1L	2.00E-04	117	96.86	96.29	96.68	95.94	0.04
RMSP	F1L	5.00E-05	67	96.57	95.96	96.32	95.65	0.04
RMSP	F1L	0.0005	179	96.62	96.03	96.16	95.93	0.04
RMSP	BCE	1.00E-04	127	96.87	96.3	96.46	96.16	0.23
RMSP	BCE	2.00E-04	131	96.85	96.28	96.64	95.94	0.37
RMSP	BCE	5.00E-05	165	96.48	95.85	96.17	95.59	0.26
RMSP	BCE	0.0005	88	96.46	95.84	96.01	95.7	0.24

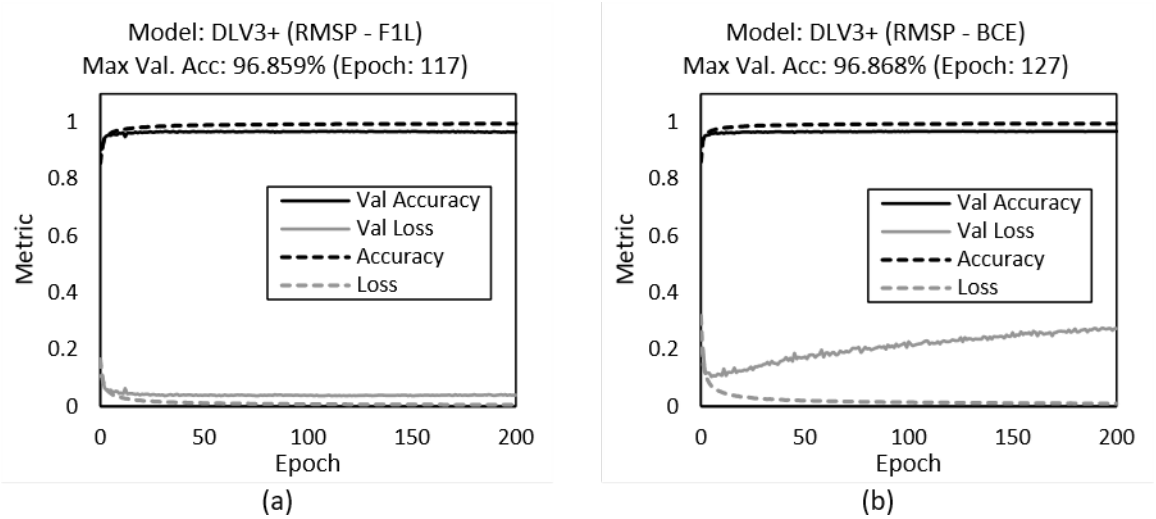


Fig. 3.5: Graphs of best models; a) DLV3+ model with F1L loss function and learning rate of 2E-4; b) DLV3+ model with BCE loss function and learning rate of 1E-4.

The highest score using F1L loss function was obtained with 2E-4 learning rate with validation accuracy equal to 96.86% (Table 3.4). The highest score using BCE loss function was obtained with 1E-4 learning rate with validation accuracy equal to 96.87% (Table 3.4). The validation accuracy of both models was very similar. Thus, the selection of the final model considered the progression of the loss on the accuracy/loss graphs (Fig. 3.5) and the visual representation of the validation set (Fig. 3.6 and Fig. 3.7).

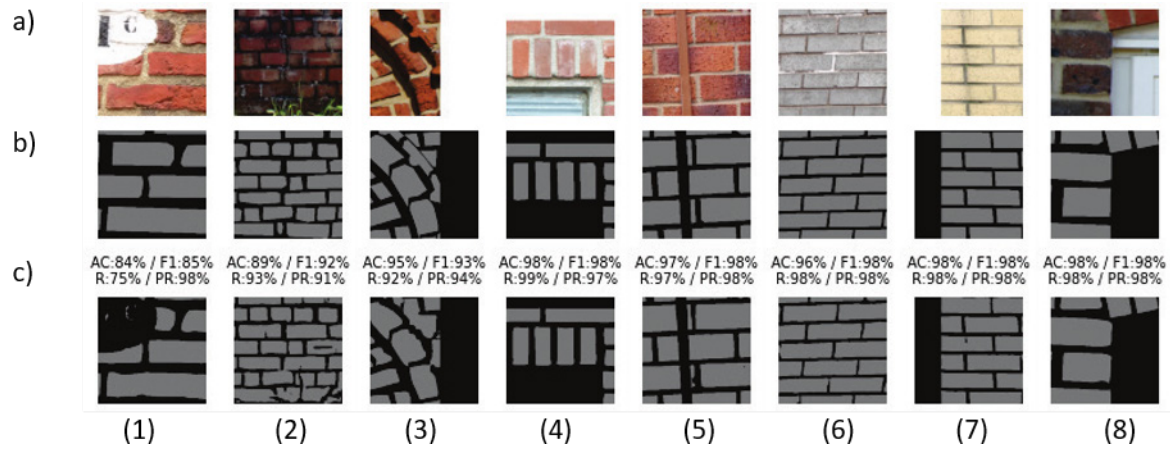


Fig. 3.6: Evaluation sample from the F1L model; a) Original image-slice; b) Ground truth; c) CNN Output (AC: Accuracy; F1: F1-Score; R: Recall; PR: Precision)

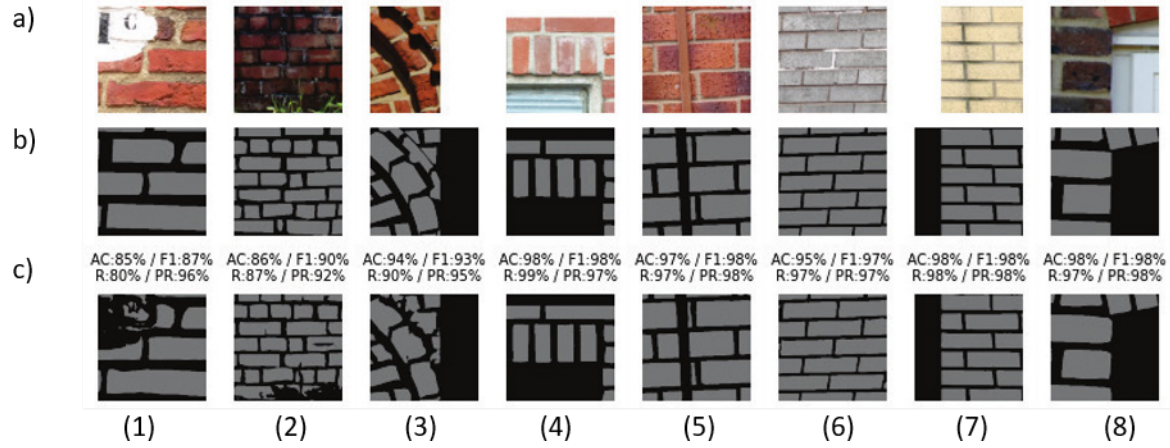


Fig. 3.7: Evaluation sample from the BCE model; a) Original image-slice; b) Ground truth; c) CNN Output (Metrics: AC: Accuracy, F1: F1-Score, R: Recall, PR: Precision)

The output graph of the BCE model shows moderate overfitting to the dataset provided (Increasing validation-accuracy and validation-loss), which reveals that the BCE model may not be generalising as well as the model with F1L. Moreover, from the samples provided, the model with F1L has reduced noise on complex locations (i.e., images 1-3 in Fig. 3.6 and Fig. 3.7). Although using BCE the validation score was slightly higher in the model, the reduction of noise assisted with the detection of individual blocks on more complex images. Also, both models provided very accurate results for images with adequate resolution per block. Moreover, both were able to recognise openings and backgrounds exceptionally well, even for bricks with varied colour (i.e., images 4 and 8 in Fig. 3.6 and Fig. 3.7). So, the model with F1L loss-function was considered as the best model and adopted here. In more detail, the specified model has a classification error equal to 1.24% for the background and 1.52% for the blocks class (Fig. 3.8). Additionally, the model is aimed to be used for images that are simpler than the trained dataset. Thus, in practice, the CNN-output is expected to provide improved results.

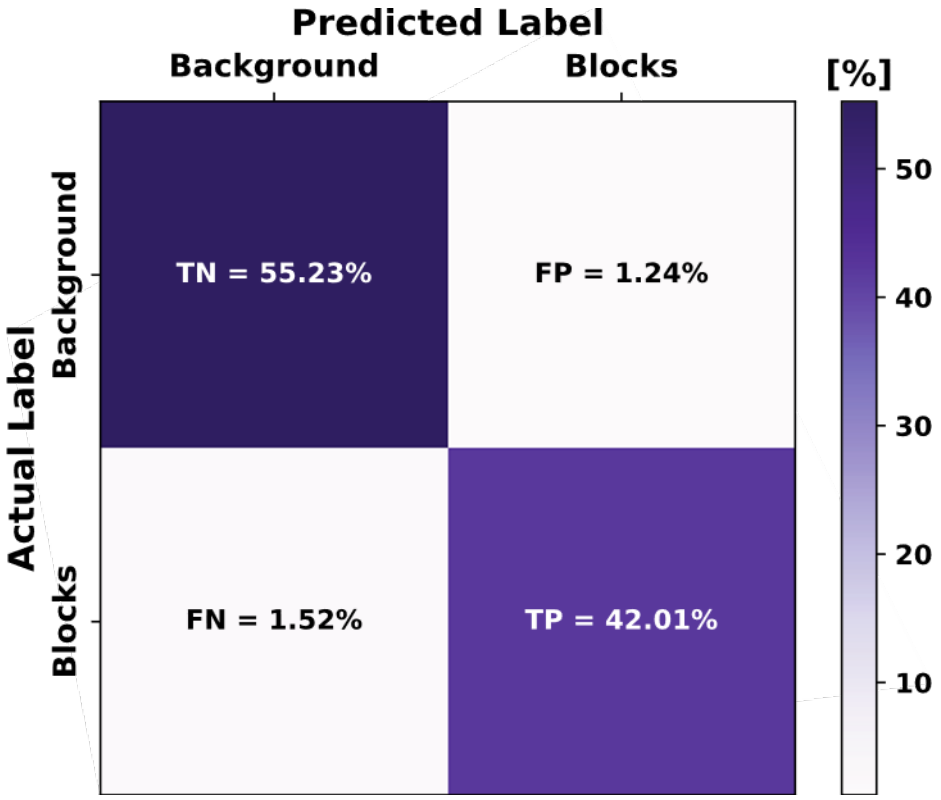


Fig. 3.8: Confusion matrix; TN: True Negatives; FN: False Negatives; FP: False Positives; TP: True Positives (Model: DLV3+ with RMSP-F1L)

3.6. Crack Detection

As mentioned before, the crack detection adopted in this study used the most efficient model presented in Dais et al., (2021). Both patch classification and pixel wise segmentation was tested. However, for the purposes of this study, only the models for pixel wise segmentation were considered (Table 3.5). The architectures tested were: a) DeepCrack; b) DeepLabV3+; c) FCN based on VGG16; d) U-Net; and e) FPN. The backbones tested were: VGG16, ResNet (multiple), DenseNet (multiple), Inception, MobileNet, and MobileNetV2. Also, multiple loss functions were tested to identify their optimal parameters. The loss functions used were: a) Weighted-Cross-Entropy (WCE), b) Cross-Entropy (CE), c) F1-Score-Loss (F1), and d) Focal-Loss (FL). All models included the use of the Adam optimiser since it provided the highest F1-Score. Transfer learning was also utilised to improve the accuracy of the detection using the ImageNet dataset.

Table 3.5: Architectures tested for defect detection of masonry structures (bold indicates best models).

							Validation Scores		
Network	Pretrained	Loss	Parameters	Model Size	Analysis Time	Best Epoch	F1 Score	Recall	Precision
-	[ImageNet]	-	[Millions]	[MB]	[Hours]	-	%	%	%
DeepCrack	No	WCE	29.5	115.5	5.2	28	74	80.1	71.6
DeepLabv3+	No	WCE	41.3	162.2	5.6	26	74.9	79	73.8
FCN-VGG16	No	WCE	27.8	108.8	2.5	95	75.6	76.6	76.9
U-net	No	WCE	34.5	135.1	5.8	75	75.7	78.9	75.7
U-net-VGG16	Yes	WCE	46.1	180.2	6	37	77.2	81.2	76.2
U-net-ResNet34	Yes	WCE	48	188.1	4.9	61	77.6	78.3	79.5
U-net-ResNet50	Yes	WCE	73.7	288.5	6.8	45	76.3	80.9	74.8
U-net-Densenet121	Yes	WCE	41.6	163.5	6.2	55	78.1	80.7	78.1
U-net-Densenet169	Yes	WCE	54.3	213.4	7.1	63	78.5	83.5	76.2

U-net-InceptionV3	Yes	WCE	68.5	268.1	6.8	31	77.7	79.2	78.9
U-net-MobileNet	Yes	WCE	37.8	147.9	4.8	45	79.6	79.9	81.4
U-net-MobileNet	No	WCE	37.8	147.9	4.8	36	75.4	80.7	73.4
U-net-MobileNet	Yes	CE	37.8	147.9	4.8	36	76.6	73	83
U-net-MobileNet	Yes	F1	37.8	147.9	4.8	29	78.2	77.1	82
U-net-MobileNet	Yes	FL	37.8	147.9	4.8	85	71.2	61.1	89.4
U-net-MobileNetV2	Yes	WCE	39.5	154.9	5.3	58	77.7	76.6	81.9
FPN-VGG16	Yes	WCE	32.2	125.8	5.6	79	77.9	82	76.2
FPN-ResNet34	Yes	WCE	38.3	150.2	5.2	36	78	81.5	77.2
FPN-ResNet50	Yes	WCE	42.1	164.8	5.8	27	77.2	81.4	75.8
FPN-Densenet121	Yes	WCE	24.6	97	6.1	31	79	83.6	77.2
FPN-Densenet169	Yes	WCE	30.6	120.8	6.6	59	78.6	80	79.5
FPN-InceptionV3	Yes	WCE	40	157.2	5.7	34	79.6	81.3	80.1
FPN-MobileNet	Yes	WCE	20.8	81.4	4.6	40	79.5	79.5	81.7
FPN-MobileNetV2	Yes	WCE	19.9	78.3	4.8	49	78.5	76.7	82.7

The architecture selected was the U-net-MobileNet with Adam optimiser and WCE loss function. The specified model achieved a validation F1-score equal to 79.6%, validation recall equal to 79.9%, and validation precision equal to 81.4%. The existing CNN-model was used to acquire the damage during the geometrical-model generation. Moreover, the dataset of the damage-detection model is similar to the block-detection model. Thus, it can be used directly to combine the results of both models (blocks and cracks) efficiently. The sample of the validation set used in the evaluation of the model is shown in Fig. 3.9.

Fig. 3.9: Sample images from the crack detection model from (Dais et al.*, 2021); a) Original image-slice; b) Ground truth; c) CNN Output (Metrics: F1: F1-Score, RE: Recall, PR: Precision)*

3.7. Final Output

To acquire the final output, image processing of the original image was undertaken. Initially the image was re-sized using the same methodology described in the Development of the database section (Eq. (34)). By combining the image slices directly to the image, distortion effects near the edges of the image-slice (Fig. 3.10: b) were observed. So, each slice assigned an overlap value. The best results were acquired using an overlap value of 50 pixels for an image slice of 224 × 224. The image was divided into sections of 124 × 124 pixels ($224 - 2 \times 50$) and included a white padding of 100 pixels (2×50). This effectively retained only the central section of each slice for use and improved the overall quality of the final output (Fig. 3.10: c). Furthermore, the use of the models to acquire the location of cracks

and masonry elements shown satisfactory results. Fig. 3.11 presents outputs from both models CNN (blocks and cracks) that were used for verification purposes i.e., not used during the training/evaluation phase.

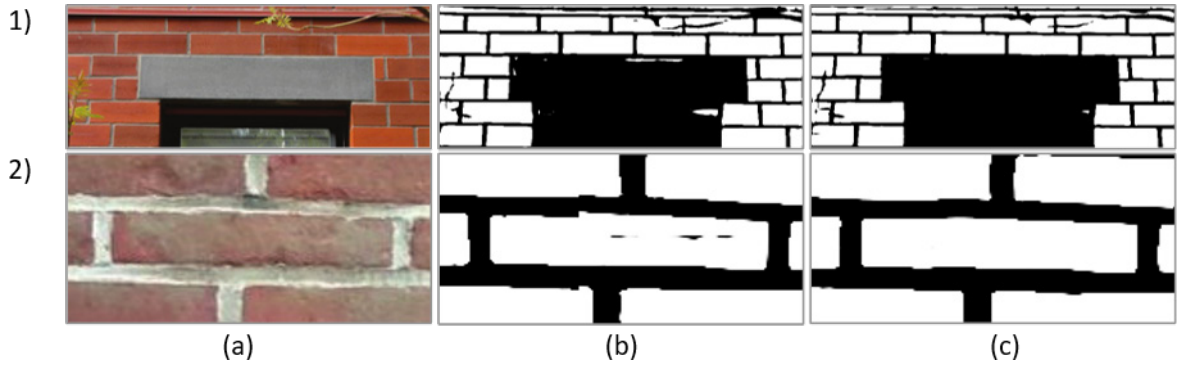


Fig. 3.10: Effect of using overlap while connecting output-slices; a) Original image; b) Direct connection of 224x224 pixel slices; c) Overlap of 50 pixels on 224x224 pixel slices.

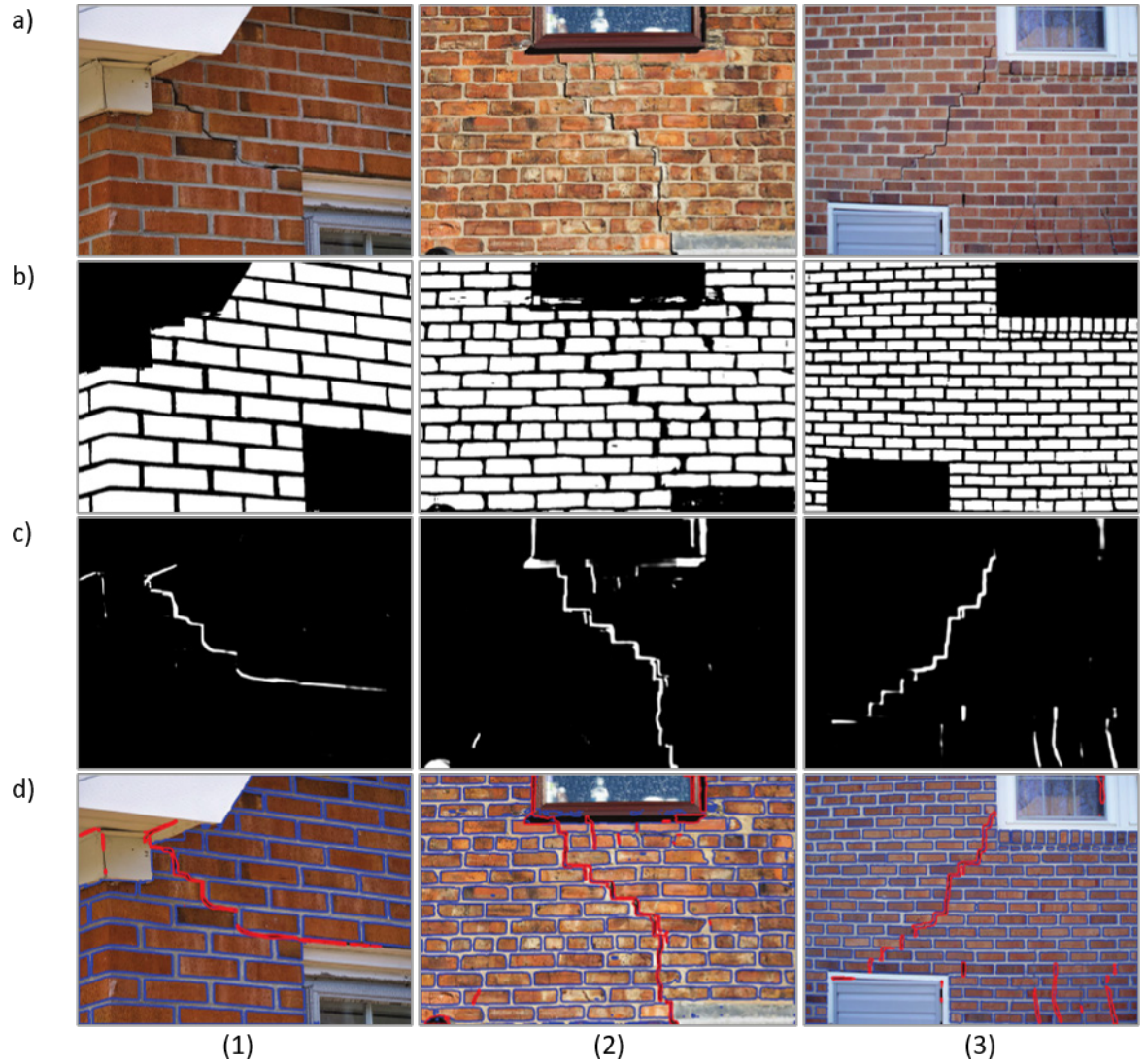


Fig. 3.11: Combined output of block and crack detection models; a) Original image; b) Block detection; c) Crack detection; d) Marked perimeter of detected elements.

3.8. Usage

One important use of the damage detection framework proposed here is to assist engineers with the visual-inspection and documentation of masonry structures in their care. Using image processing, each individual crack was identified and measured. The isolation of white elements from the CNN output was succeeded by using watershed segmentation to assign a unique label to each crack (Fig. 3.12: b). Using the individual labels of the segmentation, it was possible to acquire the area of each label by counting the total number of pixels. Each segmentation provided the linearization of its area (Fig. 3.12: c), which can be used to evaluate the length of the crack. Finally, the results obtained can be scaled to the real dimensions and provide realistic measurements of the crack properties by providing a scale factor (Table 3.6). The approximate-length (Skeleton (mm)) was acquired under the assumption that the length of each pixel is the average between its horizontal and diagonal distance.

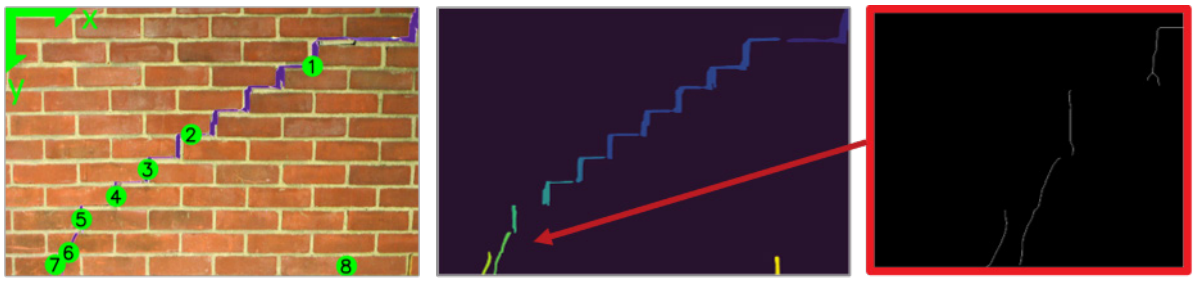


Fig. 3.12: Evaluation of damage; a) Marked cracks; b) Watershed segmentation; c) Linearization (skeleton).

Table 3.6: Crack properties acquired using image-processing.

Label	Loc Min (xmin,ymin)	Loc Max (xmax,ymax)	Loc Mid (x,y)	Area (pixels)	Skeleton (pixels)	Area (mm^2)	Skeleton (mm)
1	[1908, 0]	[2249, 189]	[2079, 170]	13474	472	5072	350
2	[923, 155]	[1872, 817]	[1397, 427]	37425	1482	14088	1065
3	[753, 804]	[913, 945]	[775, 874]	5537	247	2084	178
4	[568, 936]	[739, 1072]	[595, 993]	5675	272	2136	196
5	[402, 1069]	[428, 1220]	[414, 1144]	3076	137	1158	98
6	[302, 1215]	[397, 1448]	[343, 1325]	3971	229	1495	165
7	[237, 1323]	[289, 1448]	[270, 1390]	2217	121	835	87
8	[1843, 1348]	[1869, 1448]	[1856, 1398]	2190	79	824	57

The main use of the feature-detection was aimed for the automatic development of geometrical models for documentation and numerical models for analysing the structural capacity of masonry structures. The methodology to convert binary images of masonry blocks and cracks, to CAD drawings, is further explained in Loverdos et al. (2021b) (see chapter 2).

The algorithms described in the previous study used binary images acquired using simple photogrammetric applications (i.e., image blurring, image thresholding, edge detection). However, the use of simple image processing applications found to be not reliable and, in some cases, unusable (i.e., for large variations on illumination and/or colour (Fig. 3.13: c, e, and g)). Furthermore, the process requires to adjust the parameters of each image-processing function manually. The use of CNN, for the feature detection of masonry micro-geometry (i.e., geometry of individual masonry units and mortar), improves the results of the feature extraction by providing a better binarized output and automating the procedure (Fig. 3.13: d, f, and h).

For both applications (measurement of cracks and generation of geometrical model) the image is required to be either an orthorectified photo or an image captured vertically compared to the masonry element. This will ensure that the detected elements (i.e., blocks, cracks, openings) will have the same scale along the image used.

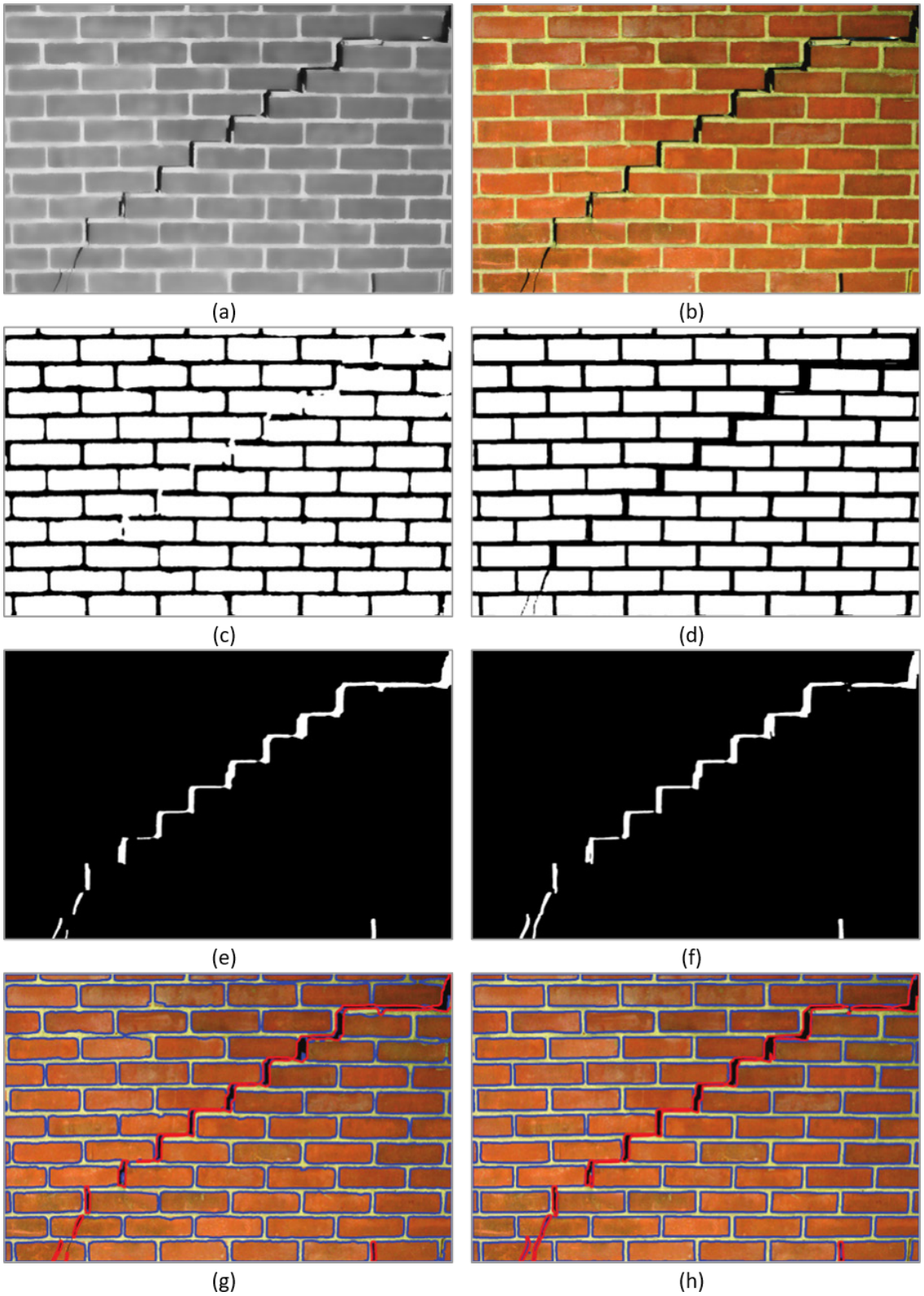


Fig. 3.13: Comparison of Thresholding and CNN output; a) Blurred/grey image for thresholding; b) Original image for CNN; c) Detected blocks using thresholding; d) Detected blocks using CNN; e) Detected cracks using thresholding; f) Detected cracks using CNN; g) Perimeter of detected elements using thresholding; h) Perimeter of detected elements using CNN.

Moreover, the simplified shapes acquired, using the binarized output from the trained models, demonstrated great improvement when compared to binarized images acquired using image-processing. The blocks were more evenly shaped and better aligned to the actual masonry bricks (Fig. 3.14). This did not only improve the reliability of the numerical analysis, but also the geometric representation of the structure when used for representation in CAD environments. It should be noted that the original image used for the Fig. 3.13 and Fig. 3.14, was the same used for the generation of the numerical model in the previous study for comparison purposes. Additionally, the use of simple image-processing applications, for the feature detection, favours the specified image since the contrast between mortar and bricks is highly visible, without large changes to illumination/colour. For general use, the difference between the resultant output is expected to be larger.

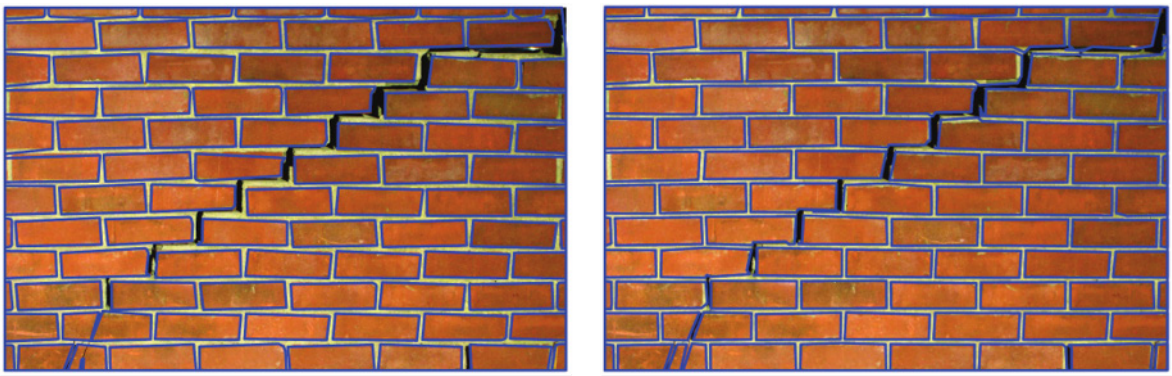


Fig. 3.14: Extracted blocks using the methodology described in Loverdos et al., (2021)*; a) Block-detection using image-thresholding; b) Block-detection using CNN.*

3.9. Shape Quality

The quality of the segmentation was evaluated to quantify the change between the ground truth and the output from either the CNN model or simple image-processing applications. The simple metrics included the accuracy, recall, precision, and F1-Score, as seen in the use of the CNN model.

However, those metrics tend to check the overall quality of the output. Since the model was aimed to be used for the generation of geometrical models and documentation, additional metrics were included to quantify the quality of the block-shapes. The shape quality was estimated by calculating the coverage, error of area, and quantity of undefined blocks (Fig. 3.15). Each segmentation of the ground-truth was compared with the output of either CNN or image-processing to identify the same object in both images. The first step was to calculate the common area between the two objects (Eq. (36)). The coverage (Eq. (37)) was calculated by comparing the common-area between both objects (ground-truth and output) while the Area-Error was calculated by comparing the area between both objects (Eq. (38)). The quantity of undefined objects was the number of objects that didn't match with a segmentation from the ground truth following certain conditions (i.e., the coverage must be similar to the area of the segmentation). The equations of the additional metrics are provided below:

$$Common = Object1_i \cap Object2_i \quad (36)$$

$$Coverage1 = Common/Area1 \quad (37)$$

$$Coverage2 = Common/Area2$$

$$Area\ Error = Area2/Area1 - 1 \quad (38)$$

$$Missing\ Error\ = \ Missing/AllObjects \quad (39)$$

, where $Object1_i$ denotes any segmentation on ground truth, $Object2_i$ any segmentation on the output (CNN or Image-processing), $Area1$ the total area of the $Object1$ in pixels, and $Area2$ the total area of $Object2$ in pixels. Individual segmentations were obtained using watershed-segmentation with the binary image as mask. The conditions for segmentation, i.e., the same object as in the ground-truth, were:

Condition1 (Required): $Common > 0$ (40)

Condition2 (Optional): $1 - Coverage1 \leq Threshold1$ (41)

Condition3 (Optional): $1 - Coverage2 \leq Threshold2$ (42)

, where the $Threshold$ is any value between 0 to 1 and denotes the difference between the common area and the total area of the segmentation. The shape-analysis in this case used a Threshold value of 0.2 (i.e., 80% of object area must be common). The first condition was used to verify that two objects have a common area (Eq. (40)). By using the second condition only (Eq. (41)), the evaluation considered objects that were erroneously merged (Fig. 3.15: c), which increased the area-error significantly. By using the third condition only (Eq. (42)), the evaluation considered segmentations that were erroneously broken into smaller elements. If multiple objects satisfy the conditions, then they were all included in the final common area. For this study, both optional conditions were used. Thus, for an object to be considered, must have a common area of at least 80% of both segmentations (ground-truth and output). Fig. 3.15 shows the undefined objects that did not match both images (Fig. 3.15: a-b & Fig. 3.15: a-c). Furthermore, it demonstrates that the CNN output has marginally fewer undefined objects, since image-processing is prone to noise caused by the change in illumination and colour within the same image. Although watershed-segmentation can close open-segmentations (Fig. 3.13: g), it is not always feasible. In Fig. 3.16, images were utilised during the validation phase of the model and are shown to compare the output acquired using image-processing. The method applied was the same as the one used to acquire the image in Fig. 3.13:c. The metrics for the complete evaluation of all images (Fig. 3.15 & Fig. 3.16), are shown in Table 3.7.

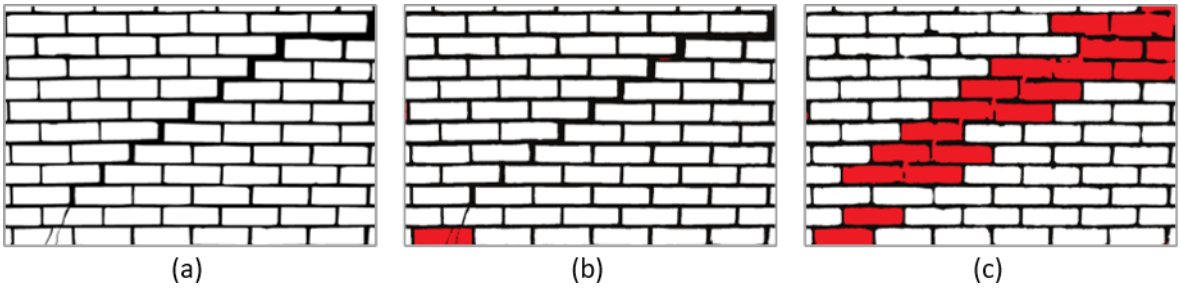


Fig. 3.15: Comparing segmentation quality of CNN and Thresholding (Red: Unidentified blocks with large change to location/area): a) Ground-truth image (Annotated); b) CNN-Output compared with annotated; c: Thresholding-output compared with annotated.

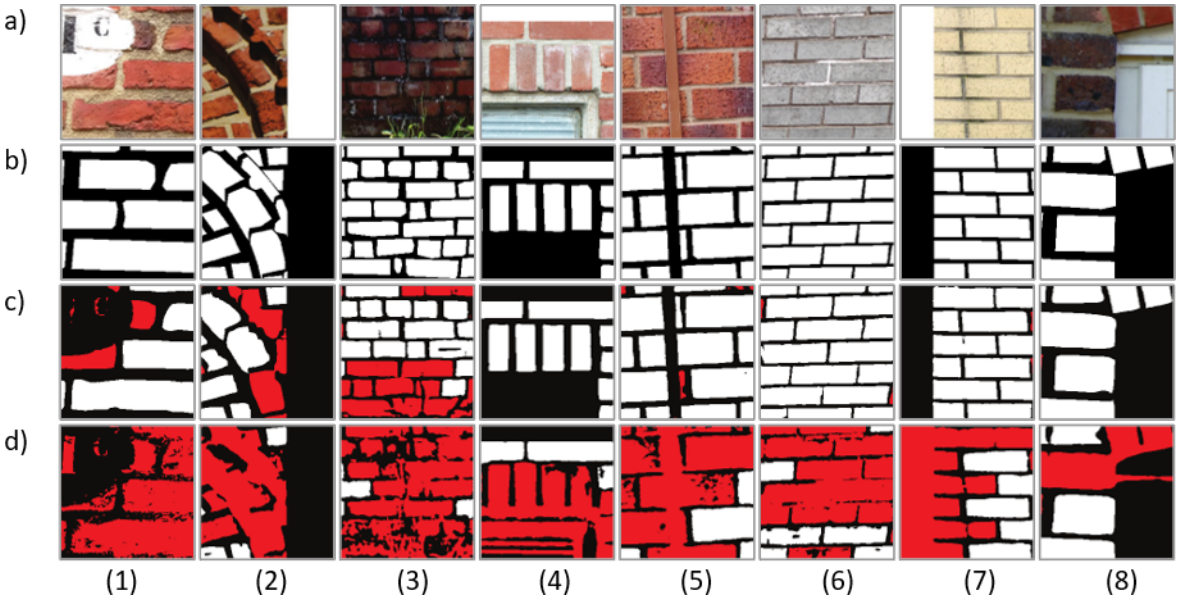


Fig. 3.16: Comparing segmentation quality of the train/validation set (Red: Unidentified blocks with large change to location/area): a) Original image; b) Ground-truth image (Annotated); c) CNN-Output compared with annotated; d) Thresholding-output compared with annotated.

Table 3.7: Metrics to quantify the segmentation quality of the output.

Name	Image1 (Gr. Truth)	Image2 (Output)	Acc.	Recall	Precision	F1 Score	Coverage (1)	Area Error	Missing Error (1)
-	-	-	%	%	%	%	%	%	%
Fig. 15: b	Annotated	CNN	95.68	94.63	99.78	97.13	94.4	-5.39	5.5
Fig. 15: c	Annotated	Thresh.	94.76	96.68	96.55	96.62	96.7	-1.64	27.47
Fig. 16: 1c	Annotated	CNN	83.63	75.24	97.65	84.99	97.31	0.76	44.44
Fig. 16: 1d	Annotated	Thresh.	66.67	68.08	75.44	71.57	0	n.a.	100
Fig. 16: 2c	Annotated	CNN	94.65	92.43	94.48	93.44	93.95	-2.11	50
Fig. 16: 2d	Annotated	Thresh.	83.24	87.17	75.82	81.1	92.03	-4.8	75
Fig. 16 :3c	Annotated	CNN	89.35	93.5	91.47	92.47	96.2	3.63	50
Fig. 16: 3d	Annotated	Thresh.	70.97	76.69	80.83	78.71	94.27	3.58	87.5
Fig. 16: 4c	Annotated	CNN	98.25	98.56	96.93	97.74	98.42	1.76	9.09
Fig. 16: 4d	Annotated	Thresh.	70.7	95.07	57.08	71.33	95.46	12.58	72.73
Fig. 16 :5c	Annotated	CNN	96.69	97.12	98.13	97.62	96.24	-1.66	16.67
Fig. 16: 5d	Annotated	Thresh.	78.28	91.85	80.03	85.53	96.99	0.81	77.78
Fig. 16: 6c	Annotated	CNN	96.29	97.63	97.77	97.7	97.91	0.28	10
Fig. 16: 6d	Annotated	Thresh.	87.8	92.68	92.21	92.45	90.35	-7.01	70
Fig. 16: 7c	Annotated	CNN	97.59	98.35	97.63	97.99	98.53	0.92	7.69
Fig. 16: 7d	Annotated	Thresh.	69.58	92.77	67.98	78.47	95.14	-2.07	61.54
Fig. 16: 8c	Annotated	CNN	98.22	98.17	97.99	98.08	97.21	-0.97	12.5
Fig. 16: 8d	Annotated	Thresh.	92.14	98.58	86.42	92.1	95.68	8.01	50
		Median CNN:	94.48	93.96	96.87	95.24	96.69	-0.31	22.88
		Median Thresh:	79.35	88.84	79.15	83.1	94.58	0.29	69.11

Most metrics provide similar values for both test cases of the damaged wall (Fig. 3.15). More specifically the accuracy on the CNN output was slightly higher, which explains the better representation of the segmentation (95.7% vs 94.8%). Although, the coverage in the CNN image was slightly lower for the objects that were detected correctly (94.4% vs 96.7%). Nonetheless, the missing error of the CNN image was much lower than the thresholding case (5.5% vs 27.5%), which was caused by the presence of multiple open shapes in the thresholding image (Fig. 3.15: c). On the CNN case, the bottom-left block was undefined due to the damage not separating the blocks completely (Fig. 3.15: b), as it is on the ground truth image (Fig. 3.15: a) and detecting the three broken elements as a single object. Also, it should be noted that the large value of the undefined blocks, in the thresholding case, would decrease the coverage and increase the area error, if they were allowed in the evaluation (Fig. 3.15: c). Thus, a higher coverage does not correspond to an overall better quality of segmentation, since it relates to fewer elements. Furthermore, the image was favourable for block detection using thresholding, due to minimal noise, which explains the better fit of the validated objects. The results will vary depending on alterations to illumination and colour (Fig. 3.16).

The metrics acquired for Fig. 3.16 demonstrate that the quality of shapes when using simple thresholding was detrimental for the accuracy of the model. This can be observed initially from the median accuracy, which was equal to 94.5% vs 79.3%, for the CNN and thresholding methods respectively. In general, thresholding provided marginally fewer validated blocks, as it was observed by the median missing error, which was equal to 22.9% vs 69.1%, for the CNN and thresholding methods respectively (Fig. 3.16: c, d). Furthermore, the overall fitting of the shapes was higher in the CNN case since the coverage calculated was 96.7% vs 94.6% for the CNN and thresholding methods respectively.

Moreover, simple image-processing methods can not recognise openings, damage, or background (Fig. 3.16: d4 & Fig. 16: d8). They are only able to identify either edges (edge detection) or pixel intensity (thresholding). Every image that contained locations of background, required modification before it was used. Thus, the use of CNN for the object detection was preferred for the current test-case, due to its higher accuracy and reliability to identify correctly almost every object (Fig. 3.15: b), except for highly complex images (Fig. 3.16: c1, c2, c3).

3.10. Conclusions

This research is contributing towards automating procedures that engineers would require considerable amounts of effort, expertise, and time to achieve while at the same time minimises the human error. For example, remote inspection is even more challenging for difficult to reach locations, where visual inspection may be required, but is challenging to implement. The study demonstrates that both damage and block detection can be achieved with adequately high accuracy by utilising deep learning approaches. Where the block-detection model achieved a validation-accuracy of 96.86% and the crack detection model an F1-Score of 79.6% (which can be improved by simply providing more annotated data). The quality of the binarized output has also been assessed showing that the CNN output outperforms simple image-processing functions even for clean images. Especially considering that simple image-processing applications do not differentiate between detected elements and background/openings. Additionally, deep learning methods allow for the improvement of the model by increasing the dataset used for training and validation. Consequently, the performance of the model can always be enhanced by acquiring additional samples of the classified elements.

The main limitation of the demonstrated application of deep learning, for the detection of features in masonry structures, is that a similar sample should be provided during training of the model to detect specific features. I.E., to be able to reliably identify irregular masonry units, images with irregular masonry should be included in the dataset. Furthermore, features not shown in the image will not be identified by the model (i.e., cracks of extremely small size). Thus, the engineer must ensure that the desired features should be visible on the image-slice passed through the network. Lastly, the use of orthorectified images is important for the accurate evaluation of detected features (i.e., if used for numerical modelling, or crack measurements).

Future work includes the implementation of the developed models to a framework that will be able to automatically generate numerical models and analyse changes on the structure in real time. This includes a detailed report of the measurement of detected defects on the structure. Currently only cracks are detected but additional classifications of defects are considered, such as spalling, vegetation, and discolouration. Additionally, the results obtained from the block/crack-detection can be coupled directly with algorithms for numerical modelling to automatically evaluate the crack patterns on the structure by performing numerical analysis or compare the results from inverse analysis by matching the outputs (from the block/crack detection models and evaluation method used, i.e., (Alessandri *et al.*, 2015; Iannuzzo *et al.*, 2018; Angelillo, 2019; Napolitano and Glisic, 2019; Tiberti *et al.*, 2020)). In all cases, the capture procedure can be replaced by remote sensing applications, such as drones, to remotely capture image/video data paired with semantic segmentation for the identification of structural elements. Hence, providing a digital twin of the structure considered for real time monitoring.

Acknowledgements

Several photos obtained by engineers from Network Rail and Helifix, UK, and were kindly offered to expand our masonry dataset. Therefore, their support in this study is highly recognised and appreciated. Dimitris Dais, Prof Ihsan Bal, and Dr Eleni Smyrou are acknowledged for their insightful comments on the implementation of the deep learning networks for crack detection and for the crack detection algorithm (github.com/dimitrisdais/crack_detection_CNN_masonry). This work was funded

by the EPSRC project “Exploiting the resilience of masonry arch bridge infrastructure: a 3D multi-level modelling framework” (ref. EP/T001348/1). The financial contribution is very much appreciated.

4. Paper #3: Geometrical digital twins of masonry structures for documentation and structural assessment using machine learning.

Dimitrios Loverdos[a], Vasilis Sarhosis[a*]

[a] *University of Leeds, School of Civil Engineering, Woodhouse Ln, Leeds LS2 9DY, United Kingdom*

Abstract

The generation of numerical models for masonry structures is a time-costly procedure since it requires the discretization of a large quantity of smaller particles. Similarly, traditional visual inspection involves the cautious consideration of each element on a masonry construction. In both cases, each brick element needs to be considered individually. The work presented in this document intends to address the challenges associated with documenting individual masonry units on a structure by employing computer vision and convolutional neural networks (CNN). Specifically, it introduces a dynamic workflow that automatically detects masonry units and cracks, which are then utilized to create a geometric digital twin of masonry structures. The outcome is a collection of space coordinates and geometrical objects that represent the masonry entity and allow the comprehension of the object for documentation and structural assessment. This interoperability between architectural, structural, and structural analysis models paves the way to use engineering to create a smarter, safer, and more sustainable future for our existing infrastructures.

Keywords: masonry, image processing, watershed transform segmentation, feature extraction, structural analysis, documentation.

***Corresponding author:** Dr. V. Sarhosis, University of Leeds, email: v.sarhosis@leeds.ac.uk

4.1. Introduction

Masonry is a commonly used construction which consists of placing individual masonry units (e.g., brick, concrete blocks, stones etc.) on top of each other and binding them together with mortar (i.e. a mixture of sand, binder such as cement or lime and water). A large portion of infrastructure that we are currently using is made of masonry. For example, masonry structures such as arch bridges and viaducts are typically over 100 years old. Over the years, these structures have attained damages (e.g. cracking, spalling) and changes to their geometry due to deterioration, subsidence, change of usage, which resulted in altering their load carrying capacity (McKibbins *et al.*, 2006)). Failure of such infrastructure could lead to significant direct and indirect costs to the economy and society and could hamper rescue and recovery efforts. Without a strategic approach to caring for our ageing masonry infrastructure, we run the risk of over-investing in some areas while neglecting others that need our attention, or risk failing to address economic and societal needs. Therefore, it is important to develop accurate and data driven management plans for these structures to retain their usefulness and extend their lifespan.

An essential element of infrastructure maintenance is structural inspection which must be conducted systematically and not only when there is a breakdown or failure (Sowden, 1990). The prime focus of structural inspection is public safety and prolonging the economic life of the structure. In addition, inspection should keep disruption to the users and third parties of the structure to a minimum. A typical inspection process gathers information from a structure with respect to defects and can record deterioration over time (Eaton, Edwards and Crapper, 2014). Traditional methods for structural inspection and surveying, of our cultural-heritage, rely on visual inspection and manual recording. However, manual recording of the structural condition is subjective, increases the risk of erroneous assessment, is time-consuming, and comes with large monetary-costs (Phares *et al.*, 2004).

Another important element of the masonry management plan is structural assessment, which aims to assess the structural capacity of a structure during its service life. In practice, structural assessments are usually conducted using traditional and standardised methods, despite knowledge that these methods often provide conservative estimates. In particular, over the last decades, there have been a significant effort to use advanced numerical methods of analysis and high-fidelity models to understand the performance of masonry infrastructure to different loading conditions (Lourenço, 1996, 2013; D'Altri *et al.*, 2020). Such methods range from macro-models, which consider the masonry as a singular element (continuum macro-models) or multiple elements not corresponding to the masonry pattern (discontinuum macro-modelling, (Caliò, Marletta and Pantò, 2012)) to micro-models that consider masonry as an assemblage of blocks and mortar joints (Asteris *et al.*, 2015). Depending on the accuracy required, a discrete model (micro-model) can also consider the inclusion of the mortar as a physical element (detailed micro-modelling, (D'Altri *et al.*, 2018; Sarhosis and Lemos, 2018)). In which case the analysis may provide more accurate results, regarding crack propagation on mortar, at the expense of computational effort. Thus, detailed micro-modelling is usually limited to small and simple structures. An alternative is the simplification of the model to consider the mortar as a zero-thickness interface (simplified micro-modelling, (Sarhosis and Sheng, 2014; Sarhosis, Garrity and Sheng, 2015; Sarhosis, Forgács and Lemos, 2019)). In which case, the numerical model contains much fewer contact points, which decreases computational effort dramatically, while still retaining the accuracy of a discrete model. Moreover, modelling strategies can extend from 2D to 3D space. A 2d model is limited to a single plane but provides a quick assessment of the structural condition (Sarhosis and Sheng, 2014; Sarhosis and Lemos, 2018; Sarhosis, Forgács and Lemos, 2019; Segura *et al.*, 2021). Alternatively, a 3d model can provide a more accurate and complete picture of the state of the structural-condition since it includes the physical interaction with all elements of the structure (Sarhosis *et al.*, 2014; Forgács, Sarhosis and Bagi, 2017, 2018; D'Altri *et al.*, 2018; Erdogmus *et al.*, 2019, 2020). However, 3d modelling comes with an enormous increase in computational time required to complete an analysis.

Although typical modelling strategies rely on idealised geometry, research in the literature demonstrated that higher precision of geometry, for both size and shape, leads to distinct and potentially more accurate results (Erdogmus *et al.*, 2019; Ferrante *et al.*, 2021; Kassotakis *et al.*, 2021). However, the accurate definition of the masonry geometry is usually disregarded for simpler modelling techniques due to the effort required to produce such models. Concerning the geometry of masonry structures (typically complex with irregular shapes), drawings about the original design may not exist. Even if drawings do exist, might not actually represent the final construction aspect due to damage and permanent deformations that might have happened during its service life. Therefore, obtaining accurate geometric conditions of the structure and defects existing in them is of vital importance for the accuracy and reliability of such structural assessment models.

Over the last decade, advances in photogrammetry and laser-scanning made it feasible to generate three dimensional records of the masonry geometry rapidly and accurately. Photogrammetry (Altuntas, Hezer and Kırlı, 2017; Historic England, 2017; Napolitano and Glisic, 2019) and laser scanning (Sithole, 2008; Historic England, 2018; Valero, Bosché and Forster, 2018; Valero *et al.*, 2019) is often used to produce the 3d-point cloud data of masonry structures. Although useful for visual inspection and documentation, points cloud data require further post-processing if are to be used for the geometric definition of the structure. Some photogrammetric and laser scanning applications allow the generation of continuum macro-models in 3-dimensional space, but their accuracy is limited to a single object (Altuntas, Hezer and Kırlı, 2017; Bassier *et al.*, 2019; Kalfarisi, Wu and Soh, 2020). Point-based voxelisation of 3d point cloud data offers a solution by producing discontinuum macro-models of structures (Hinks *et al.*, 2013), which allows the use of discrete analysis, but does not consider the effect of the masonry pattern or defects.

Computer vision applications have also been used to generate BIM models (Building Information Modelling, (Volk, Stengel and Schultmann, 2014; Kassotakis and Sarhosis, 2021)), to assist with management of structural assets. Generation of BIM models is nowadays part of the development process of modern structures using specialized software (i.e., AutoCAD, Revit). Though, older constructions do not benefit from modern approaches to design/management of assets. However, PCD (point cloud data) acquired from photogrammetry or LiDAR can be used to generate HBIM (historical-BIM) models for existing structures (Andriasyan *et al.*, 2020). Moreover, post-processing of the PCD can be used to automate the classification and reconstruction of structural elements on the BIM model (Bassier and Vergauwen, 2020). Furthermore, certain approaches are using the PCD to generate BIM models, which are then converted to numerical models for analysis (Barazzetti *et al.*, 2015; Rolin *et al.*, 2019). Nonetheless, such methods are still limited to a macro-model approach.

Image-processing is a powerful tool, which can be used to create detailed geometric records of the masonry micro-geometry with minimal effort using feature detection (Canny, 1986; Martin, Fowlkes and Malik, 2003; Arbeláez *et al.*, 2011; Bora, 2017) and segmentation algorithms (Beucher, S.; Meyer, 1993; Arbeláez *et al.*, 2011; Kornilov and Safonov, 2018). Feature detection and segmentation has already been applied to identify the location of brick units from an image (Oses, Dornaika and Moujahid, 2014; Cluni *et al.*, 2015; Brackenbury and Dejong, 2018) or 3D point cloud data (Sithole, 2008; Valero, Bosché and Forster, 2018). However, in most cases their use proves challenging due to digital noise caused by the change in illumination, colour, and texture presented within the digital data. The research presented in (Valero, Bosché and Forster, 2018), showcased an efficient methodology to remove noise from the inner-region of detected masonry blocks. However, the procedure followed in their research is limited to concave shapes only.

Now, machine learning (M.L.) applications have already seen use in structural engineering research due to the immerse potential it has to assist with visual inspection and monitoring applications (Spencer, Hoskere and Narazaki, 2019), especially in defect detection since it provides the means to identify, locate, and asses detected damages on structural elements. For example, (Valero *et al.*, 2019) extracted statistical data from a 3D point-cloud and use them to train a M.L. algorithm for the

detection and classification of chromatic and geometric defects on ashlar masonry. Most typical applications of defect detection include the use of D.L. due its architecture, which allows the detection of complex features on unstructured data. For example, (Brackenbury, Brilakis and Dejong, 2019) proposed the use of GoogleNet-Inception-V3 algorithm for the classification and segmentation or mortar and defects of masonry elements, including the use transfer-learning. Each defect was classified separately as cracking, spalling, or vegetation. Moreove, (Kalfarisi, Wu and Soh, 2020) used Mask-RCNN and FRCNN-FED to detect bounding boxes that contain cracks on structures and performed pixel-wise segmentation within the detected areas. Furthermore, they transferred the segmented locations to a 3D reality-mesh object which was generated using photogrammetry. Recently, (Dais *et al.*, 2021) tested different state-of-the-art CNN algorithms for the detection of cracks on masonry images. They investigated both patch-classification (with 95.3% accuracy) and semantic-segmentation (with 79.6% f1-score) on pre-trained networks. In this way, cracks in masonry walls were accurately captured from images.

One could argue that the use of CNN algorithms for the detection of brick/mortar is also necessary to provide a complete composition of the micro-geometry of masonry structures. In (Ibrahim, Nagy and Benedek, 2019), the authors proposed the use of U-Net for the segmentation of mortar of different types of masonry (including rubble). Furthermore, they also used watershed-transform for the isolation of each brick unit and extraction of its perimeter. Also, (Ergün Hatir and İnce, 2021) proposed the use of Mask-R-CNN for the classification and segmentation of masonry stones in ancient buildings. Each stone detected was classified to a different lithology based on their detected features (i.e., colour, texture).

The purpose of this paper is to introduce the creation of an automated workflow for generating a comprehensive geometric digital twin, facilitating the documentation and structural assessment of masonry infrastructure through image-based data. In particular, the geometric digital twin of masonry infrastructure will include the location of masonry units and mortar as well as metrics of detected cracks in it. The detection of masonry units and cracks in the image was acquired using CNN (Dais *et al.*, 2021; Loverdos and Sarhosis, 2022a). Regarding the acquisition of metrics of cracks, the methodology has been completely overhauled to acquire the precise measurements and not an approximation, compared to previous and other studies (Cabaleiro *et al.*, 2017; Kalfarisi, Wu and Soh, 2020; Loverdos and Sarhosis, 2022a). In addition, a feature-detection and feature-extraction methodology is proposed, which is able to identify the background in images and non-masonry elements (i.e., window openings, concrete lintel etc). The process is fully automatic and overpasses the difficulties discussed in (Loverdos *et al.*, 2021a). Finally, the development of a novel methodology which allows common mesh processing functionalities of structural elements for the efficient structural analysis of masonry structures is proposed.

4.2. Workflow for the geometric digital twin of masonry structures

The complete algorithmic sequence includes multiple steps (Fig. 4.1) to derive the final models (e.g., geometrical digital twin and numerical model for the structural analysis of masonry structures). Each part of the workflow runs individually to allow the user to adjust the options provided, if required, without the need to re-apply previous packages.

The first step is the selection and scale of the input image (Section 4.3). Any image quality can be used, but the workflow is designed to work with images captured vertically to the examined plane (step 1.1). Then, the input is scaled programmatically, to allow the generation of the scaled-geometry and precise evaluation of the metrics of cracks (Step 1.2, Figure 1).

The second step is the feature-detection to acquire the binary-images of the micro-geometry of masonry (i.e., blocks, cracks, background; see Section 4.4-4.7). The location of the blocks and cracks is identified using CNN (steps 2.1-2.2, Figure 1). The remaining elements (i.e., background, masonry) are

derived from the source-image and the CNN outputs (steps 2.3-2.4, Fig. 4.1). The binary-images acquired from the feature-detection are combined to form the complete-structure (step 2.5, Fig. 4.1), which is used during the feature-extraction to generate the geometrical model. Each individual output of the feature-detection is post-processed separately (not shown in the workflow chart; applied on steps 2.1-2.5, Fig. 4.1), to improve the binary-image and subsequently the final-output (see Section 4.7). The post-processed images do not replace the initial-output, to allow its modification without generating the image again.

The third-step is the registration and measurement of the cracks (see Section 4.8). Those include the geometrical properties of each crack and their coverage regarding the overall masonry-area. It is worth mentioning that the step about the crack-measurements is not mandatory for the remaining steps of the workflow.

The fourth-step is the feature-extraction of the detected-features in the form of generalized-polylines (see Section 4.9-4.10). It is initially starting by creating the watershed to provide separated labels for each element (step 4.1). It follows with segmentation-adjustments to include mortar/damage to the geometry and improve the output (step 4.2). Finally, the geometry is extracted in the form of contours and is adjusted to reduce the number of vertices and improve the general shape of every detected-element (steps 4.3-4.4).

The final-step relates to the development of the geometric digital twin that can be exported in a CAD based environment, where every element is separated to different layers/groups (i.e., blocks, cracks, background). Also, geometric information stored in the CAD based environment can be used as input in a numerical model (e.g., UDEC) for the structural analysis of masonry infrastructure (see Section 4.11). The proposed method includes the generation of convex-mesh, optimized for numerical modelling to allow separation of blocks during the analysis and investigate crack propagation.

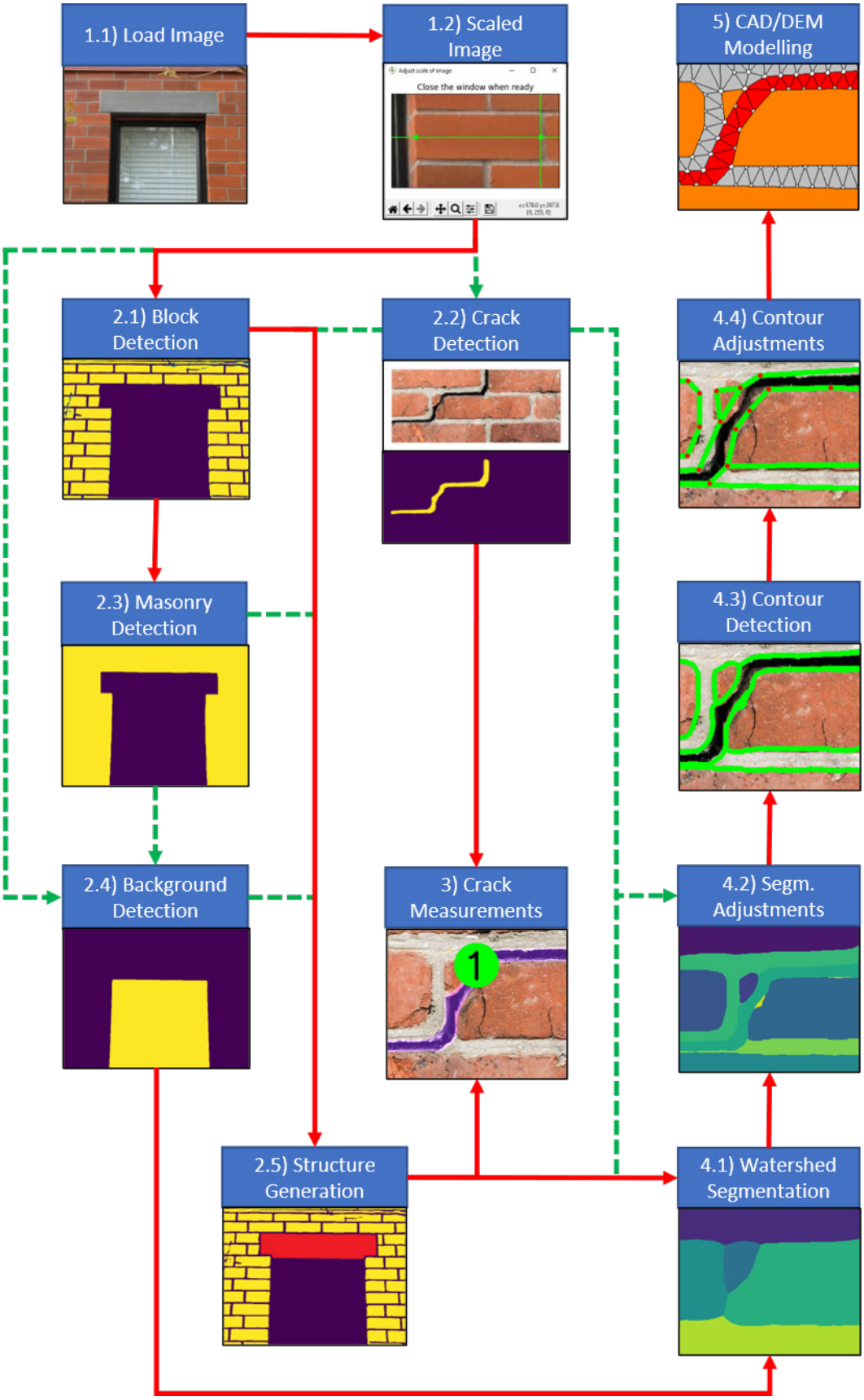


Fig. 4.1: Proposed workflow; The label-numbers denote the sequence-order, the red-lines denote mandatory-requirements, and the green-lines denote optional-requirements based on user input.

4.3. Image input and scaling

The first step is the image input and scaling. The optimal input for the specific use of the framework is an orthorectified image (Fig. 4.2). The orthorectified image should be adjusted to have equal ratio of distance per pixel along the X-Y axes. Additionally, the ortho-projection should eliminate optical distortion. Additionally, the background of an orthorectified-image can be a uniform colour, which simplifies the detection of background i.e., openings. However, simple images (i.e., captured from a smartphone or DSLR) can also be used, if the captured angle is vertical to the plane examined.

Fig. 4.2: Optimal input; a) 3D Model generated using the "RealityCapture" software; b) Orthorectified image.

Image scaling is evaluated programmatically to ensure that the geometry extracted will resemble the dimensions of the structure under consideration. This is necessary for a realistic documentation and structural assessment. Furthermore, the scale acquired from this section is used to automate multiple variables/options during execution (i.e., automate image-resizing before passing through the network). The image-scale is obtained by selecting two points on the image and providing the real distance between the two locations (Eq. (43)). Images can be plotted using the matplotlib python package, since it allows to acquire the location in a non-integer form and provides the coordinates and pixel value at the cursor location. Furthermore, a plotted cross at the mouse location ensures that the points selected are positioned completely vertically or horizontally.

$$ImgScale = Input(Real\ Units)/Distance(Pixels) \tag{43}$$

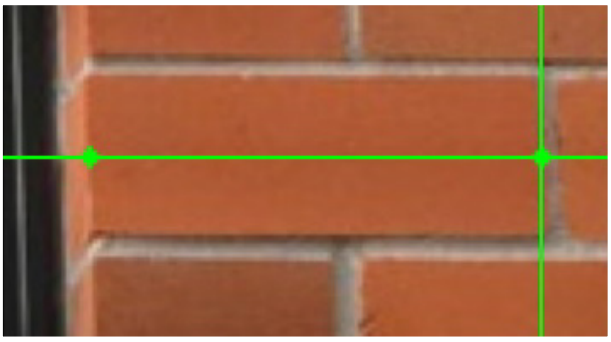


Fig. 4.3: Selection of points on image to evaluate the image scale.

Alternatively, if the image has been acquired using photogrammetric-software, the image-scale may have a pre-specified value. This is possible since the orthorectified-image may have a specified scale in the form of units per pixel.

4.4. Block and crack detection

Block and crack detection was applied using different CNN models for semantic segmentation. The block-detection model was developed in (Loverdos and Sarhosis, 2022a), while the crack-detection model in (Dais *et al.*, 2021). The model for the block detection was created using the DeepLabv3+ architecture with a modified Xception backbone (Chen *et al.*, 2018). The database includes 107 annotated images of typical masonry with mortar, where the positive values represent the block units. Each image was divided into sections of 224x224x3, which provided 2,814 image slices that were used to train the network with 25% validation data. The parameters of the architecture were adjusted as follows: 16 OS (feature-extractor output ratio), "*Xception*" backbone, "*Pascal-Voc*" pre-trained weights, and "*Sigmoid*" activation. The optimiser of the highest performance model was RMSProp (RMSP) with F1-Loss function (F1L). The best model was trained for 200 epochs with a batch size of 8, learning rate of 2E-4, and decay equal to 1E-6. The block-detection model achieved a 96.86% validation accuracy, 96.29% validation F1-score, 96.68% validation precision, and 95.94% validation recall.

The architecture of the crack-detection model is the U-Net with MobileNet as backbone. The database includes 351 photos of masonry walls with cracks and 118 without cracks. Each image was divided into sections of 224x224x3, resulting in 4,057 image slices used to train the network with 40% validation data. The optimiser of the highest performance model was Adam using Weighted Cross Entropy (WCE). The model was trained over 100 epochs with a batch size of 4 and a constant learning rate of 5E-4. The crack-detection model achieved a validation F1-score of 79.6%, validation precision of 79.9%, and validation recall of 81.4%.

Fig. 4.4: Sample images of the database used for the training/validation of the block detection model.

The methodology developed for the acquisition of the CNN-output was similar to the one presented in ((Loverdos and Sarhosis, 2022); Fig. 4.5). However, the methodology here has been modified to include the dynamic-resizing of the input-image based on the image-scale (Eq. (46)), which automates the process.

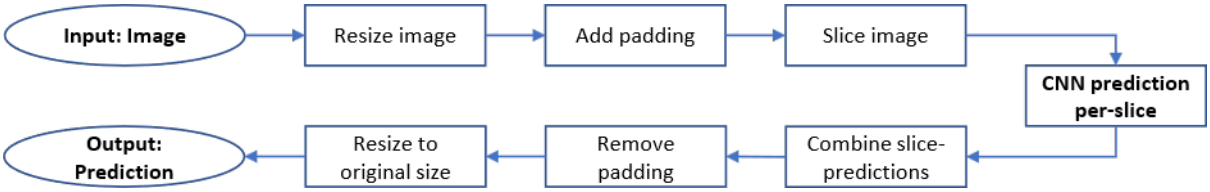


Fig. 4.5: Workflow: Acquiring the complete CNN-output.

Initially, the image was resized (Eq. (47)) to the nearest upper/lower limit (Eq. (46)), to normalize block-resolution and improve detection (i.e., avoid slices with excessive number of blocks). A white padding was added to the resized-image with respect to the slice size and overlap value, to retain the original aspect ratio (Eq. (48)). The input was passed though the network in parts (slices) to improve the resolution of the output (i.e., 224x224x3 pixels). Each slice included an overlap (i.e., 50 pixels) with the aim to improve connectivity (i.e., retain only the inner-region of 124x124 pixels). Finally, all slices were combined to a single image, removing the added padding, and resized to the original size.

$$Norm_{img} = \sqrt{x_{img} * y_{img}} \quad (44)$$

$$Norm_{slice} = \sqrt{x_{slice} * y_{slice}}$$

$$\text{Case \#1 (Manual Resizing): } Limit_i = F_i * Norm_{slice} \quad (45)$$

$$\text{Used factors (adjustable): } F_1 = 2 \;\&\; F_2 = 4$$

$$\text{Case \#2 (Dynamic Resizing): } Limit_i = F_i * Norm_{img} * ImgScale \quad (46)$$

$$\text{Used factors (adjustable): } F_1 = 200 \;\&\; F_2 = 600$$

$$NormScale = Limit_i / Norm_{img} \quad (47)$$

$$Pad_i = Ceil(ImgDim_i / InnerDim_i) * InnerDim_i + Overlap * 2 \quad (48)$$

$$Pad1 = Round(Pad_i/2) => Pad2 = Pad_i - Pad1$$

$$\text{Used factors (Adjustable): } Overlap = 50\ pixels$$

, where the $NormScale$ was used to adjust the size of the original-image and retain the aspect-ratio. The $ImgDim_i/InnerDim_i$ were the dimensions of the image and the inner-region (the slice-size minus the overlap) of the slice respectively, calculated for every axis individually. Pad_i is the total-padding of either axis, $Pad1$ is the left/top padding and $Pad2$ is the right/bottom padding. The final-result is a grey-scale image, where the brighter regions indicate the location of either the detected blocks or cracks. The final-output was further post-processed to acquire the binarization and improve the result, before creating the final structure (see Section 4.7). Furthermore, the factors in Case #2 (dynamic-resizing) resemble the units per pixel (Eq. (46)). Thus, a slice size of 224x224 pixels and F_i of 200, specifies a window of 1.12x1.12 meters² ($1.12 = 224/200$), when the $ImgScale$ is provided in meters (Fig. 4.3: b). This ensures that every slice has adequate size and will contain several blocks.

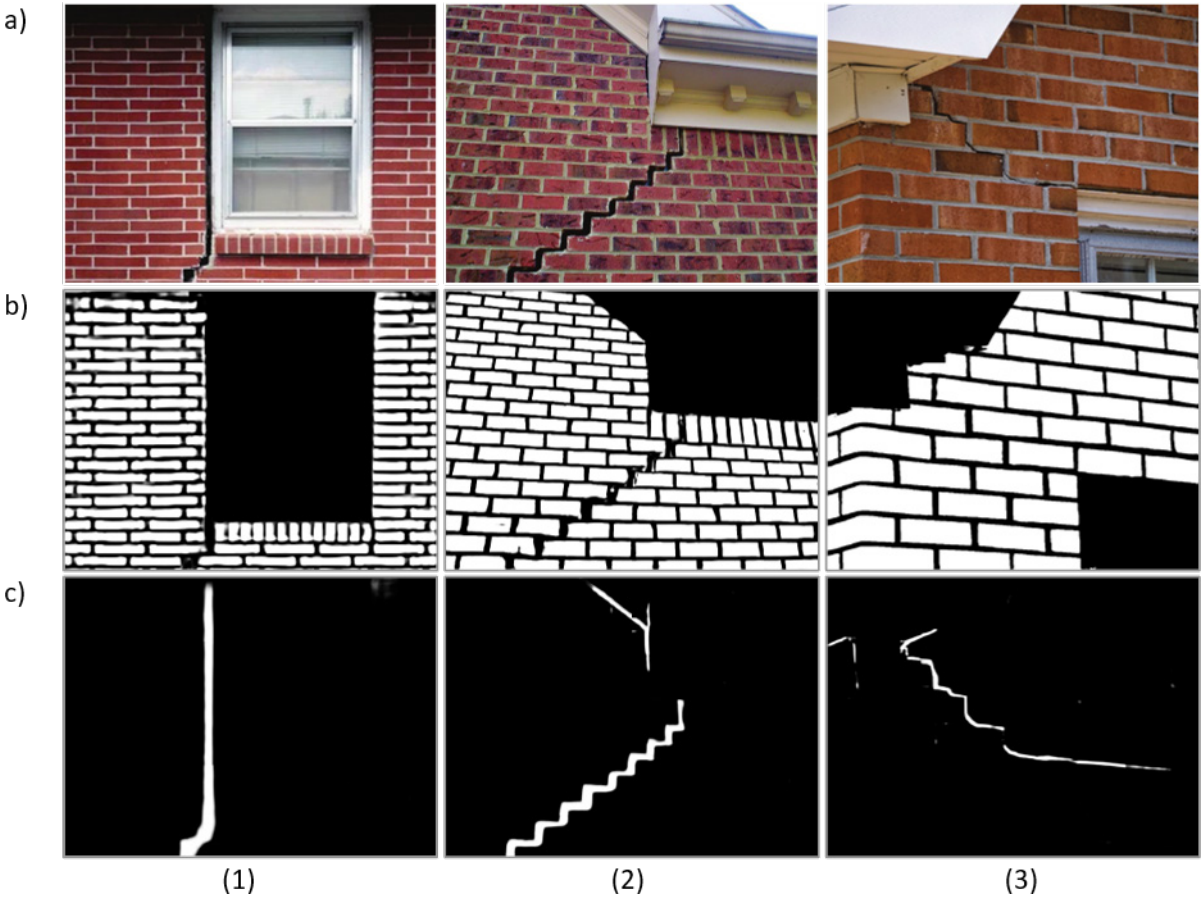


Fig. 4.6: Complete CNN output (after combining all slices); a) Original image; b) Block-detection; c) Crack-detection.

4.5. Definition of masonry and background

The last part to define a masonry structure is the definition of the background and openings located at the structure. The background and openings should be removed from the geometrical model to ensure that the correct domain is considered in the numerical analysis. Optimally, a separate model could be trained to remove the background and openings from the final segmentation. However, the dataset for the background was not sufficiently large to train a new FCN-model. Thus, two different methods were proposed to remove the background/openings from masonry images (Fig. 4.7).

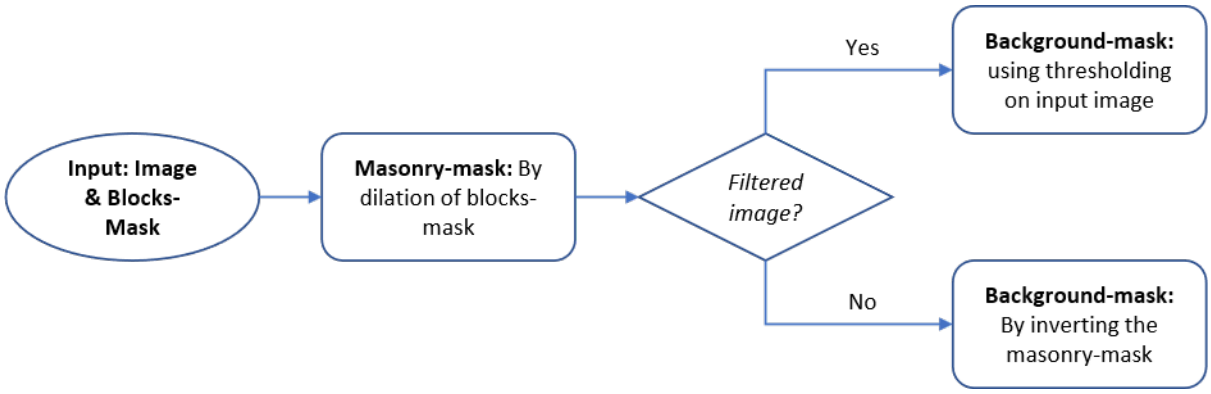


Fig. 4.7:Workflow: Creating the masonry and background masks.

The complete methodology to acquire the masonry and background-mask is provided below:

1. Masonry-Mask (combined-blocks): Dilation followed by erosion (image-opening) of the block-detection output to combine all blocks (Fig. 4.8: c). The values used are provided in Eq. (49).
2. Background-Mask (choose one option):
   2.1. No background: Empty array of zero values as background.
   2.2. Unfiltered-image: Inversion of the masonry-mask as background (Fig. 4.8: d).
   2.3. Filtered-image (white background): Threshold the original image to separate the structure from the background (Eq. (50), Fig. 4.9: b).

$$\text{Masonry-Mask: } Iters = F * MTh/ImgScale,\ KSize = 3x3 \quad (49)$$

$$\text{Used factors (adjustable): } F = 2$$

$$\text{Filtered-image threshold (adjustable): } Thld = 0.95 = 242.25/255 \quad (50)$$

Where $Iters$ is the iterations of the dilation/erosion, MTh is the mortar-thickness in the preferred units (i.e., in meters), $KSize$ is the size of the rectangular kernel used for erosion/dilation. Additionally, $Thld$ is the binary threshold used to create the background mask. Furthermore, for most cases the mortar-thickness is assumed to be equal to 10mm. Additionally, the multiplication of $2 * MTh$ in Eq. (49), effectively closes mortar-openings equal to 4 times the average mortar-width (i.e., 40mm). This ensures that larger gaps are also covered (Fig. 4.8: c). This step exports both the masonry (combined-blocks, Fig. 4.8: c) and background (Fig. 4.8: d, Fig. 4.9: b). Both images are post-processed before generating the final structure (see Section 4.7).

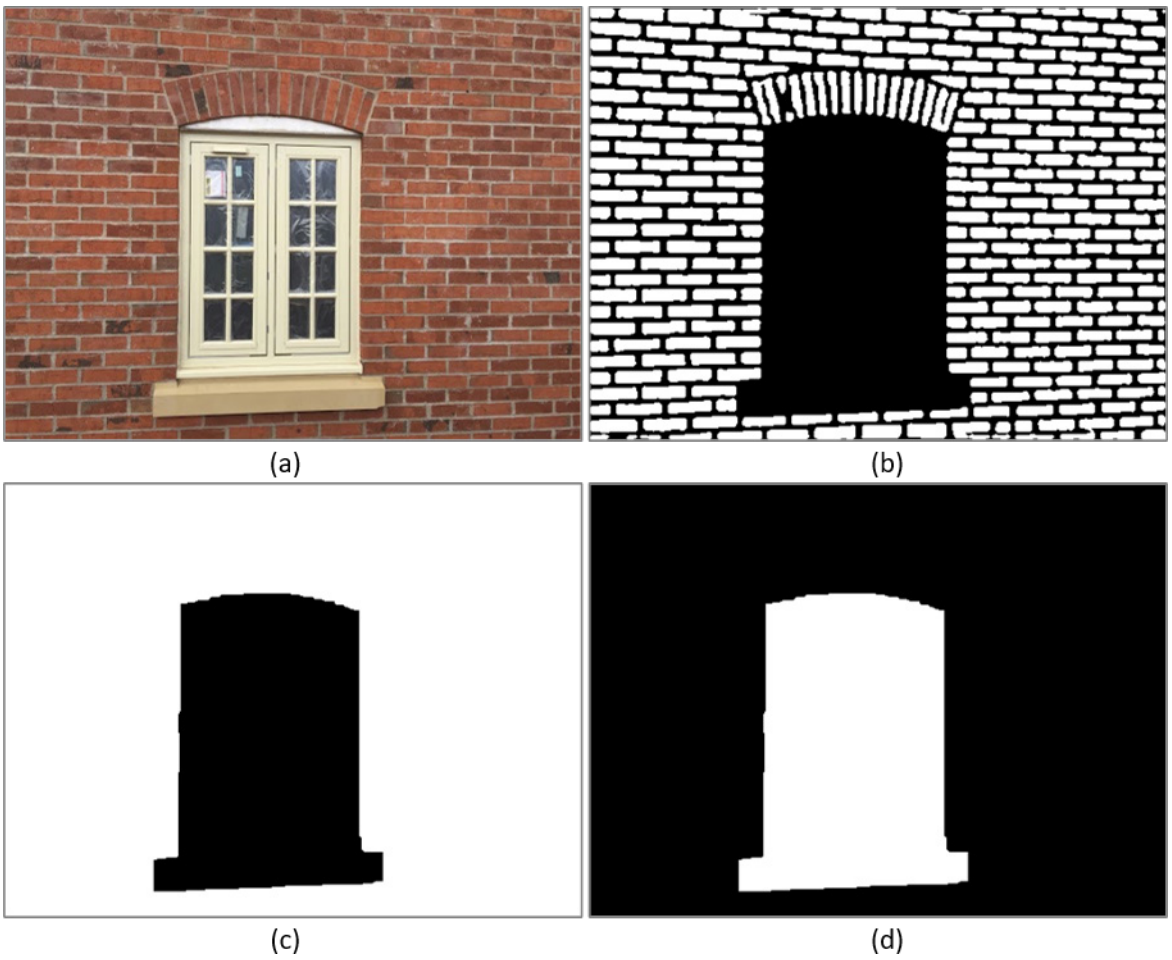


Fig. 4.8: Generating the background for unfiltered images; a) Original image; b) Output from the FCN-model (block detection); c) Masonry (combined-blocks) after dilation/erosion of blocks; d) Background after inversion of masonry.

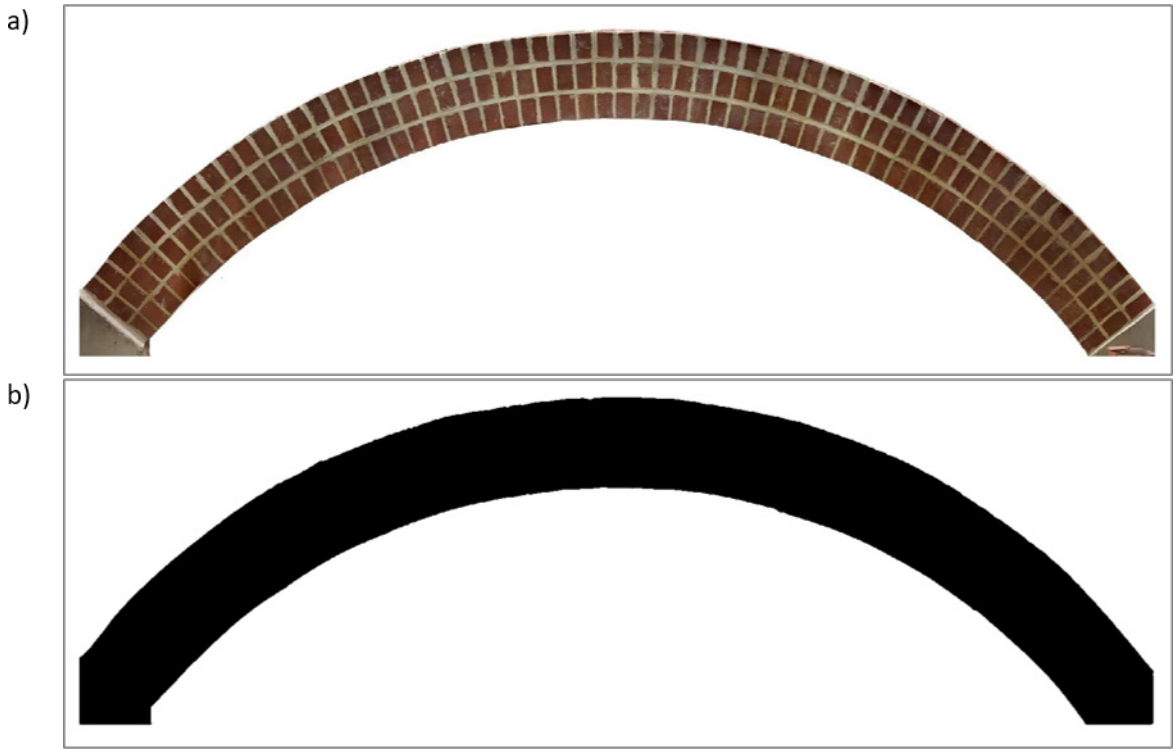


Fig. 4.9: Generating the background for filtered images; a) Original image; b) Binarization using thresholding.

4.6. Detection of non-masonry elements

For simple cases where the original image doesn't include additional structural elements (other than the blocks detected), then the geometry can be defined by using the block-mask only. However, if the structure includes other structural elements (i.e., steel, or concrete elements) not detected by the block-detection, then the structure image needs to be generated.

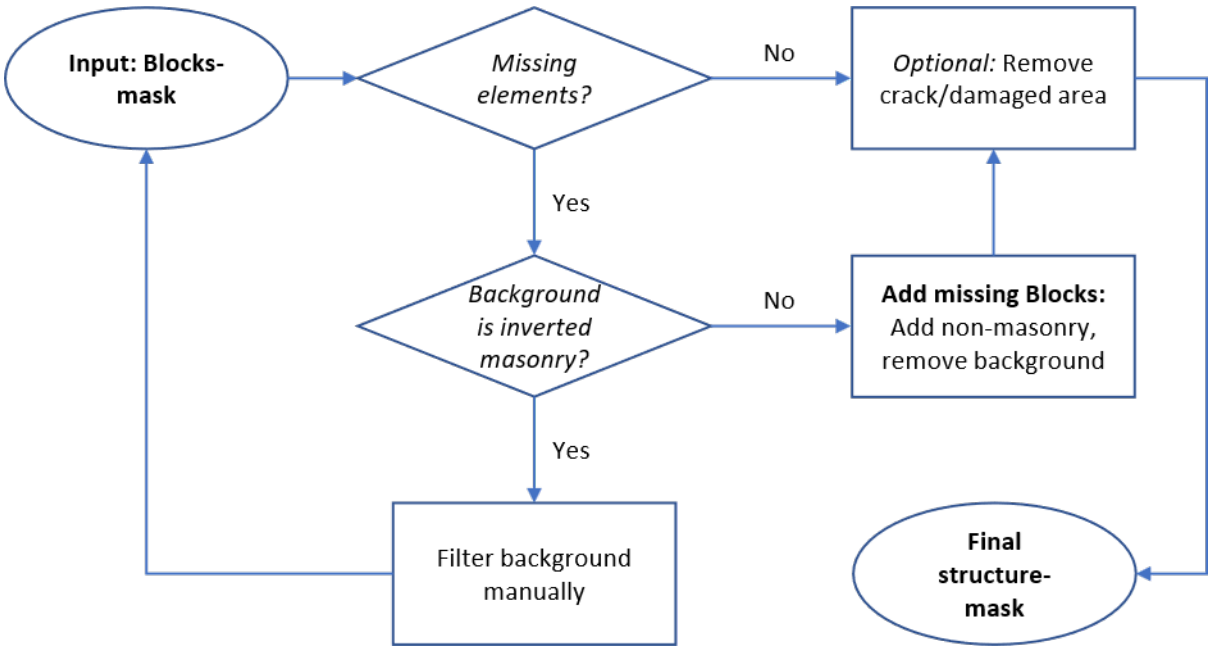


Fig. 4.10: Workflow: Detection of non-masonry elements.

The proposed methodology can be used to detect other structural elements only, if the masonry-mask was not used to define the background (i.e., by not using the method described in (Fig. 4.8)).

This is because the masonry-mask is used to identify the location of additional elements instead of creating the background. The gap between the concrete and the blocks that generates the final-structure is provided by applying an additional dilation to the masonry-image (Fig. 4.11: e) with width equal to the scaled mortar-thickness in pixels (Eq. (51)). The complete sequence that generates the final-structure is provided below:

1. [Required] Create a copy of the blocks-mask as the structure-mask (Fig. 4.11: b).
2. [Required] Create the modified-masonry-mask by applying an additional dilation equal to the expected mortar-thickness (Eq. (51), Fig. 4.11: c).
3. [Required] Where the modified-masonry-mask is equal to 0, the value of the structure is 255. Skip this step if the background was created from the masonry-mask (chapter 4.5, step 2.2).
4. [Optional] Where the damage-mask is equal to 0, the value of the structure is also 0 (used to ensure that broken blocks will produce multiple segmentations).
5. [Required] Where the background-mask is equal to 0, the value of the structure is also 0.

Additional dilation for masonry-mask (circular kernel): (51)

$$Iters = MTh/ImgScale;\ KSize = 3x3$$

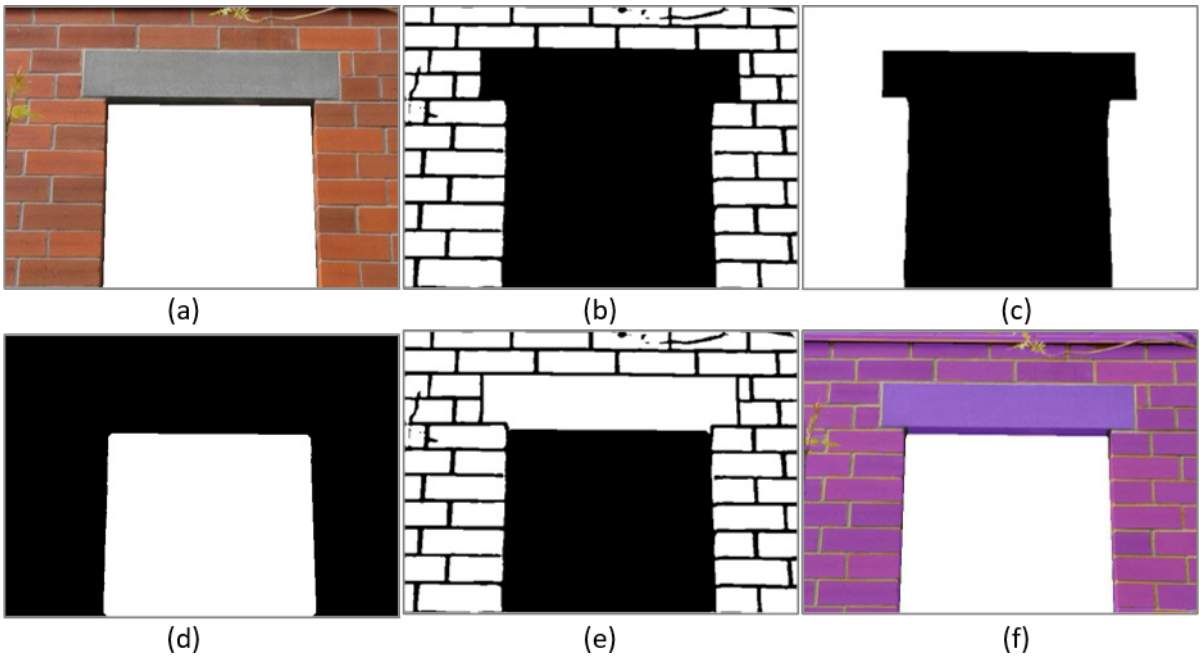


Fig. 4.11: Generating complete structure (if the image contains undetected elements); a) Original image; b) Output of the block-detection model; c) Masonry after dilation/erosion/dilation of blocks; d) Background mask using image thresholding; e) Final structure; f) Overlay of structure on image.

4.7. Post-Processing of binary images to improve the geometry extraction.

All binary images acquired are post processed to improve the binarized outputs and provide better shape definition (Fig. 4.12). This aims to improve the damage-evaluation and geometry-extraction. The post-processing of the images includes the binarization of outputs acquired from CNN, object removal by area and simple image-processing to separate and/or merge detected objects (using dilation -> erosion -> dilation sequence). Post-processing is applied on the following images (including typical options/actions):

- Detected-blocks (Removal of small foreground objects, image-opening)
- Detected-cracks (Removal of small foreground objects, image-closing)
- Masonry (Removal of small foreground/background objects, image-closing)
- Background (Removal of small foreground/background objects, image-closing)

- Final-Structure (Removal of small foreground objects, image-opening)

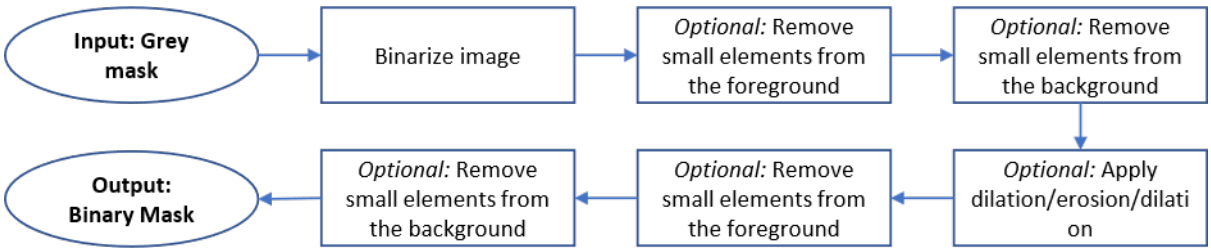


Fig. 4.12: Workflow: Post-processing of image-masks

The complete methodology is provided below.

1. [Required] Binarize the image using thresholding ($Thld = 0.5$, adjustable).
2. [Optional] Small object removal based on the scaled pixel area.
   2.1. [Optional] Remove small objects from foreground (white pixels).
   2.2. [Optional] Remove small objects from background (black pixels).
3. [Optional] Apply simple image-processing: Dilation/erosion/dilation in a sequence to separate or merge objects (i.e., either image-opening or closing by ignoring the first/last dilation).
4. [Optional] Small object removal, based on pixel area (repeat step 2). Only used if image-processing is applied (step 3).

The binarization threshold is equal to 0.5 ($0.5 * 255 = 127.5$ pixel-intensity, step 1). However, this is adjustable in cases where an object is incorrectly defined, and a lower/higher value could produce more accurate output. The removal of small objects from either foreground or background (steps 2 & 4), and before or after image-processing (step 3) is optional. Each foreground-object is isolated and evaluated in an empty-array. If the object doesn't satisfy the area-requirement, the value of the pixels in that location are inverted. For background-objects the method is the same but is applied on an inverted binary-image, which is then inverted back to normal. The area-threshold can either be the area in pixels or the scaled-area in real-units (Eq. (52)). The area-threshold is typically given equal to 0.0001m², with the aim to remove objects smaller than the area of the squared mortar-width.

Normally is best to preserve small objects from the background of the blocks/structure due to the ability of watershed-transform to close open-segmentations, when paired with distance-transform (see Section 4.9). This is especially useful since in certain cases the block-detection output may have an open line representing the mortar. A higher area-threshold (of foreground) could remove small blocks that were defined incorrectly (i.e., by using an area-threshold equal to 1/5th of the typical brick). Furthermore, the small threshold assists on removing small objects that would mostly cause issues during geometry-extraction (see Section 4.10) and numerical analysis (Section 4.11).

Additionally, simple image-processing usually considers a rectangular kernel of 3x3 size with 0/1/1 (image-opening) or 1/1/0 (image-closing) iterations of dilation/erosion/dilation respectively. This aims to merge non-structural, mortar and crack entities that are barely separated. The application of post-processing has varied results depending on the accuracy of the CNN output (blocks, cracks). The conversion of the scaled area-threshold to pixels can be obtained using equation 10:

$$\text{Threshold of scaled-area: } Thld' = Thld / ImgScale^2 \qquad (52)$$
$$\text{Area Threshold (Adjustable): } Thld = 0.0001\ m^2$$

4.8. Evaluation of the geometric properties of cracks

The output of the crack-detection was used to acquire the geometrical-metrics of the individual cracks (Fig. 4.13). Those include the location, area, length, average-width, and coverage of each detected object (Fig. 4.14, Table 4.1). Previous research has shown how to acquire the approximate length of

the crack using image-processing (Cabaleiro *et al.*, 2017; Kalfarisi, Wu and Soh, 2020; Loverdos and Sarhosis, 2022a). However, the methodology provided here is able to evaluate the precise metrics of individual cracks (assuming that their annotation is accurate).

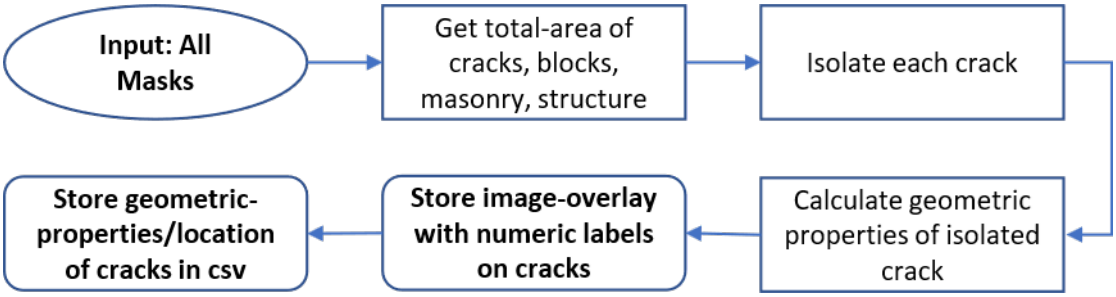


Fig. 4.13: Workflow: Evaluation of the geometric-properties of cracks.

The complete procedure used for the "*Crack Measurements*" stage of the workflow is provided below:

1. Acquire the damage-mask (Fig. 4.14b), masonry-mask (Fig. 4.8c), background-mask (Fig. 4.8d), and image-scale (Eq. (43)).
2. Calculate the total-area by counting the non-zero pixels of masonry, non-zero pixels of cracks, zero pixels of background, and all pixels of the image (used for different coverage ratios).
3. Perform watershed-transform (4-connectivity) using as input, markers, and mask the binarized-image of cracks, to isolate and identify every object by label (Fig. 4.14: c).
4. Loop over every label of the watershed-segmentation, ignoring label 0 (background).
   4.1. Isolate each crack to a new empty-array of equal size to the binarized image.
   4.2. Calculate the area by counting the pixels of the isolated-crack.
   4.3. Perform linearization to the isolated-crack to obtain its skeleton (Fig. 4.14: c).
   4.4. Perform border-following on the skeleton to obtain the correct order of the coordinates of the contour. Add the first vertex at the end of the extracted coordinates of the contour, if the first and last vertices are not equal.
   4.5. Calculate the length of the contour using the Pythagorean-theorem between every vertex. Retain only half the result since the border-following returns the perimeter (thus, doubling the line-length).
   4.6. Calculate the average-width by dividing the crack-area with the length.
   4.7. Calculate the coverage by dividing the area of the isolated-crack by the total-area of the masonry, cracks, inverted-background, and image.
   4.8. Identify the minimum and maximum coordinates of the isolated-crack to define its bounding-box (i.e., $[xmin, ymin]$, $[xmax, ymax]$). Acquire the middle-location of the crack by identifying the nearest-pixel of the skeleton at the centre of the bounding-box (i.e., $[xmin + dx/2, ymin + dy/2]$, used for visualisation purposes).

The skeleton of the cracks was acquired using the "*Scikit-Image*" function "*morphology.skeletonize()*" (step 4.3, Fig. 4.14: c). The accuracy of the obtained metrics depends highly on the accuracy of the crack-detection model (Table 4.1). The coverage provided depends on the total-area of the generated masonry-image (see Section 4.5, Fig. 4.8: c). However, the same method is used for the coverage over all the cracks, the whole structure (from background-mask), and the whole image. Furthermore, the metrics are scaled based on the image-scale where possible (Chapter 4.3, Eq. (43)). However, the length is calculated on a scaled-contour, since the vertical/horizontal scales may differ (step 4.5).

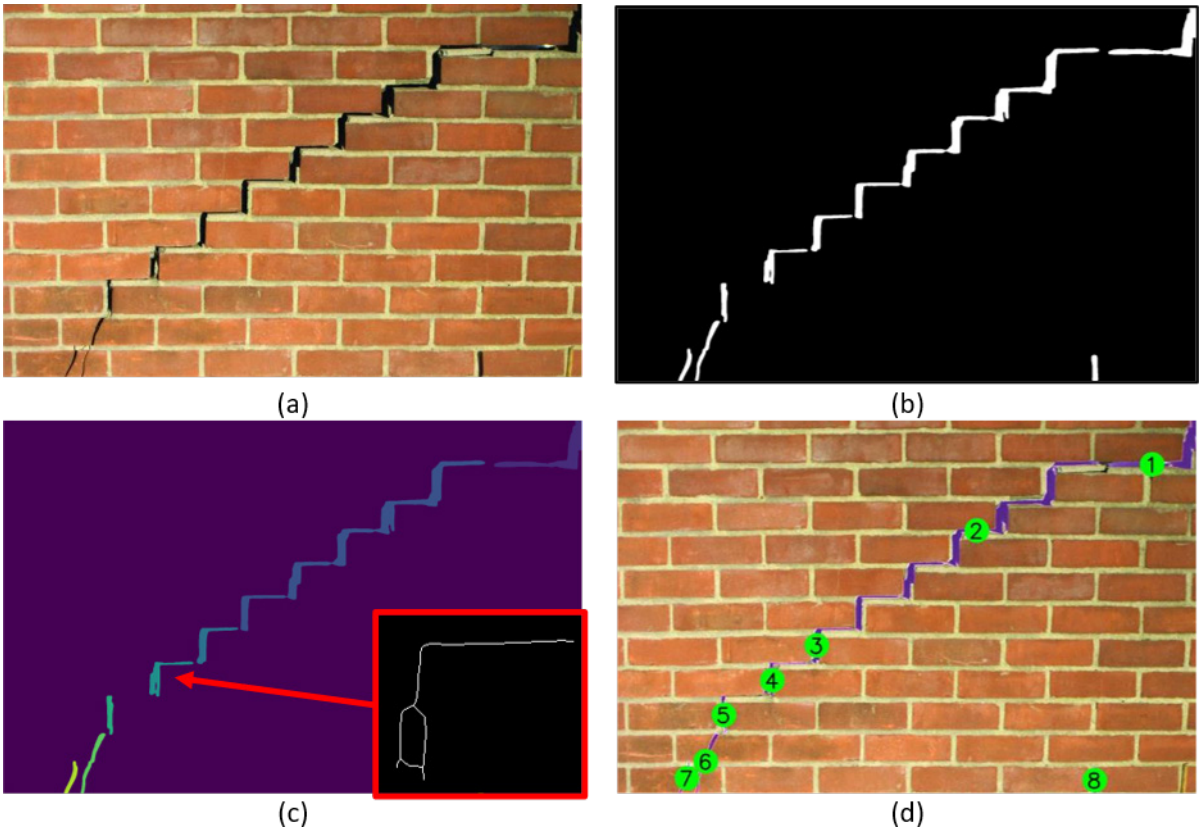


Fig. 4.14: Damage evaluation; a) Original-image; b) Output of the crack-detection model; c) Isolation and linearization of cracks using watershed-transform; d) Isolated-cracks labelled on original-image.

Table 4.1: Metrics of cracks in a masonry wall (the location starts from the top-left side of the image).

Label -	Location (xMid, yMid)	Area pixels	Length pixels	Width pixels	Area meters²	Length meters	Width meters	Coverage (M) %
1	[2079, 170]	13474	495.853	27.173	0.0051	0.306	0.017	0.413
2	[1397, 427]	37425	1571.955	23.808	0.0142	0.969	0.015	1.148
3	[775, 874]	5537	258.012	21.46	0.0021	0.159	0.013	0.17
4	[602, 1010]	5555	283.912	19.566	0.0021	0.175	0.012	0.17
5	[414, 1144]	3076	142.627	21.567	0.0012	0.088	0.013	0.094
6	[343, 1325]	3971	261.966	15.158	0.0015	0.161	0.009	0.122
7	[270, 1390]	2217	137.811	16.087	0.0008	0.085	0.01	0.068
8	[1856, 1398]	2190	79.657	27.493	0.0008	0.049	0.017	0.067
	Total:	73445	3231.793	172.312	0.0278	1.992	0.106	2.252

4.9. Watershed segmentation

The watershed segmentation process is used to separate open-shapes (i.e., when the mortar-line between 2 blocks is open) and label the individual objects with a unique ID. The process presented here is an improved and simplified version of the procedure demonstrated in ((Loverdos *et al.*, 2021a), Fig. 4.15). The main changes are the use of the CNN output (instead of edge-detection), the incorporation of the image-scale to automate mortar-generation, and exclusion of mortar at damaged locations. Those aspects improve not only the final-result but also fully automate the whole process, since it does not require any adjustments from the user.

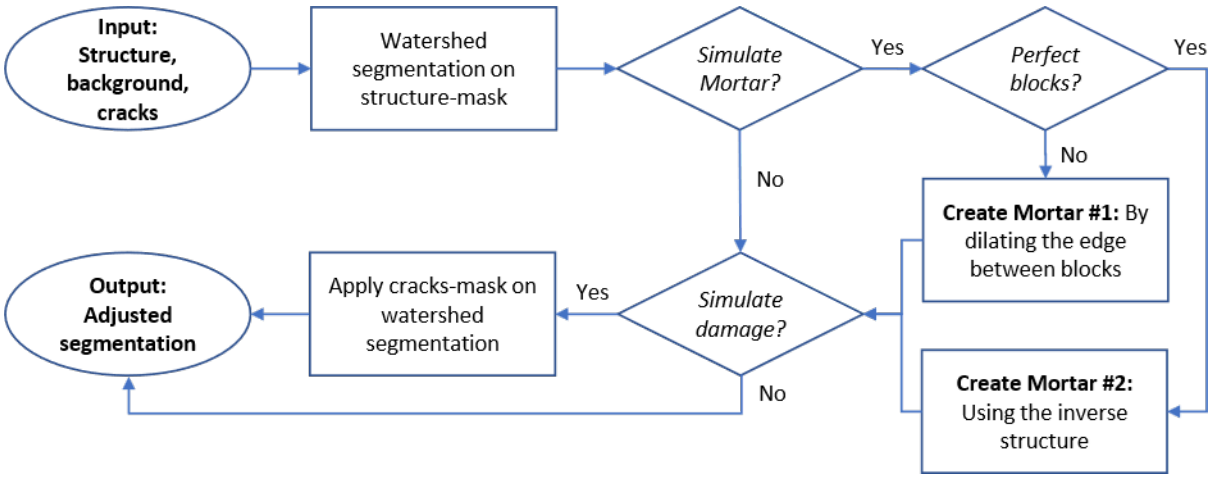


Fig. 4.15: Workflow: Adjusted watershed-segmentation

The complete sequence used to produce the "*Watershed Segmentation*" is provided below:

1. Source: Either the blocks-mask or the structure-mask (Fig. 4.16: c).
2. DT1: Acquire the distance-transform of the black area (used as input to WS, Fig. 4.16: d)
3. DT2: Acquire the distance-transform of the white area (used for markers, Fig. 4.16: e)
4. HMT: Perform h-minima-transform to the inverted DT2 (used for markers, Fig. 4.16: f)
5. LM: Acquire the local-maxima from the inverted HMT (used for markers, Fig. 4.16: g)
6. FM: Acquire the markers from the LM using 4-connectivity (final markers)
7. WS: Perform watershed-segmentation using as input the DT1, as markers the FM, and as mask the background-mask (Fig. 4.16: i).

The H-minima transform is using a custom function that replicates the MATLAB function. The segmentation process is using the "*SciPy*" library to generate the markers using the "*ndimage.label()*" function (which allows 4/8 connectivity). The watershed is applied using the "*Scikit-Image*" function "*segmentation.watershed()*". The remaining processes are using the *"OpenCV"* package.

This process is using the distance transform of the black area as input to watershed to ensure that each segmentation ends at the middle of the mortar (Fig. 4.16: d). Furthermore, the h-minima transform is used to remove false-minima and flatten the peaks (Fig. 4.16: f). A value of 5-pixels threshold for the h-minima transform is typically used, but for very low block-resolution; in other cases, a value of 3-pixels threshold may provide better results.

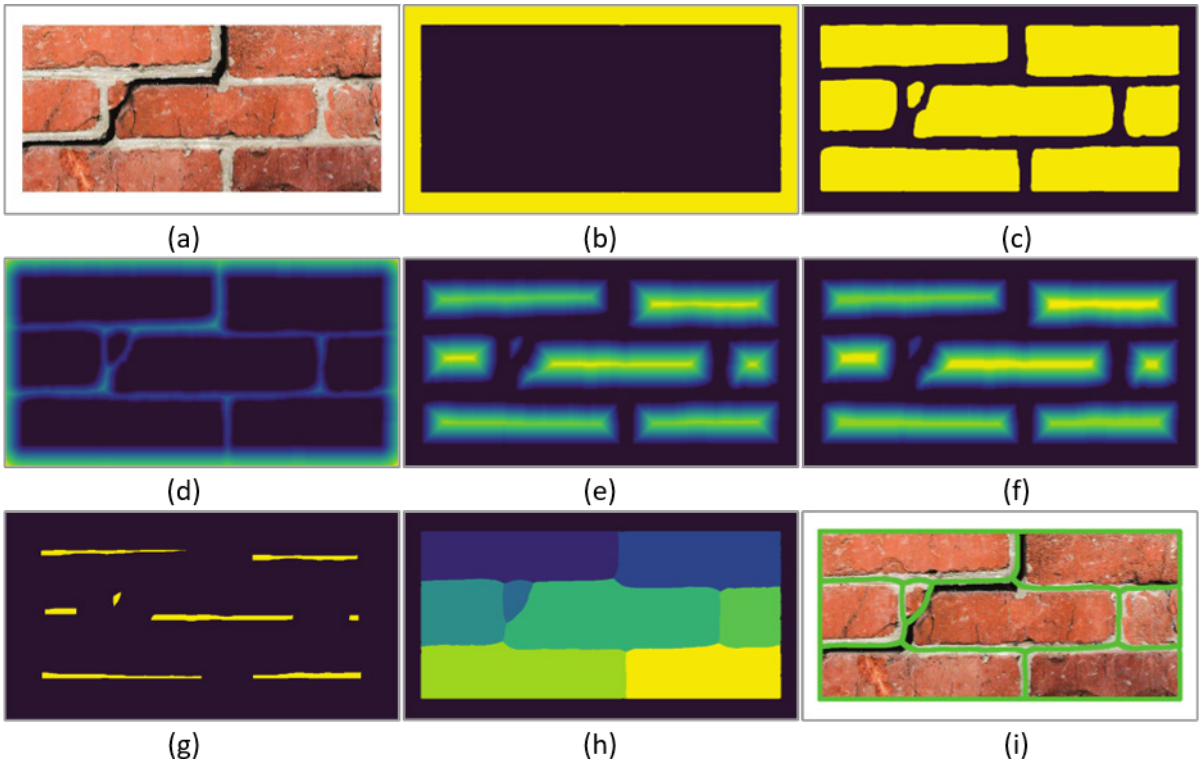


Fig. 4.16: Watershed-segmentation: a) Original image; b) Background mask; c) Binarized blocks; d) Distance transform of black area; e) Distance transform on white area; f) Inverted H-Minima transform; g) Local-Maxima of (f); h) Watershed-segmentation (Input: [d], Markers: [g], Mask: [b]) i) Segmentation-lines on original image.

Furthermore, the watershed-segmentation is adjusted to include the mortar and damage, using a unique-label that will be identified during the geometry-extraction. The complete process of the initial "*Segmentation Adjustments*" is provided below:

8. [Optional] Where the damage-mask is 255, the value of the watershed is [-2]. Used only if the mortar avoids damaged locations (step 9).
9. [Optional] Create the proposed mortar by marking the locations where two segmentations are connected (using a 2x2 ROI). Excluding background and optionally damaged locations (i.e., Labels [0], [-2]). Dilate the proposed-mortar with respect to the image-scale.
10. [Optional] Include the filtered structure to improve the definition of mortar. The structure-mask is filtered using a dilated background to remove excessive mortar from the perimeter.
11. [Optional] Where the final mortar-mask is 255 the value of the watershed is [-1].
12. [Required] Where the damage-mask is 255, the value of the watershed is [-2].
13. [Required] Where the background mask is 255, the value of the watershed is [0]. Use the dilated background-mask if the structure mask was used to create the mortar (step 10).

Mortar-line dilation (circular-kernel): (53)

$$KSize = MTh/ImgScale - 1, Iters = 1$$

Every step of the mortar-generation is optional (steps 8-11). For example, by ignoring the generation and application of the mortar-mask (i.e., step 11), the geometry-extraction will not include the mortar (Fig. 4.17: f), which is ideal for simplified micro-modelling using let's say the discrete element method. Avoiding generating mortar at damaged locations is preferable when damage is present at blocks (step 9), to avoid generating mortar where only the crack separates a block (Fig. 4.17: g).

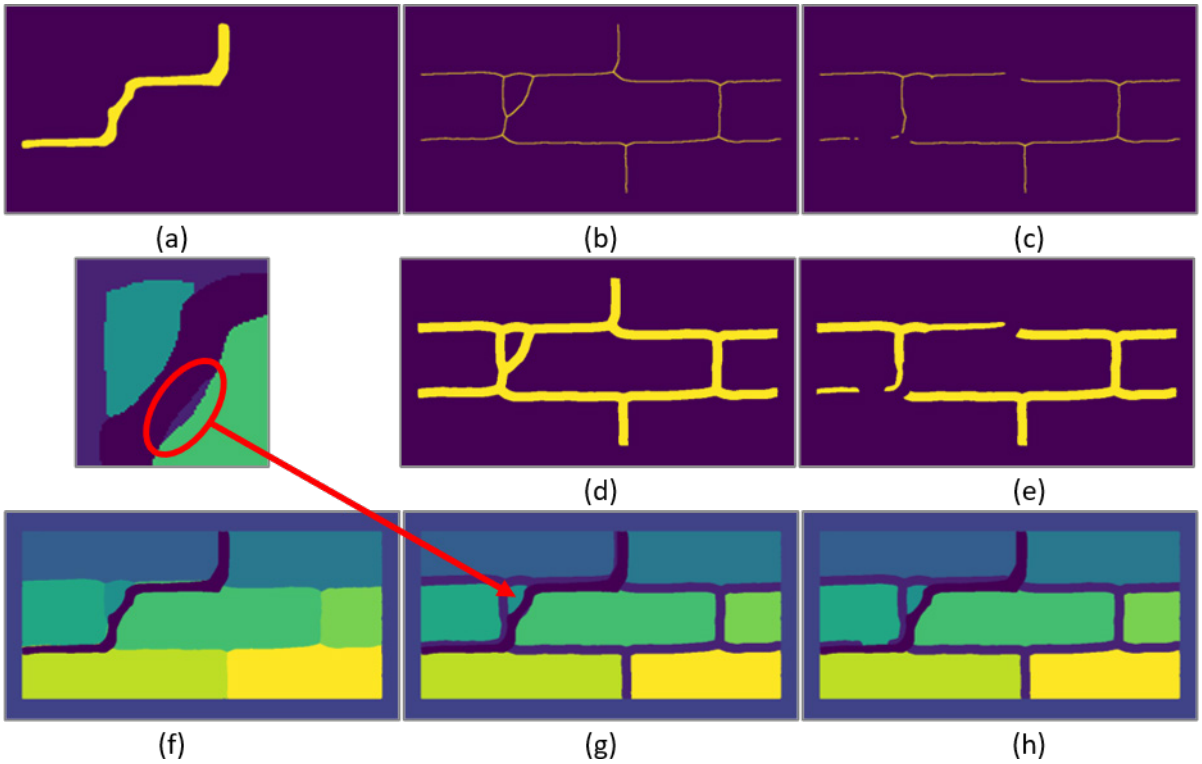


Fig. 4.17: Including mortar and damage; a) Damage-mask; b) Proposed mortar #1; c) Proposed mortar #2 (excluding damaged locations); d) Generated mortar #1; e) Generated mortar #2; f) Adjusted segmentation without mortar; g) Adjusted segmentation with mortar #1; h) Adjusted segmentation with mortar #2.

The original source of the watershed segmentation can be used to improve the definition of the mortar in the final watershed (step 10). The modification of the mortar-mask using the block-detection output was typically avoided. This is because the blocks-mask may contain noise (i.e., black spots inside the blocks), which produces inaccurate segmentation. However, because the post-processing removes efficiently small-spots (section 4.7) and due to the reliability of the CNN model, the blocks-mask can be used in many cases to adjust the mortar-mask. Although, the post-processing options should be adjusted to remove small-objects of background from the final-structure (to avoid mortar within blocks). Additionally, the dilation of the background (step 10), to filter the imported mortar-mask, is optional and is used to remove excessive-mortar from the perimeter of the structure (Fig. 4.18: a). The iterations of the dilation depend on a case-by-case scenario using the smallest possible value to remove all excessive mortar from the perimeter. Furthermore, the use of the dilated-background, to limit the watershed-segmentation (step 11) will generate a uniform distribution of mortar but will also decrease the structural perimeter (Fig. 4.18: c).

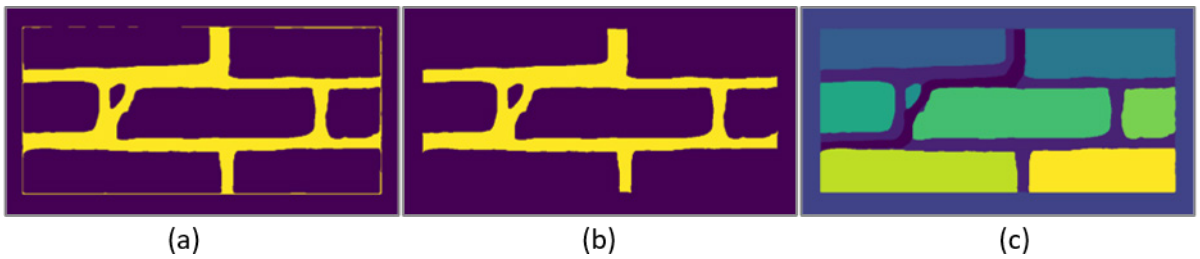


Fig. 4.18: Imported-mortar (alternative method); a) Mortar-mask from structure (excluding background); b) Filtered mortar-mask (excluding dilated-background); c) Adjusted segmentation with imported-mortar-mask.

Moreover, the watershed-segmentation is again modified to avoid common issues with the geometry-extraction (i.e., falsely detected end-points that define an interface). The simplified sequence is presented below (part of the "*Watershed Adjustments*"):

14. Watershed-Cleaning: Remove isolated pixels not connected with a segmentation in 4-connectivity (i.e., if the pixel has a unique value compared to the vertical/horizontal pixels).
15. Watershed-Corrections: Adjust the value of duplicate labels in the watershed (i.e., two segmentations having the same label). Store adjusted labels with their original value to identify the properties of materials (i.e., Eq. (54)).

Watershed Corrections: (54)

$$Corrections_i = [NewID, OldID].$$

4.10. Geometry extraction for BIM (Building Information Modelling)

The geometry-extraction includes the extraction of coordinates of each detected-object (blocks, cracks, background), simplification of the extracted contours (generalization), and adjustments to the extracted-shape. Those aim to provide an accurate but efficient geometrical-model for either documentation of numerical analysis. Similar to the "*Watershed Segmentation*" section, the geometry extraction is an improved and simplified version of the process demonstrated in ((Loverdos *et al.*, 2021a), Fig. 4.19). The main change is the incorporation of the image-scale to automate every input required by the user. More details, about individual sections, are found in the original manuscript.

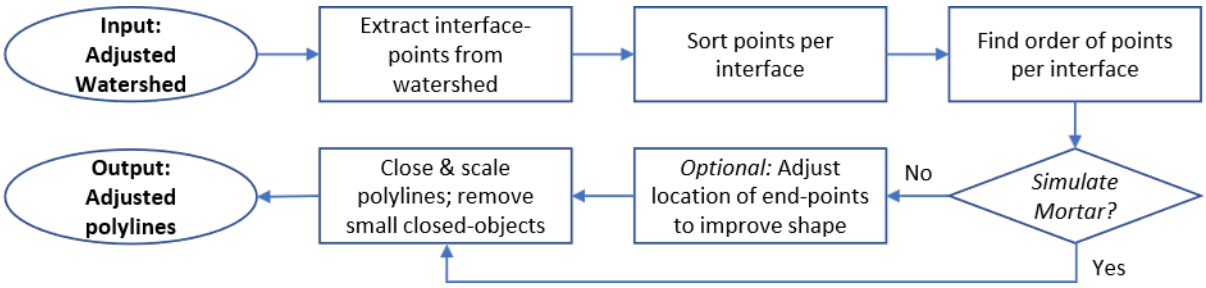


Fig. 4.19: Workflow: Geometry extraction.

The extraction of coordinates is applied to every interface between detected objects (i.e., connection between 2 objects), with end-points the points of the contour that define the start/end of the interface. This process does not extract the perimeter of the whole object, otherwise, the interface between 2 objects would be extracted twice. This would cause issues when a single interface would be modified separately from the other (i.e., during generalization). Furthermore, that would increase the distance between segmentations by 1-pixel. Thus, it would be impossible to simulate simplified micro-modelling (where the mortar is modelled as a zero-thickness interface). The simplified sequence, of the "*Contour Detection*" part of the workflow, is provided below:

1. [Optional] Double the size of watershed segmentation by considering every pixel a square of 2x2 size. Used to avoid merging U-shaped interfaces of 1-pixel width.
2. Pad the modified watershed-segmentation by 1 pixel of zero value on every direction.
3. Scan the image with a 2x2 ROI to identify interface-points (2+ labels) and end-points (3+ labels). Store the points with their position (top-left corner) and ID value (the unique values of the ROI). End-points are also broken to every combination of 2 labels and stored as interface-points.
4. Separate the interface-points and end-points by every unique interface (of 2 labels).
5. Isolate every interface to an empty array. Perform border-following to identify the correct order of the points of the interface.
6. If the contour has 2+ end-points, store every section between end-points once (the border-following returns the perimeter; thus, 2 occurrences of each section). If it has less than 2 end-points, the whole contour is stored (closed-object). Each entry is stored in the "*Line-List*" with its ID (the 2 unique-labels).

The border-following is provided by the *"OpenCV"* library using the command *"findContours"* (Suzuki and Abe, 1985).

Furthermore, each contour stored is adjusted to reduce the number of vertices (Fig. 4.20), improve the general shape (Fig. 4.21), and connect the interfaces to form closed-objects. The following sequence is part of the "*Geometry Adjustments*" part of the workflow:

7. [Optional] If RDP generalization is used and the object is closed (i.e., less than 2 end-points), reform the contour to start from the point with the maximum distance from the first (ensures that the first point is a corner).
8. [Optional] Generalize the contour using either RDP or the proposed algorithm in (Loverdos *et al.*, 2021a), using the scaled threshold provided in Eq. (55) (Fig. 4.20).
9. [Optional] Relocate end-points attached to 3 blocks, so that the angular difference between 2 line-segments that form ~180° is exactly 180° (Eq. (56), Fig. 4.21). Additional conditions are required to avoid flattening rubble/arch-lines. This must be applied 3x times to adjust all lines. Furthermore, this step is applied only if generalization is used.
10. Connect open-interfaces (i.e., contours with 2 end-points) to form closed-shapes and store the combined contour. Closed-objects (i.e., contours with less than 2 end-points) are stored as is. Each entry is stored in the "*Block-List*" with its ID (segmentation-label).
11. Test the area of the closed-objects and remove entries with area lower than the threshold, using the scaled threshold provided in Eq. (52).
12. Remove interfaces that were not used in the final-list of closed-objects (from step 11).
13. Scale every individual interface and closed-object to the real dimensions (Eq. (43)).

Scaled Threshold: $Thld' = Thld / ImgScale$ (55)

Generalization Threshold (Adjustable):
$Thld = 0.005\ m$ (for higher accuracy)
Or $Thld = 0.01\ m$ (for lower accuracy)

Condition #1 (main): $|Ad_i - 180°| \leq t_a$ (56)
Condition #2: $|Ad_j - 90°| \leq t_a;\ i \neq j;$
Condition #3: $|Dir_i - A_t| \leq t_a;\ A_t = [0° \vee 90° \vee \ldots \vee 360°];$
Angular Threshold (Adjustable): $t_a = 20°$

Ad_i is the largest angle formed between the 3 line-segments. Ad_j is any remaining angle. Dir_i is the direction of the proposed flattened-line in the global coordinate system. The generalisation threshold of $Thld = 0.01$, is forced when the image-scale is close-to or higher-than 0.05 meters/pixel. Furthermore, the generalization with the RDP algorithm is applied using the "*OpenCV*" function "*approxPolyDP*" (Ramer, 1972; Douglas and Peucker, 1973). The area of the closed-objects is evaluated using the "*Gauss's area formula*".

Reforming the contour to start from a different location is only required when RDP is used for generalization (step 7), since the function is already incorporated in the proposed generalization-algorithm in (Loverdos *et al.*, 2021a). If the reformation of the contour is not used, the top-most point will be included in the generalized-contour and may form a triangular shape incorrectly. The generalization is required when the geometry is used to create numerical-models (step 8). Otherwise, the model will contain excessive detail, which will cause the analysis to fail.

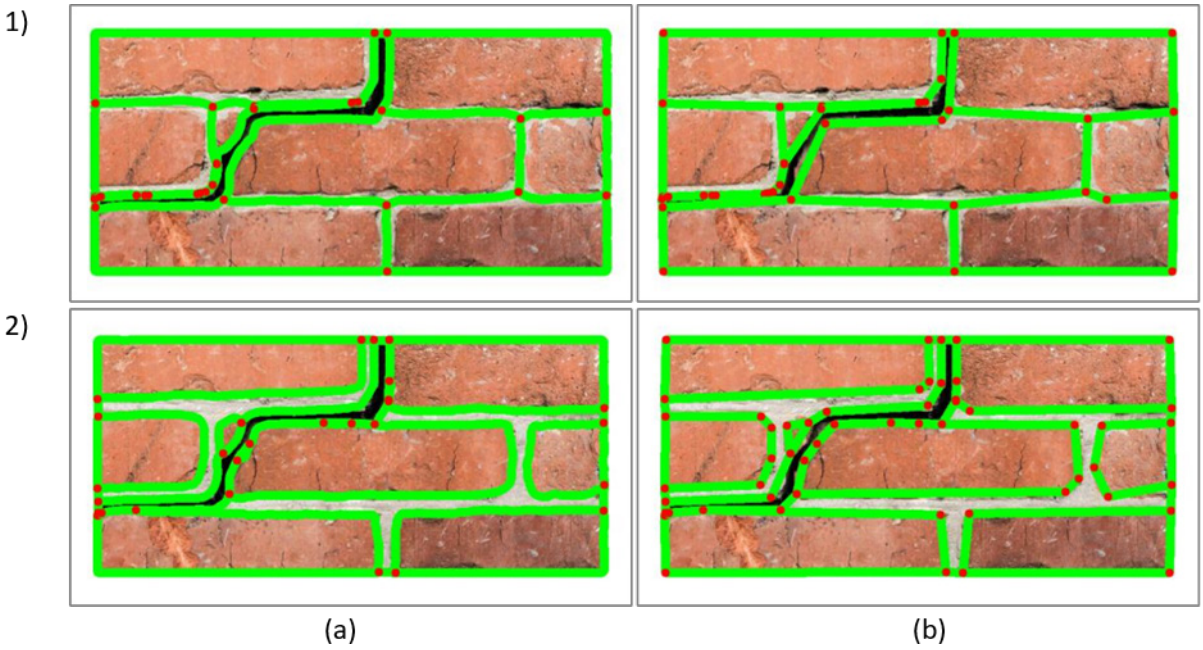


Fig. 4.20: Generalization; a) Adjusted-contours (Red: End-points only); b) Generalized-lines (Red: All points); 1) Without mortar (Segmentation from Fig. 4.17:f); 2) With mortar (Segmentation from Fig. 4.18:c).

The relocation of the end-points is applied only on end-points with exactly 3 unique and positive $OldID$ labels (Eq. (54)). Additionally, the proposed conditions test the angle and direction of line-segments and not the whole interface (step 9). Where a line-segment is formed by considering the end-point and the previous of the interface. The main condition tests if any angle formed between 2 line-segments is ~180°. The 2nd condition tests if any remaining angular-difference is ~90°, to avoid flattening rubble/arch-lines. The 3rd condition ensures that the direction of the flattened-line (that forms 180°) is of multiples of 90°, to avoid flattening arch-lines.

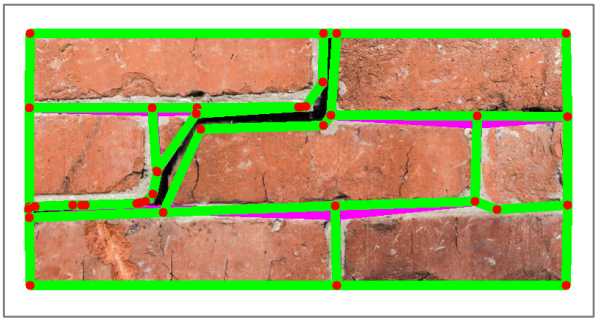


Fig. 4.21: Adjusting the geometry of generalized-lines (Magenta: Original lines)

Finally, individual interfaces are connected because certain analysis software require closed-shapes to define the geometry (step 10), to filter blocks based on their area (steps 11-12), and to identify the location of closed objects (section 4.11; steps 4-6). However, mortar-blocks are excluded from forming closed-shapes, since their definition is unnecessary and unreliable.

4.11. BIM to structural analysis of masonry structures

The modelling procedure includes the generation of triangular mesh, identifying the location of every object in space, storing the geometry in a CAD file, and generation of the numerical-model (Fig. 4.23). The user is allowed to select either the contours or the generalized-lines to generate the model. Contours and generalized-lines are interchangeable for all parts of the methodology presented in this

sub-chapter (except for meshing). However, the generalized-lines are preferred since the contours contain excessive number of points.

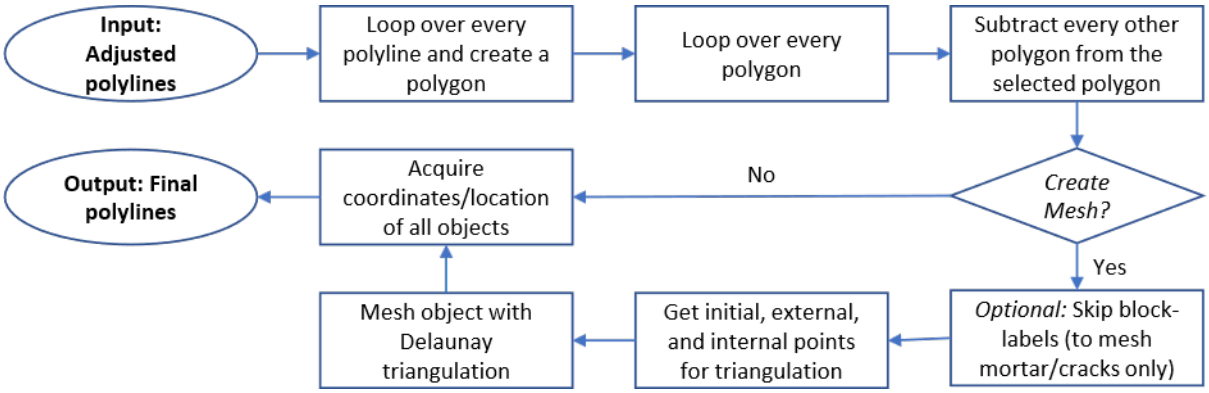


Fig. 4.22: Workflow: Acquire final polylines for CAD/DEM modelling.

Optionally, the mesh is generated to investigate crack propagation (Fig. 4.23: a). This improves the procedure highly, since it ensures that every mesh element has a convex-shape. Mesh of concave-shape could cause interlocking between mesh-elements and thus, overestimation of the structural capacity. The mesh can be generated for blocks, mortar, and/or damage. However, in most cases the failure is expected on the mortar. Thus, the presented case excludes meshing the bricks. It should be noted that the mesh should not be generated for the contours, since they contain too many vertices that need to be seeded. The size of the mesh and tolerance is adjustable based on the accuracy required (step 3). The complete procedure to generate the mesh is provided below:

1. Loop over every closed-polyline in the "*block-list*".
   1.1. Create the polygon for the polyline and store it in the "*polygon-list*", including its ID.
2. Loop over every polygon to adjust its shape.
   2.1. Subtract every other polygon from the selected polygon, to ensure the shapes don't overlap. Update the old polygon with the adjusted-polygon.
   2.2. [Condition] Skip polygons that their subtraction would cause the new-polygon to have zero area (i.e., to retain inner-openings on materials).
3. Loop over every polygon in the "*polygon-list*" to create the mesh.
   3.1. [Optional] Skip polygons with positive old-label (i.e., $OldID > 0$), to avoid meshing the bricks.
   3.2. The points that form the perimeter of the polygon are the initial outer-seeds.
   3.3. Create additional outer-seeds, between initial-seeds, every $x1'$ adjusted-distance (distance adjusted to fit evenly, i.e., $x1 = 10mm$). Only add seeds if the distance between the initial-seeds is larger than $x1$. Do not add seeds that are near an outer-seed at $x2$ distance (tolerance value for corners, i.e., $x2 = 0.1 * x1$).
   3.4. Create the seeds for the inner, in a bounding-rectangle covering the whole object, every $x1$ distance (i.e., $5x5$ points to cover a bounding-rectangle of $40x40mm$). Do not add inner-seeds that are near an outer-seed at $x1$ distance.
   3.5. Apply Delaunay triangulation using all seeds. Retain only the area of the triangulation that covers the polygon (Delaunay returns a large triangulated-rectangle).
   3.6. Loop over every mesh-element and extract the coordinates that form the perimeter. The extracted coordinates form the polyline of the mesh-element.
   3.7. Store the ID (block ID), polygon, and polyline of the mesh-element in the "*mesh-list*".
   3.8. [Optional] Break-down the polyline into 3 lines and store them individually in the "*mesh-line-list*", along with its ID. Do not store sections that have the same coordinates as an existing line within the "*mesh-line-list*".

In certain cases, the location of every object is required to assign the groups to the numerical analysis software (i.e., when using lines to slice the model instead of closed-shapes). The locations are acquired from the polygon of every segmentation and every mesh object (Fig. 4.23: b & c). However, the

location acquired should be an inner-point and not the centroid. Mostly because the centroid may fall outside concave-shapes. The simplified procedure to acquire the locations is provided below:

4. Loop over every polygon in the "*polygon-list*" and "*mesh-list*" to identify their location.
   4.1. Acquire the inner-location of each polygon using the appropriate function (not centroid).
   4.2. Store the coordinates of the location in the "*location1-list*", along with its ID (equal to block ID) and type of the object (i.e., mesh or segmentation).
5. [Optional] Loop over every object in "*block-list*" to acquire its location from its segmentation (alternative method; may be used to assign the background).
   5.1. Isolate and linearize the individual segmentation to obtain its skeleton (i.e., Fig. 4.14: c).
   5.2. Acquire the location of the pixel closer to the middle of its bounding box.
   5.3. Store the scaled-location in the "*location2-list*", along with its ID.
6. [Optional] Scan every pixel in the watershed to identify the location of damage/mortar (alternative method; may be used to assign the mesh of mortar/damage).
   6.1. If the pixel-label is that of damage, store the scaled coordinates to the "*location3-list*".
   6.2. If the pixel-label is that of mortar, store the scaled coordinates to the "*location4-list*".

The surface/polygon of each object is acquired using the "*Shapely*" function "*Polygon()*". The modification of the surface is applied using the appropriate functions of the same package. The inner-location of each object is acquired using the "*representative_point()*" function of "*Shapely*".

The user can select either the closed-shapes or the interfaces for the model. Certain numerical analysis programs require closed-shapes to generate an object. In that case, the mesh is designed using the "*mesh-list*" and "*block-list*". However, if the numerical analysis program allows slicing, the interfaces can also be used and improve the generation speed, since the number of points provided is halved. In that case, the mesh is designed using the "*mesh-line-list*" and the "*line-list*". The complete geometry is stored to a DXF file to allow compatibility with multiple numerical analysis programs (Fig. 4.23: d). Each object is assigned to a different layer, depending on the object's old-ID (chapter 4.9, step 15).

Then, the DXF file is converted to crack coordinates to allow integration with UDEC. The conversion from DXF to UDEC is accomplished with a companion program that reads the DXF file and acquires the coordinates and layer of every polyline (Fig. 4.23: e). This allows to adjust the CAD file before modelling, if required. The groups in UDEC are assigned using the inner-locations (step 4), of every object except the background. That's because the inner-area of the background may contain the structure. The background is defined correctly by the middle-location of the linearized-segmentation (step 5). The conversion/reading of DXF files is accomplished using the *"ezdxf"* package in python.

More specifically, the outputs acquired from the program are the following:

- [Main] DXF file with the geometry of every object assigned to the appropriate layer.
- [Main] CSV files of inner-locations of all objects (separated by material, step 4).
- [Main] CSV files of the middle-locations of all objects (separated by material, step 5).
- [Main] CSV files with the location of every pixel of mortar/damage (step 6).
- [Main] CSV file with the location and geometric properties of cracks (Table 4.1).
- [Companion] TXT files to generate the geometry in UDEC (separated by material).

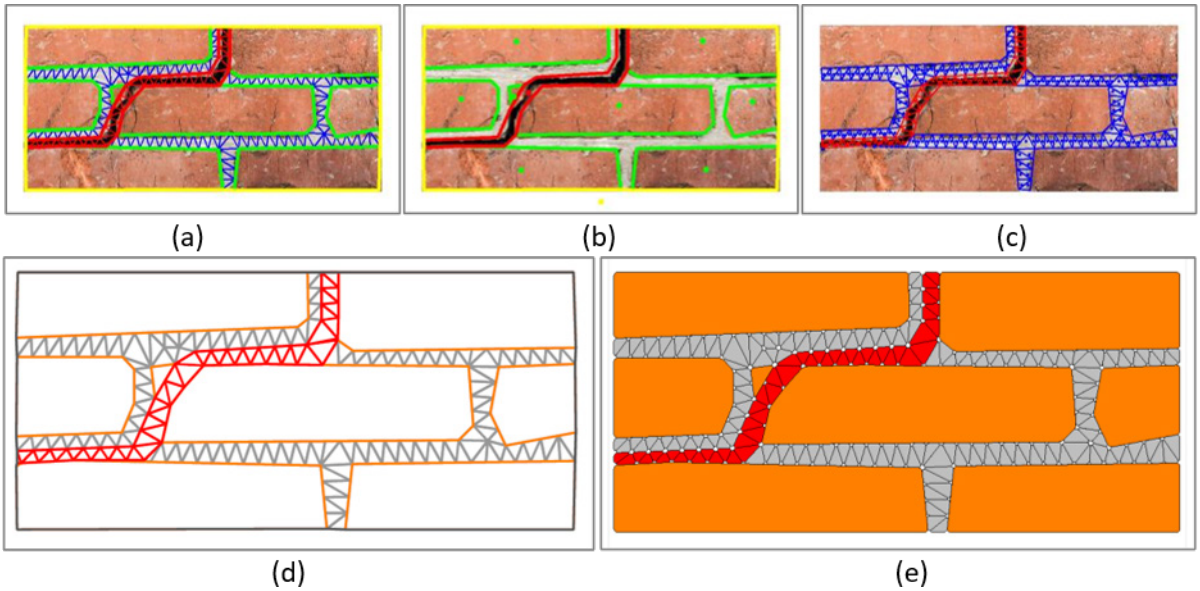


Fig. 4.23: Creating the numerical model; a) Creating triangular mesh; b) Inner-locations of segmentations (excluding mortar); c) Inner-locations of mesh elements (mortar and damage); d) AutoCAD drawing of the geometry (layers assigned using the block ID); e) UDEC model for numerical analysis (groups assigned using the inner-locations).

4.12. Case Study of Railway Bridge

The entire workflow was tested on a real case study of the railway bridge located at Viaduct Ln, Leeds, UK. (Fig. 4.24). Regarding the structure itself, the specified bridge has undergone frequent renovations to replace deteriorated brickwork, which explains the varying levels of brick quality along the structure The structure also includes areas with other defects (e.g., mold, efflorescence, etc.), which provide a good example of possible outcomes. The size of the sample also demonstrates that the methodology can be applied to larger constructions. The input-image was acquired from the 3d-mesh, generated using photogrammetry, which allows to obtain realistic dimensions. The model was filtered, within the photogrammetry software, to remove background/openings (Fig. 4.24). The filtering of background/openings allows the identification of undetected elements using the methodology described in Fig. 4.11. The image was then imported to the proposed workflow.

Fig. 4.24: Orthorectified image of a section of the railway bridge in Leeds, UK.

The output of the block-detection is the first examined (Fig. 4.25). Visually, renovated regions have excellent detection rate, while deteriorated regions have been classified sufficiently. Moreover, the model correctly identifies the brick-pattern on elements affected heavily by efflorescence. In contrast, locations affected by mold-discolouration have not been classified correctly, forming large gaps on the structure. Such severe cases were not used during training of the CNN model and thus, the model is exhibiting difficulties in identifying the brick-pattern. Nevertheless, the accuracy of the CNN model can be enhanced by incorporating samples of the specific defect. Likewise, foreground objects that cover the brickwork are not identified as bricks, which also forms large-gaps on the structure. Spalling and other defects have minimal effect on block-detection and thus, such locations are identified correctly.

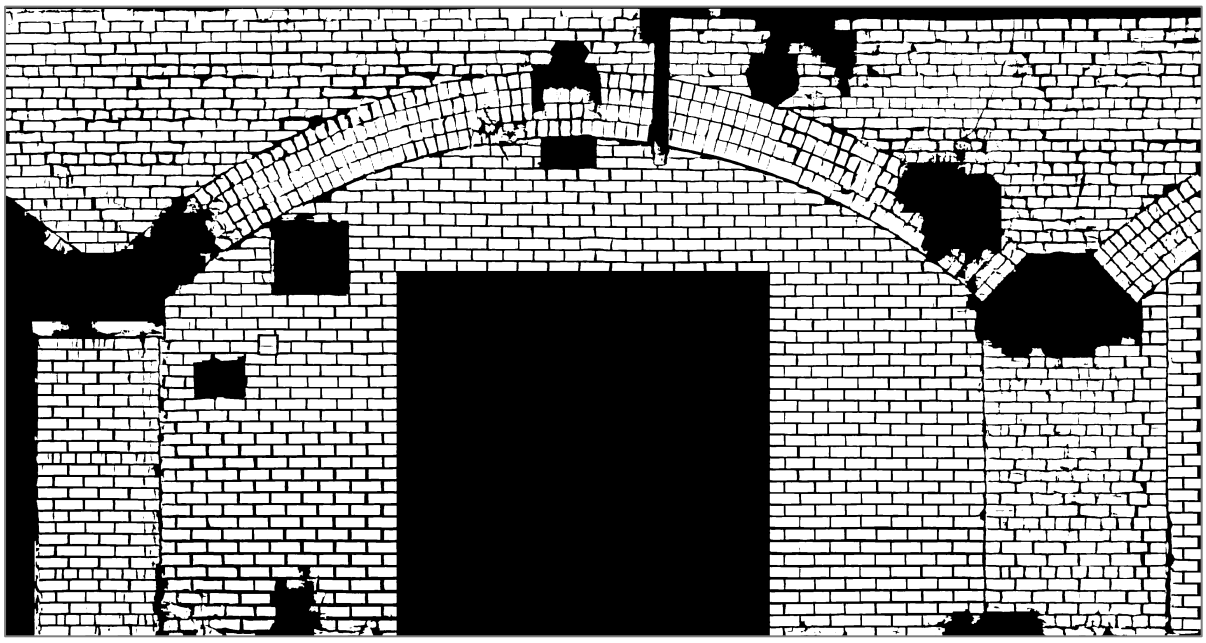


Fig. 4.25: Initial block-detection output, after post-processing to remove small foreground-objects (small white regions).

Large gaps on the structure could cause either of the following issues. If the background is generated using the masonry-mask, then the structure will have gaps in those locations. Otherwise, if the background is generated by the white-regions of the image, then the gaps will be assigned to neighbour blocks. None of the aforementioned is appropriate. Thus, the preferable alternative is to assign an area to those gaps, using the method described in Fig. 4.11, and generate the structure-mask (Fig. 4.26). A possible solution is to modify manually the block-mask directly before the watershed-segmentation. After post-processing (to cover the large-gaps), the final-structure is then ready for geometry extraction.

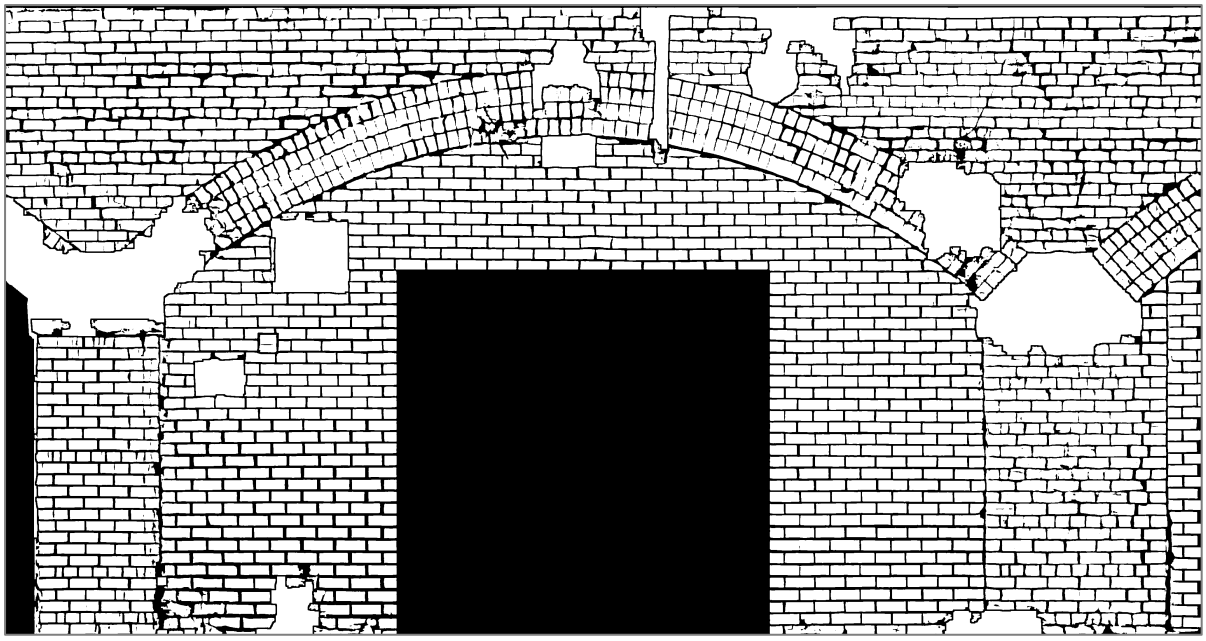


Fig. 4.26: Final structure used to generate the geometry (using the methodology described in section 4.6).

The modelling method considered is the simplified micro-modelling, where the mortar is simulated as a zero-thickness interface. Detailed micro-modelling is avoided due to the large number of blocks in the structure, which would become much larger due to the mesh elements. A large number of interfaces would require much more time to generate the numerical model, would increase largely the computational time for the analysis, and the analysis may even fail to complete. Detailed micro-modelling is mostly advised for smaller structures. Nonetheless, the output of the proposed workflow demonstrates impressive results (Fig. 4.27). The lines were flattened correctly to improve the output, without affecting arch-lines (Fig. 4.27: b, Eq. (56)). Furthermore, deteriorated regions have proper brick-shape. The few regions that have issues (merged-blocks) are locations where the mortar is barely visible or not visible at all.

Fig. 4.27: Final generalized-lines (Red-Points: Vertices of the lines); a) Overall view; b) Zoomed-in view.

Finally, the CAD file is produced (Fig. 4.28), which can be used to import the geometry to a numerical analysis software (i.e., UDEC). If required, large-gaps can be adjusted in the CAD file (or in the analysis software directly), by dividing the regions with simple lines. Although, when the geometry is used only for simple documentation, gaps in the model highlight severely discoloured areas, proving useful for analysis. In which case, it is preferable to identify and assign a separate group for those regions, which may be part of future work. Moreover, the user has the option to use the interfaces instead of closed-objects for the CAD model. In that case, the total number of vertices would be reduced since most interfaces are defined twice (interfaces between connected-elements). For instance, only 12,441 vertices are required to describe the geometry in the form of interfaces, while 14,920 vertices are required to describe the geometry as closed-objects. It should be noted that only verified closed-objects, or interfaces that are part of a closed object, are exported to the CAD file. The layers in CAD and the groups in UDEC (i.e., background, mortar, cracks, blocks), where assigned automatically during the "*Modelling*" part of the workflow (section 4.10).

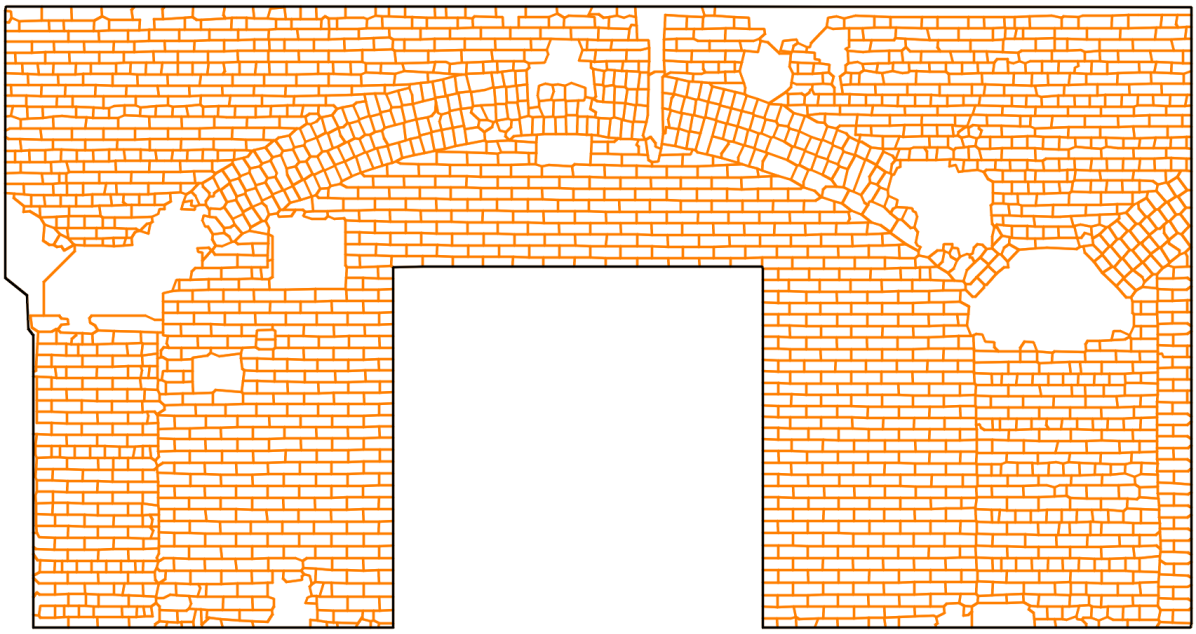


Fig. 4.28: Geometrical model in AutoCAD (for simplified micro-modelling).

4.13. Case Study of Simple Structure

An additional case-study is shown to demonstrate the potential of the workflow on structures with higher mortar-visibility (Fig. 4.29). The following example is a simple structure, which was chosen because it contained both simple-pattern and arches above the opening. The orthorectified image was acquired using photogrammetry and was filtered to identify undetected-elements (similar to the previous case study; section 4.12)

Fig. 4.29: Orthorectified image of simple structure.

In this case, the detection of block-elements is optimal (Fig. 4.30). There are some distorted units, but only those in close-proximity to foreground-objects. In that regard, the pipe (on the left side of the structural section) could be filtered out completely, if the 3d-mesh was generated using more photos of the area behind the pipe. For the specific case, most obstructed brick-units have been visualized, but not all to filter the pipe out completely. Additionally, the concrete-elements were identified automatically using the same procedure shown in section 4.6.

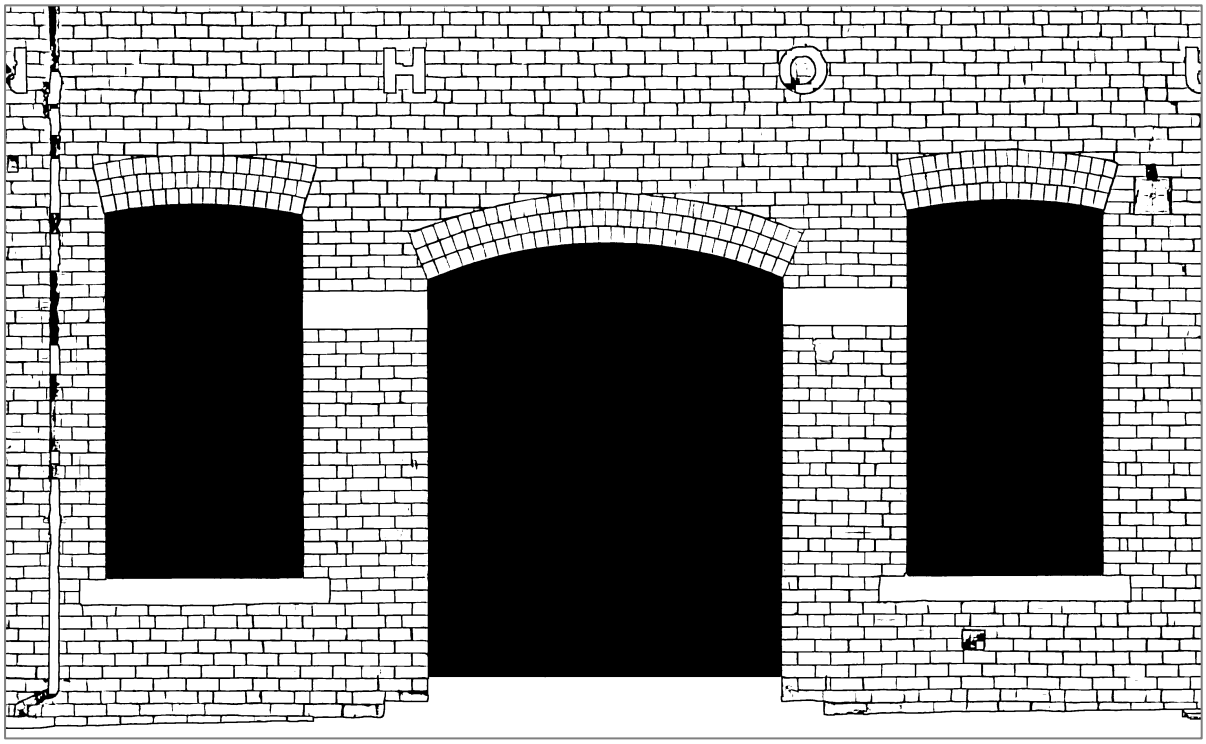


Fig. 4.30: Block detection, after post-processing to identify the concrete-elements.

The geometry-extraction demonstrates exceptional results, even for large structures with a great number of brick elements (Fig. 4.31). Moreover, the generalization algorithm reduced efficiently the number of vertices (to 12,231; see red points in Fig. 4.31). The shape of the blocks is also correctly defined, which should be coined partially to the line-adjustments applied by the provided algorithmic-logic (Fig. 4.21). This is easier observed in the AutoCAD drawing (Fig. 4.32), which clearly shows the shape of the final generalized-lines. The results of this case-study demonstrate that the extracted-geometry is mainly influenced by the quality of the binary-images (CNN output). Therefore, the primary enhancement that can be implemented for the presented method involves improving the

CNN models.

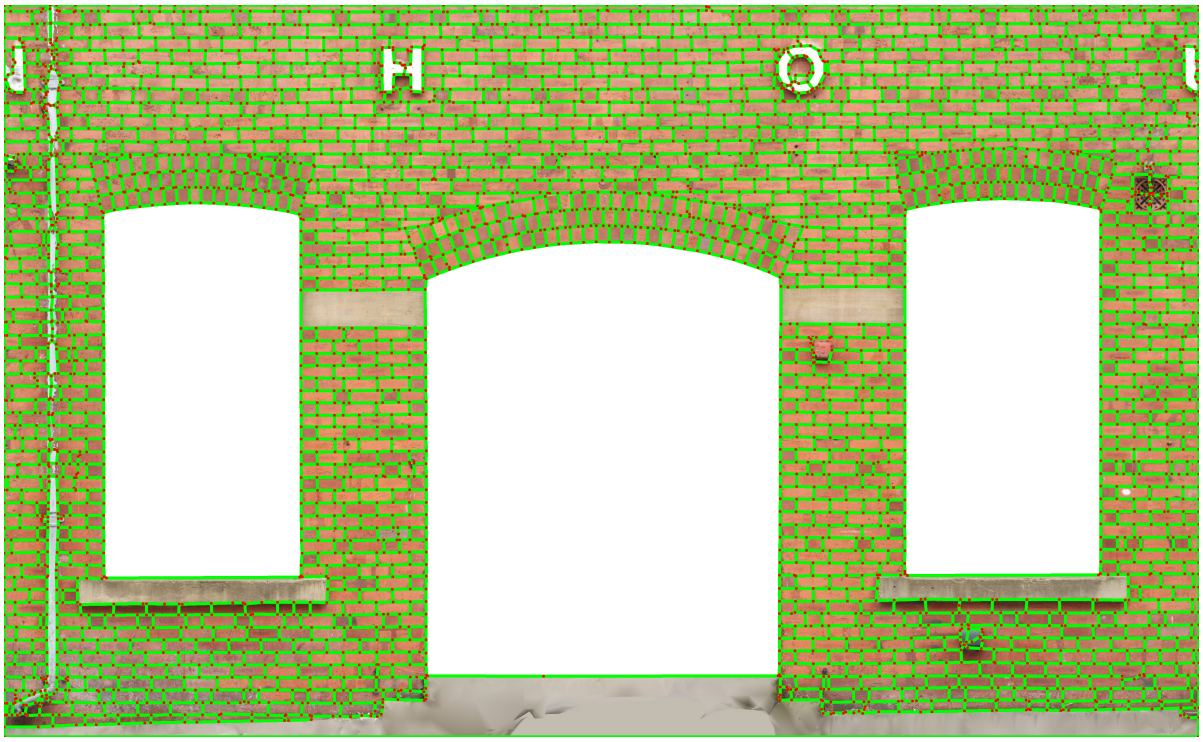


Fig. 4.31: Generalized lines on image (Red-Points: Vertices of the lines).

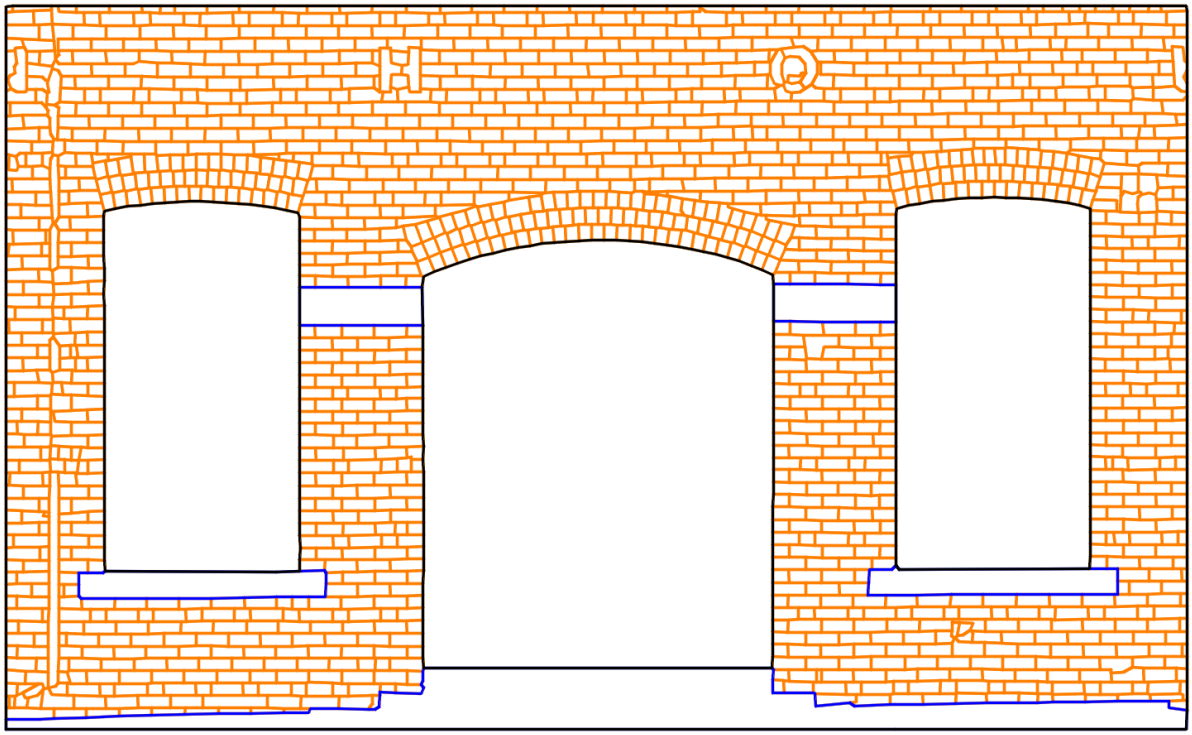


Fig. 4.32: AutoCAD drawing of the structure.

4.14. Conclusions

The work presented in this document demonstrates the potential of algorithmic solutions for evaluating and visually inspecting masonry structures. The novelty of the research is the complete combination of multidisciplinary approaches to identify and evaluate structural elements, on masonry structures, for the generation of geometrical models for digital twin or BIM applications. Those include but not limited to: Research in artificial intelligence for pattern-recognition and object-detection; image-processing for enhancement of object-detection and object-definition; and mathematical formulation for the object-definition, shape-enhancements, and shape-measurements. The major breakthrough is that the proposed approach offers a fully automated method to replace the manual process of visual inspection and assessment of masonry structures.

The methodology can be applied for large-constructions, and even identify elements not detected by the CNN-models. The complete workflow is aimed for health-monitoring using stable cameras to record progress of crack propagation and displacements. It can even be used to produce automatically numerical-models, to inspect changes to the structural-capacity on-the-fly. It has been demonstrated that the accuracy of the extracted geometry largely depends on the performance of the CNN models. In which case, the outcome can be improved highly by including, in the dataset, annotated-images of the structure that the health-monitoring system is installed for. Other uses include a quick and unbiased overview of the state of the structure, with precise measurements of large crack-patterns. The software can also be utilized for comparing crack propagation between various time periods and generating detailed records of different metrics, especially useful for inaccessible locations (i.e., by taking advantage of unmanned-aerial-vehicles).

The main limitation of the proposed system is the accuracy of the CNN-models used (section 4.4), which can be improved simply by enlarging the dataset. Especially with samples of similar cases of lower detection-rate (i.e., with areas affected by severe discolouration). Another limitation is the inability of the CNN-model to identify the brick-pattern behind foreground-objects (i.e., signs, pipes, vegetation, etc.). It is believed that the CNN algorithm has the capability to identify obstructed elements, since the CNN-architecture is able to identify local-features on the image (i.e., replicate the nearby pattern of brickwork). This may be possible by altering the dataset to include obstructed blocks, for valid regions only (i.e., not background). The last limitation of the workflow is that the procedure has been designed for 2D-ouptut. 3D-modelling is supported only for structures with equal depth along their plane (i.e., single-leaf walls). There are plans to translate the same methodology to 3D-plane. However, that would require an additional step to capture the structure from different locations/angles and identify the 3d coordinates of every pixel (i.e., with the assistance of a 3D reality-mesh; Fig. 4.2: a).

Future-work may include classification of additional defects using CNN, such as vegetation, spalling, discolouration, etc. Those can be used during the evaluation of the geometric-properties of defects (section 4.8), to provide more data for the structure. Or even to exclude specific defects from the numerical-model, if deemed appropriate (i.e., black stains registered as cracks). Another improvement includes further-adjustments of the geometry (section 4.10; step 9), to apply minor-corrections to the shape of blocks (i.e., adjust the verticality of interfaces that are almost vertical). Furter work may also involve the automatic-adjustment of material-properties, of a specific region of the numerical model, based on the deterioration-level detected. Other improvements and future-work include the amendment of the limitations of the workflow by following the suggestions provided in the relevant paragraph.

Acknowledgements

This work was funded by the EPSRC project "*Exploiting the resilience of masonry arch bridge infrastructure: a 3D multi-level modelling framework*" (ref. EP/T001348/1). The financial contribution is very much appreciated. Also, several photos obtained by engineers from Network Rail and Helifix,

UK, and were kindly offered to expand our masonry dataset. Therefore, their support in this study is highly recognised and appreciated.

5. Paper #4: Image2DEM: A geometrical digital twin generator for the detailed structural analysis of existing masonry infrastructure stock.

Loverdos Dimitrios.[1], Sarhosis Vasilis [1]

School of Civil Engineering, University of Leeds, LS2 9JT, Leeds, UK

Abstract

Assessing the structural performance of ageing masonry infrastructure is a complex task. Geometric characteristics and the presence of damage in masonry structures may influence greatly their rate of degradation and in-service mechanical response. Therefore, identifying approaches to assess the actual structural condition of these assets is vital. In the last ten years, advances in laser scanning and photogrammetry have started to drastically change the building industry since such techniques are able to capture rapidly and remotely digital records of objects and features in points cloud and image format. However, the direct and automatic exploitation of images for use as geometry in high fidelity models for structural analysis is limited. In this framework, the aim of this paper is to present the development of a software able to fully automate the "scan to structural modelling" procedure for the efficient and accurate structural assessment of ageing masonry infrastructure. "Image2DEM" is based on Python libraries with graphical interface. The images can be captured from DSLR (Digital Single-Lens Reflex) cameras, smartphones, or drones. The image selected is then imported to the program to detect and extract the masonry micro-geometry. The algorithm provides reliable detection using Artificial Intelligence. Convolutional Neural Networks (CNN) are used to identify the location of masonry units and cracks, with ~96% and ~80% accuracy, respectively. The geometry is extracted in the form of simplified lines to improve efficiency and reduce computational effort. The output is provided in DXF format for compatibility between different programs. Finally, the geometry extracted is converted to a numerical model for structural analysis. The proposed software has the potential to revolutionise the way we assess existing masonry infrastructure in the future.

5.1. Motivation and significance

Masonry infrastructure, such as bridges, viaducts and tunnels form a significant part of the UK's critical infrastructure stock; e.g. there are more than 70,000 masonry arch bridges which constitute over 40% of UK's bridge stock (McKibbins *et al.*, 2006). The majority of our masonry infrastructure is ageing, often well beyond 120 years, and showing significant signs of deterioration and damage. Weathering, demands of increasing load-intensity, axle-loads, and factors such as increased frequency of flood events due to climate change have introduced extreme uncertainty in the long-term performance of such infrastructure assets. Also, much of our masonry infrastructure has significant heritage and cultural value (e.g., the Grade II-listed Hungerford Canal Bridge, in Berkshire, England) and the UK has a policy to "retain and repair", rather than "demolish and replace". Failure of such infrastructure could lead to direct and indirect costs to the economy and society and hamper rescue and recovery efforts. For example, during the 2009 floods in Cumbria, three masonry arch bridges collapsed while nine were severely damaged, leading to nearly £34m in repair and replacement costs. The economic and societal impact were even larger, with increased travel time estimated to cost the economy almost £2m per week. In March 2017, approximately 200 tonnes of rubble fell on to the railway line when a masonry wall collapsed just outside Liverpool Lime Street station, which had the potential to crush or derail a passing train, with disastrous consequences. Therefore, there is an urgent need to better assess the in-service performance of ageing masonry infrastructure stocks, and to provide detailed and accurate data that will better inform maintenance programmes and asset management decisions.

Assessing the structural performance of ageing masonry infrastructure is a complex task. Previous research has clearly demonstrated that the assessment methods currently used by the industry are antiquated and/or over-simplistic (Phares *et al.*, 2004; McKibbins *et al.*, 2006). For example, for the assessment of masonry arch bridges, the Military Engineering Experimental Establishment (MEXE) method of assessment is still in use, dates back to the 1940s, has very limited predictive capability, and offers little scope for future enhancement. Also, although the primary focus of past research has been into the prediction of structural failure of ageing masonry infrastructure, prediction of the service load above which incremental damage occurs is now a key priority for infrastructure owners, who are under increasing pressure to provide transport networks which are secure and resilient.

Over the last three decades, significant efforts have been devoted to the development of numerical models to represent the complex and non-linear behaviour of masonry structures subjected to external loads (Lourenço, 1996, 2013; Asteris *et al.*, 2015; D'Altri *et al.*, 2020). Such models range from considering masonry as a continuum (macro-models), to the more detailed ones that consider masonry as an assemblage of units and mortar joints (micro-models; (Sarhosis and Sheng, 2014; Sarhosis *et al.*, 2014; Sarhosis, Garrity and Sheng, 2015; D'Altri *et al.*, 2018; Forgács, Sarhosis and Bagi, 2018; Sarhosis and Lemos, 2018; Erdogmus *et al.*, 2019; Sarhosis, Forgács and Lemos, 2019; Segura *et al.*, 2021)). Since ageing masonry infrastructure is typically characterised by low bond strength, cracking is often a result of the de-bonding of the masonry units from the mortar joints. Given the importance of the masonry unit-to-mortar interface on the structural behaviour of aged masonry structures, micro-modelling approaches (i.e., those based on Discrete Element Method) are better suited to simulating their serviceability and load carrying capacity. However, a vital aspect when modelling masonry structures based on the micro-modelling approach is the accuracy in which the geometry of the masonry structure is transferred in the numerical model. So far, the geometry of masonry infrastructure is captured with traditional techniques (e.g., visual inspection and manual surveying methods) which are labour intensive and error prone.

In the last ten years, advances in computer-vision, photogrammetry, and laser-scanning have started to drastically change the building industry. Especially since such techniques are able to capture rapidly and remotely digital records of objects and features in 2d-images (Sithole, 2008; Oses, Dornaika and Moujahid, 2014; Cluni *et al.*, 2015; Brackenbury and Dejong, 2018; Bal *et al.*, 2021) and point-cloud/3D-mesh formats (Volk, Stengel and Schultmann, 2014; Barazzetti *et al.*, 2015; Valero, Bosché

and Forster, 2018; Valero *et al.*, 2019; Andriasyan *et al.*, 2020; Bassier and Vergauwen, 2020). Even with the use of artificial intelligence for both 2d and 3d environments (Brackenbury, Brilakis and Dejong, 2019; Ibrahim, Nagy and Benedek, 2019; Spencer, Hoskere and Narazaki, 2019; Valero *et al.*, 2019; Kalfarisi, Wu and Soh, 2020; Dais *et al.*, 2021; Ergün Hatir and İnce, 2021). Although some work has been done in transitioning from point cloud to structural analysis models those are limited to continuum macro-modelling (Korumaz *et al.*, 2017; Bassier *et al.*, 2019; Rolin *et al.*, 2019; Funari *et al.*, 2021; Kassotakis and Sarhosis, 2021; Pepi *et al.*, 2021), or discontinuum macro-modelling (Hinks *et al.*, 2013; Tiberti and Milani, 2020b). Thus, there are still many challenges to overcome, especially regarding the discretisation of the numerical models generated. Additionally, a prominent factor in the assessment of masonry infrastructure is the impact of existing pathologies, such as deformations and cracks. According to Heyman (Heyman, 1966), geometric changes and existing damage in masonry structures can greatly influence their rate of degradation and in-service mechanical response. The lack of convenient tools that enable image and point cloud data to be readily transformed for use in a structural analysis model has hindered uptake in the engineering community.

In this framework, Loverdos and Sarhosis proposed a workflow to exploit images directly and automatically from ageing masonry infrastructure to generate geometrical digital twins which can be used for the structural assessment, inspection, and documentation (Loverdos *et al.*, 2021a; Loverdos and Sarhosis, 2023b). The procedure is as follows. Initially, images can be captured using DSLR or a smartphone or from a drone. Any image from any source can be used. Images that include any background (random objects, sky, ground, etc) are also compatible. Orthorectified images (with equal height/width scale) and good resolution are preferred but are not necessary. Then, the image selected is imported to "image2DEM" software to detect and extract the masonry micro-geometry. The algorithm provides reliable detection using Artificial Intelligence. Convolutional Neural Networks (CNN) are used to identify the location of masonry units and cracks, with ~96% and ~80% accuracy, respectively (Dais *et al.*, 2021; Loverdos and Sarhosis, 2022a, 2023a). Furthermore, background elements (non-masonry) are filtered out automatically. The geometry is then extracted in the form of simplified lines to improve efficiency and reduce computational effort and a "geometric digital twin" is created. Blocks, mortar, and cracks are assigned to different layers automatically. The mesh is optionally generated for blocks, mortar, and cracks. The output is provided in DXF format for compatibility between different CAD and BIM environment programs. Finally, the geometry extracted could be converted to numerical modelling software for the analysis of masonry structures. Furthermore, the mesh generated allows to investigate separation (loss of contact) during the analysis. To enhance simplicity, the elements are allocated to different groups depending on the layers assigned on the CAD file.

5.2. Software description

The main workflow of the software is simple (Fig. 5.1). Any image can be used to identify the micro-geometry of masonry (i.e., blocks, cracks, background), using artificial-intelligence and image-processing (Fig. 5.1; "*Detection*"). More specifically, blocks and cracks are detected using individual FCNs (Fully Convolutional Networks), while the background and other elements are detected using image-processing. Then, the geometry is extracted to a CAD file for documentation (Fig. 5.1; "*Documentation*"). Where each object is assigned to a separate layer automatically, based on the detection method used to generate the binary-image. Finally, the exported geometry is used to generate the numerical model of the structure (Fig. 5.1; "*Analysis*"), with a numerical-analysis software (such as UDEC, a discrete-element-method software). The numerical-model is used to evaluate the current state of the structure and estimate the maximum capacity.

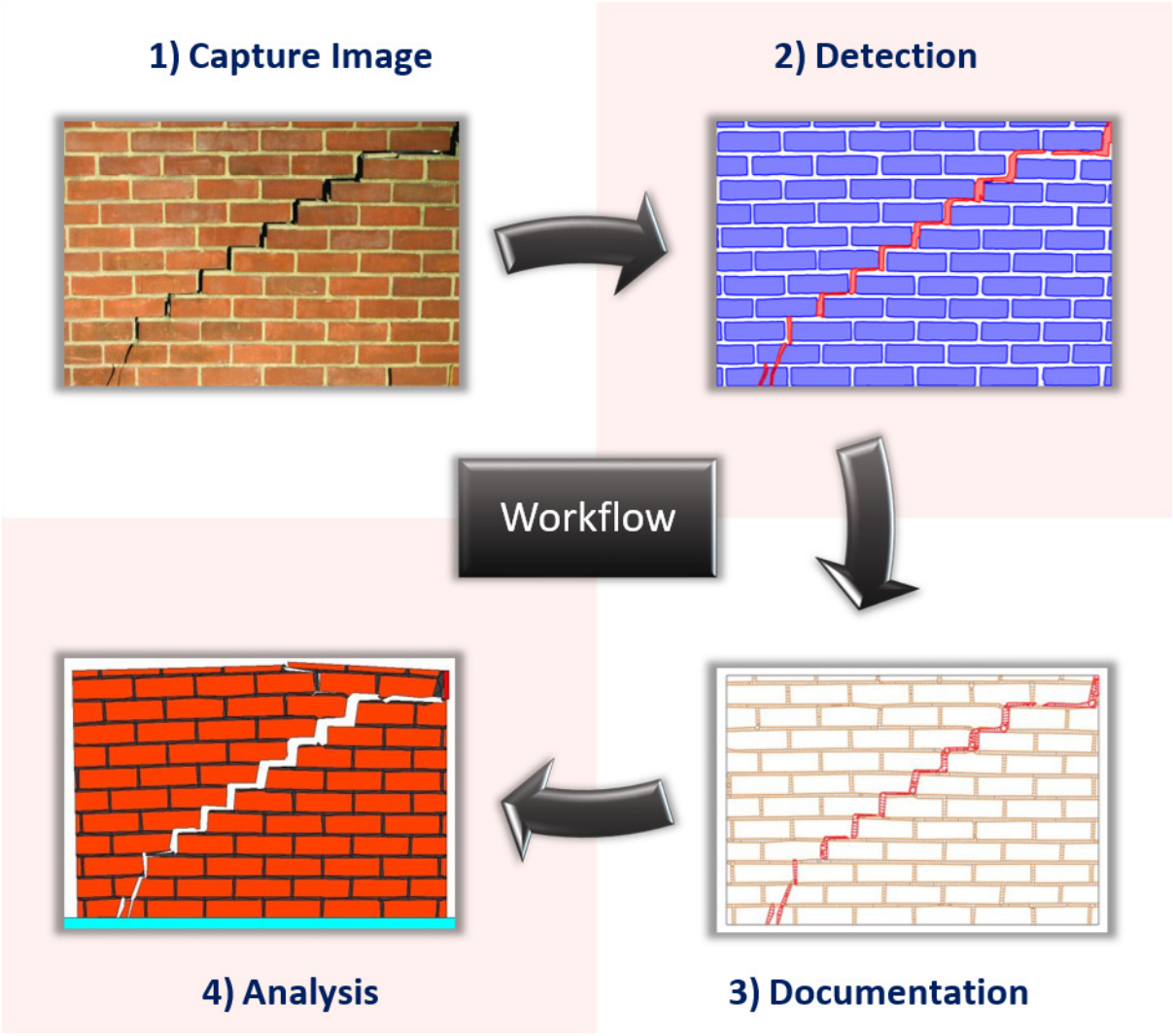


Fig. 5.1: Workflow of the algorithm

The graphical interface was developed to allow the user to easily select the modules that wants to run and allow the modification of the options of the software (Fig. 5.2). The GUI includes the ability to run all modules together, selected number of modules, or a single module (as seen on the left side of Fig. 5.2). The options are adjusted as an imported text-file (seen on the right side of Fig. 5.2). This allows to include many variables on the software that adjust the final output.

Regarding the main part of the software, there are multiple modules, each with a specific task. Those include: Functions to load and adjust the image (P1, P2); detect the micro-geometry of masonry (P5, P7, P9, P11); improvements to the binary-images using image-processing (P6, P8, P10, P12); damage-evaluation in terms of their geometrical properties (P13); feature-extraction of the micro-geometry (P14, P15, P16, P17); and finally, model generation for CAD documentation or analysis (P18). A separate module is used to convert the CAD file to UDEC geometry ("DXF to UDEC"), although the DXF file can also be used to create the numerical model in most software. More specifically, all the modules of the program are described in the table below (Table 5.1):

Table 5.1: Separated modules of the program.

P1	Load Image: Used to select an image with a file-browser.
P2	Adjust Scale: Adjusts the scale of the image programmatically.
P3	Detect Blocks: Detects blocks on image using CNN.
P4	Adjust Blocks: Improves the blocks-mask using image-processing.
P5	Detect Cracks: Detect cracks on image using CNN.
P6	Adjust Cracks: Improves the cracks-mask using image-processing.
P7	Detect Masonry: Detects the overall location of masonry by merging the detected blocks.
P8	Adjust Masonry: Improves the masonry-mask using image-processing.
P9	Detect Background: Identifies the background, either by image-thresholding (for white background), or by inverting the masonry-mask (for undefined background).
P10	Adjust Background: Improves the background-mask using image-processing.
P11	Create Structure: Creates the overall structure by combining all binary-images. It can be used to identify undefined elements, not detected by the block-detection (such as concrete beams).
P12	Adjust Structure: Improves the structure-mask using image-processing.
P13	Evaluate Cracks: Creates a CSV file with the geometric-properties of each isolated-crack (i.e., location, area, length, average-width, and coverage).
P14	Create Segmentation: Applies watershed-segmentation to isolate detected blocks.
P15	Adjust Segmentation: Adjusts the watershed-segmentation to include mortar and damage. Also, applies corrections to the watershed, to ensure the proper geometry-extraction.
P16	Extract Contours: Extracts the micro-geometry of masonry as polylines (using the watershed).
P17	Adjust Contours: Applies line-generalization to the extracted geometry (reducing the number of vertices of the polyline) and filters-out small elements with near zero-area. Furthermore, it adjusts the geometry to improve the general shape of the structure. Additionally, it generates the mesh, optimized for numerical-analysis, to investigate crack-propagation.
P18	Create Model: Creates the DXF file of the geometry of the structure, with every material assigned to individual layer. Furthermore, it provides multiple CSV files with the inner-location of every detected-object, separated by material.
-	DXF-to-UDEC: Convert the AutoCAD file to "fish" commands for analysis using UDEC. Separates materials to different groups (classifications).

Where P1 and P2 are relevant to the "*Capture*" step; P3-P12 are part of the "*Detection*" step; P13-P18 are part of the "*Documentation*" step; and finally, P18 and DXF-to-UDEC are relevant to the "*Analysis*" step (see Fig. 5.1). Those are further explained in the sections 5.3-5.6.

Furthermore, the adjustable-options, provided in the GUI (Fig. 5.2: right-side), can be used to improve the quality of the final-output. However, the default values are appropriate for almost every case and do not need any adjustment. The only exceptions are a few basic options, which modify the geometrical-model to meet different needs, such as the inclusion of damage (cracks), mortar (for detailed micro-modelling), and background (if the masonry doesn't cover the whole image). For example, if the structure is large, detailed micro-modelling is avoided since the model will be very complex and may cause the analysis to fail. In which case, the user should turn off the definition of mortar (*"P0_Use_Mortar=False"*) for simplified micro-modelling.

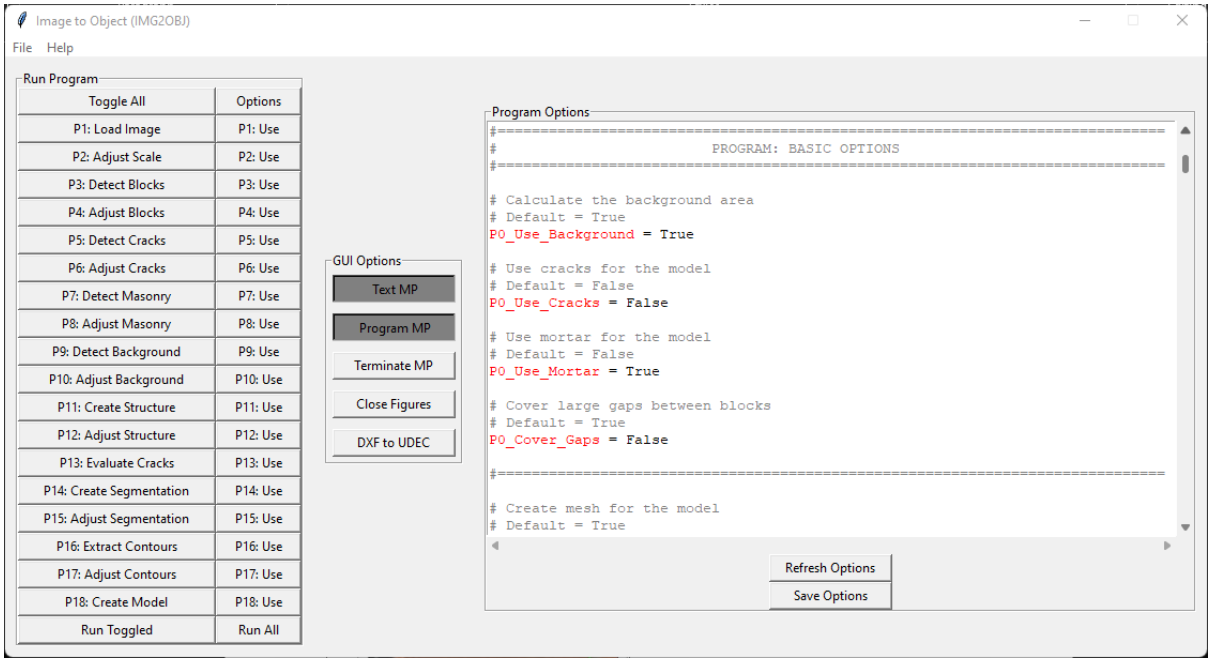


Fig. 5.2: Graphical Interface of the Software

5.3. Step #1: Input Image

Initially a representative image, of a masonry-structure, is captured and imported to the software using a browser-window typical to windows-applications (Fig. 5.2: P1). As it was mentioned earlier, any image can be used with the developed software. However, orthorectified images will provide higher accuracy in the "*Documentation*" and "*Analysis*" part of the software, due to equal-scale between the [x, y] axis and due to corrections to image-distortion caused by the camera lens (i.e., barrel effect). Orthorectified images can be produced using photogrammetry software, such as "*Context Capture*".

Additionally, the scale of the image is an important aspect of the process for multiple reasons. Firstly, it allows to automatically adjust most variables and thus, minimize user interaction with the GUI. An example of the automated adjustment of a variable is the resize-value of the image, before passing through the CNN networks. But more importantly, it allows the acquisition of the true dimensions of the structure. Those are used for the geometrical-model generation and for the evaluation of the geometric-properties of detected cracks. The scale of the image is acquired programmatically for convenience (Fig. 5.2: P2). More specifically, the scale is acquired by selecting two points on the input-image and providing the distance between them (Fig. 5.3).

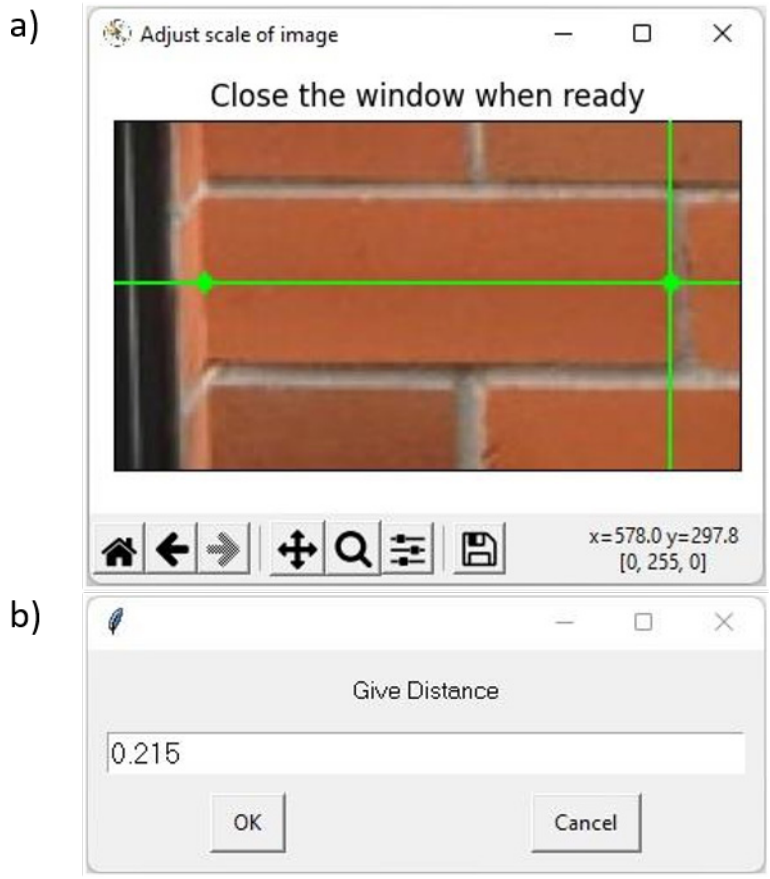


Fig. 5.3: Image scaling (in meters)

5.4. Step #2: Object Detection

Blocks on the image are detected using CNN for reliable detection with a validation accuracy equal to 96.86% and validation F1-score of 96.3% (Fig. 5.2: P3; Fig. 5.4: a2). Similarly, the cracks are also detected using CNN with a validation F1-Score equal to 79.6% (Fig. 5.2: P5; Fig. 5.5: b). Both models were trained with images of typical masonry structures (i.e., not rubble), bonded with mortar.

The overall location of masonry is detected automatically from the detected blocks (Fig. 5.2: P7), using simple image-processing functions (i.e., image-closing to combine the detected-blocks). The background is detected based on user-preference (Fig. 5.2: P9) and can be acquired either from the masonry-mask (Fig. 5.4: a3), or the white section of the image (Fig. 5.4: b3).

The structure combines all the binary-images (Fig. 5.2: P11; blocks, cracks, masonry, background) and has the capability to identify undetected-elements automatically (Fig. 5.4: b4). More specifically, the areas that do not belong to either background or masonry are assigned as undetected-elements. However, the detection of other structural-elements can only be applied for images with white-background, no-background at all, or a custom-background mask. This is due to the reason that the masonry-mask must be different from the background to identify undetected-elements.

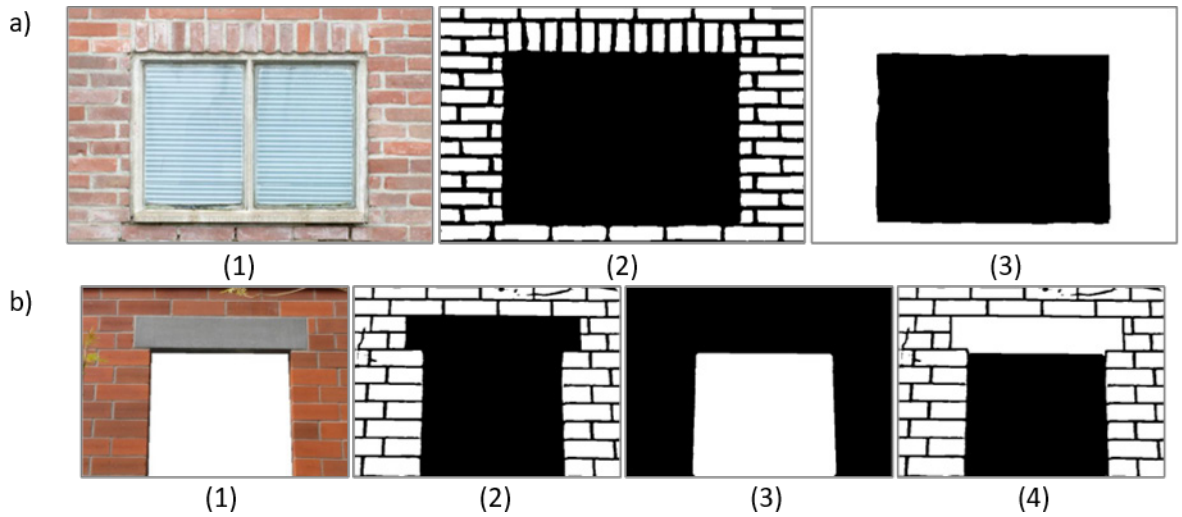


Fig. 5.4: a] Detect Background: 1) Original Image; 2) Blocks; 3) Masonry; b] Detect Other Elements; 1) Original Image; 2) Blocks; 3) Background; 4) Final Structure

The adjustment of the binary masks is using the same module with different arguments (Fig. 5.2: P4, P6, P8, P10, P12). They mostly remove small elements from either the background and/or foreground, based on the object-area (converting the number of pixels to scaled-area). Thus, ignoring the extraction of excessively small-elements that are possibly labelled-incorrectly. Those adjustments improve the output considerably. They are applied after each mask-detection to avoid repetition of the detection (Fig. 5.2: P3, P5, P7, P9, P11). Especially for blocks and cracks since their detection is slower due to the application of a CNN model (Fig. 5.2: P3, P5).

5.5. Step #3: Documentation - Geometrical Model

The detected cracks are used to acquire the geometric properties of each detected-defect (Fig. 5.2: P13). Each crack is isolated and measured individually using image-processing (Fig. 5.5). The calculation of the crack metrics is precise, assuming the accurate output of the crack-detection module. Those metrics can be used to assist engineers with the visual inspection of masonry structures.

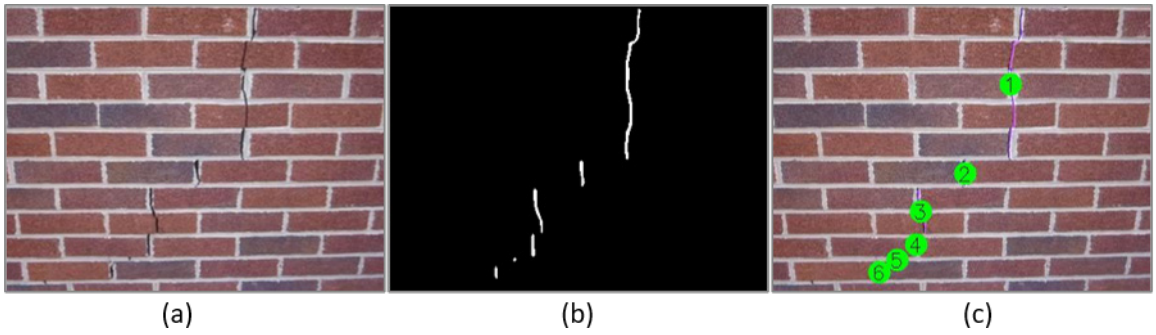


Fig. 5.5: Crack measurements; a) Image; b) Detected cracks; c) Overlay and labels of cracks.

The acquired crack-metrics are the location, area, width, length, and coverage (Table 5.2). The coverage refers to the percentage coverage of the crack-area over the area of all defects or the overall masonry-area. The CSV file includes both scaled (i.e., in milometers) and unscaled (in pixels) values. However, only the scaled values are provided here due to size-limitation of the document width.

Table 5.2: Geometrical-properties of detected-cracks (the location starts from the top-left side of the image).

Label - (No)	Location [xmid,ymid] (pixels)	Area Scaled (mm^2)	Length Scaled (mm)	Width Scaled (mm)	Coverage Cracks (%)	Coverage Masonry (%)
1	[251, 82]	2516	376	7	59.507	0.423
2	[203, 176]	432	52	8	10.211	0.073
3	[156, 216]	700	107	7	16.549	0.118
4	[151, 251]	372	48	8	8.803	0.063
5	[131, 267]	45	3	17	1.056	0.008
6	[112, 280]	164	25	7	3.873	0.028
	Total:	4229	607	53	100	0.71

The detected geometry of masonry from the binary images is used to generate the geometrical model (Fig. 5.2: P14-18; Fig. 5.6). Initially, watershed-segmentation is used to isolate every individual block (Fig. 5.2: P14). The segmentation is then adjusted to include mortar and damage, but also to test that every separated-segmentation is assigned a unique-label (Fig. 5.2: P15). The geometry is then extracted in the form of polylines and is scaled to the real-dimensions for the precise documentation of the structure (Fig. 5.2: P16). The geometry is also simplified to reduce the number of vertices, so that it retains the minimum number of vertices that best describe each object (Fig. 5.2: P17), using a generalization algorithm (Fig. 5.6: a & b). The accuracy of the generalization is adjustable. The mesh is also generated to allow separation between objects and investigate crack-propagation, during the numerical analysis (Fig. 5.2: P18; Fig. 5.6: c & d). Finally, the generated geometry is exported to DXF. Both simplified micro-modelling (Fig. 5.6: b) and detailed micro-modelling (Fig. 5.6: d) are supported for the model-generation.

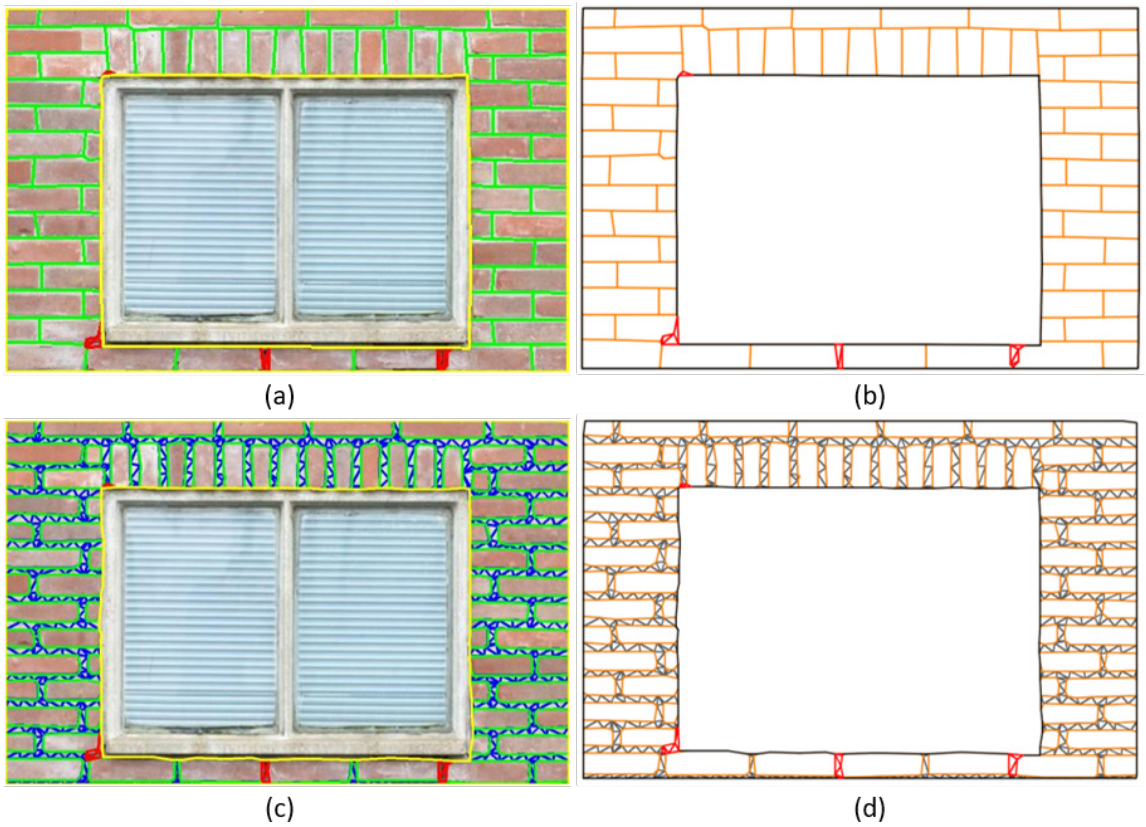


Fig. 5.6: geometrical model; a) Generalized-lines for simplified micro-modelling; b) AutoCAD drawing (simplified micro-modelling); c) Generalized-lines for detailed micro-modelling; d) AutoCAD drawing (detailed micro-modelling).

5.6. Step #4: Analysis – Numerical Model

The final step is the generation of the numerical model of the structure in 2D (Fig. 5.2: DXF to UDEC; Fig. 5.8: b; Fig. 5.10: b). For that the geometry, location, and classification of every object is required to define the model in UDEC. The geometry is defined by the DXF file (Fig. 5.8: a; Fig. 5.10: a). The location is defined for every object separately and is considering an inner point within the area enclosed by its individual element. The location is also used to define the class of every object in the numerical model. This is required because in UDEC every individual element is generated by dividing an existing large-block into multiple parts. Thus, the initial classification (of the main block) is irrelevant.

The classification of every object in the numerical-model is made using the inner-location of every object, except for the background where the segmentation-inner location is used instead. The reason why the segmentation-inner-location is used for the background is because the polyline of the outer-background includes all the other objects as well (blocks, cracks, mortar, mesh-elements, etc.). Thus, using the segmentation location avoids the incorrect classification of an unspecified-object enclosed within the outer-background area.

The geometry extracted is limited to 2D analysis in general. A simple 3D model can be generated under the assumption that every drawn-object has equal depth. Although, automatic generation of 3D models would require a lot of manual-effort to adjust the model for discrete numerical-analysis. Mostly because the inner-materials, block pattern, etc, cannot be detected from image and point-cloud data (i.e., backfill, multi-leaf walls). However, regarding the use of the software for simple documentation, multiple faces can be extracted separately.

5.7. Output-Files and Folder-Structure

The output folder-structure is divided into multiple-sections. The input-data used during the program-execution are stored in the "Basic" folder. The user-adjustable options are stored in text-format in the "Data" folder. The "Images" folder includes figures of all the processes, aiming to identify issues with any part of the workflow. Additionally, the folder-structure includes an "Override" folder, which can be used to copy and manually-adjust the binary-images to improve the final-result. More specifically, any image with the keywords "Blocks", "Cracks", "Masonry", "Structure", will replace the original-input during the program-execution.

The "Results" folder contains the final-output from the crack-measurements and geometry-extraction (Fig. 5.2: P13 & P18). Those include the DXF file of the geometry, the CSV file of the crack-measurements, and CSV files of the location of each element (separated by material and detection method). If the companion-program is used (Fig. 5.2: "DXF to UDEC"), the "Results" folder will include the TXT files of the complete-geometry in fish-commands (scripting-language of Itasca-software).

The location of every object is extracted individually per class, in a csv file format. The location-classifications are the following: blocks, block-mesh, mortar, mortar-mesh, damage, damage-mesh, and background. The location-types extracted are: Inner (inner location of the closed-polyline), centre (centroid of the closed-polyline), segmentation (inner location of the watershed-segmentation), and pixel (pixel location in scaled coordinates; for mortar/damage only).

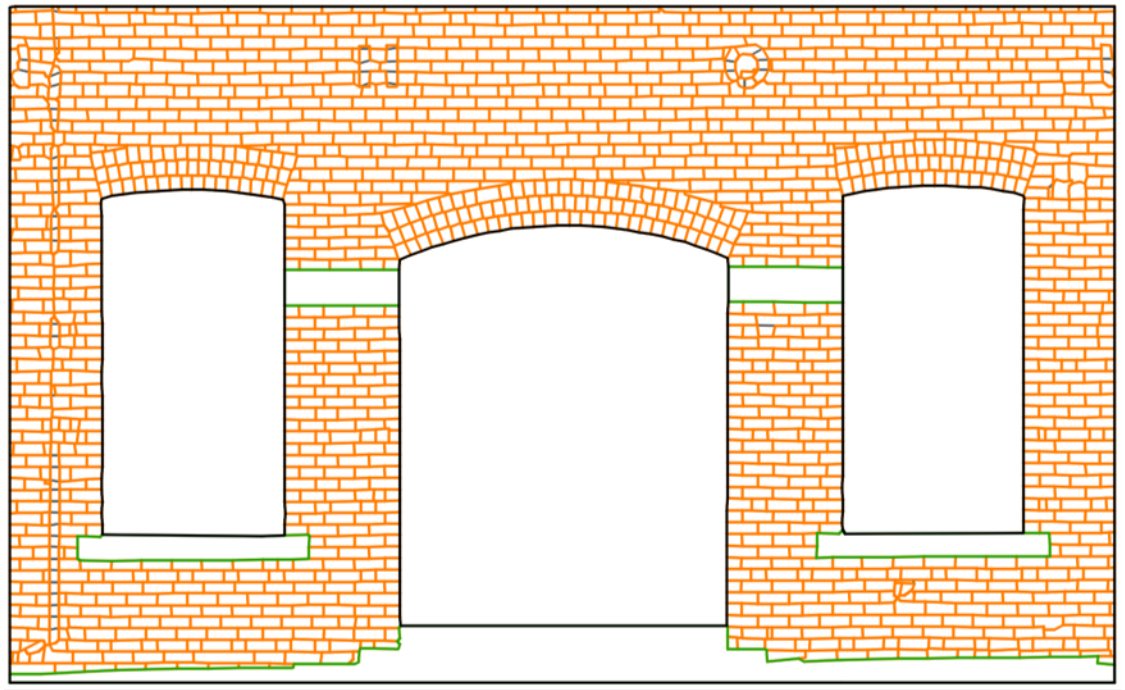
5.8. Illustrative example #1

The following example is a section of the façade of the "Town House" in Leeds, UK (Fig. 5.7: a). The input is an orthorectified image generated using photogrammetry software (Fig. 5.7: b). The specific case was selected because it contained both arch and straight brick-pattern. The software produced an excellent drawing (Fig. 5.8: a), except for locations covered with foreground objects. More noticeable near the pipe on the left side of the image. The pipe could be filtered out, within the photogrammetry software, if more images were acquired from the side of the object. In which case, the bricks behind the pipe could be reconstructed fully. However, minor corrections can be applied to the drawing directly (Fig. 5.8: a (grey lines)); or alternatively, by manually-editing the binary-output of the block-detection algorithm (Fig. 5.2: P4), which is then placed in the "Override" folder to be used instead of the initial-output. Finally, the 2D geometry was then transferred to UDEC (Fig. 5.8: b), to allow the numerical evaluation of the structure using the discrete element method. The complete procedure took ~915 sec (for modules P1-P18). The largest amount of time was required by P15 (387 sec). The time recorded is for an image-resolution of 4239x2594 pixels; Image-Scale of ~0.002 meters/pixel; and 4,714 polylines that formed 1,577 individual blocks. The computer used is a laptop with i7-9750H CPU, RTX-2060 GPU with 6gb VRAM, and 16x2gb RAM. The computational-time can be largely optimised by allowing multi-core calculations on developed-algorithms used by the software. Currently, most developed-algorithms are only allowed single-core calculations.

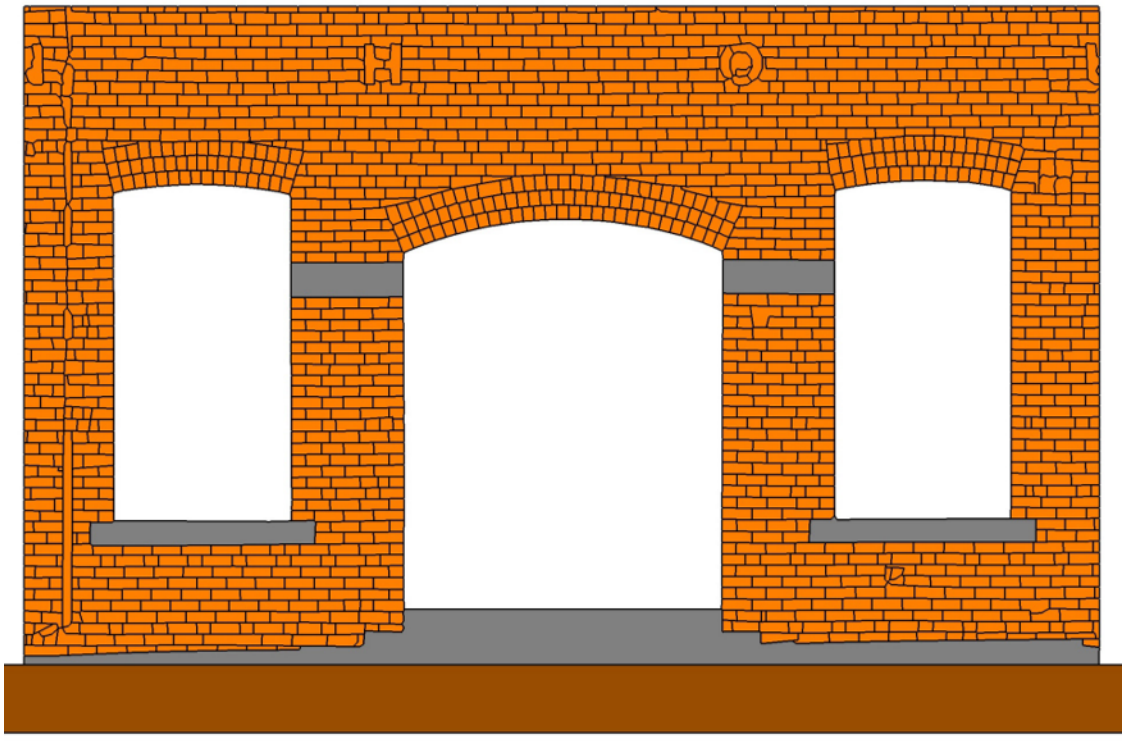
(a)

(b)

Fig. 5.7: a) Sample of x173 images of the "Town House"; b) Orthorectified image of the Town House, Leeds, UK (Loverdos and Sarhosis, 2023b).



(a)



(b)

Fig. 5.8: a) Output of software - AutoCAD Drawing: Grey-lines were drawn manually, over problematic areas, to ensure in-plane separation on the horizontal-axis during the numerical-analysis; b) Numerical model in UDEC (created using the supplementary program "DXF To UDEC").

5.9. Illustrative Example #2

The following example is a masonry arch-bridge experiment that is being conducted at the laboratory of the University of Leeds (Fig. 5.9: a). The orthorectified image (Fig. 5.9: b) was acquired using photogrammetry software, after it produced a proper 3D model (reality-mesh). It can be observed

that the stair (presented in the original images; Fig. 5.9: a) was filtered-out from the 3D mesh (Fig. 5.9: b), which allows the detection of covered-bricks. Then the developed software was used to generate the AutoCAD drawing (Fig. 5.10: a) and subsequently, the numerical model in UDEC (Fig. 5.10: b). The block-detection demonstrates excellent results, especially considering the stained surface of the bricks (more noticeable in the middle-right side). The numerical model considers the discrete element method, and more specifically simplified micro-modelling, to investigate crack propagation in later stages of the experiment. The complete procedure took ~391sec (for modules P1-P18). The time recorded is for an image-resolution of 3,840x1899 pixels; Image-Scale of ~0.002 meters/pixel; and 2,290 polylines that formed 766 individual blocks.

(a)

(b)

Fig. 5.9: a) Sample of x1,217 images of the "Arch Bridge" used to generate the 3D mesh; b) Orthorectified image of masonry arch-bridge (laboratory experiment in UoL).

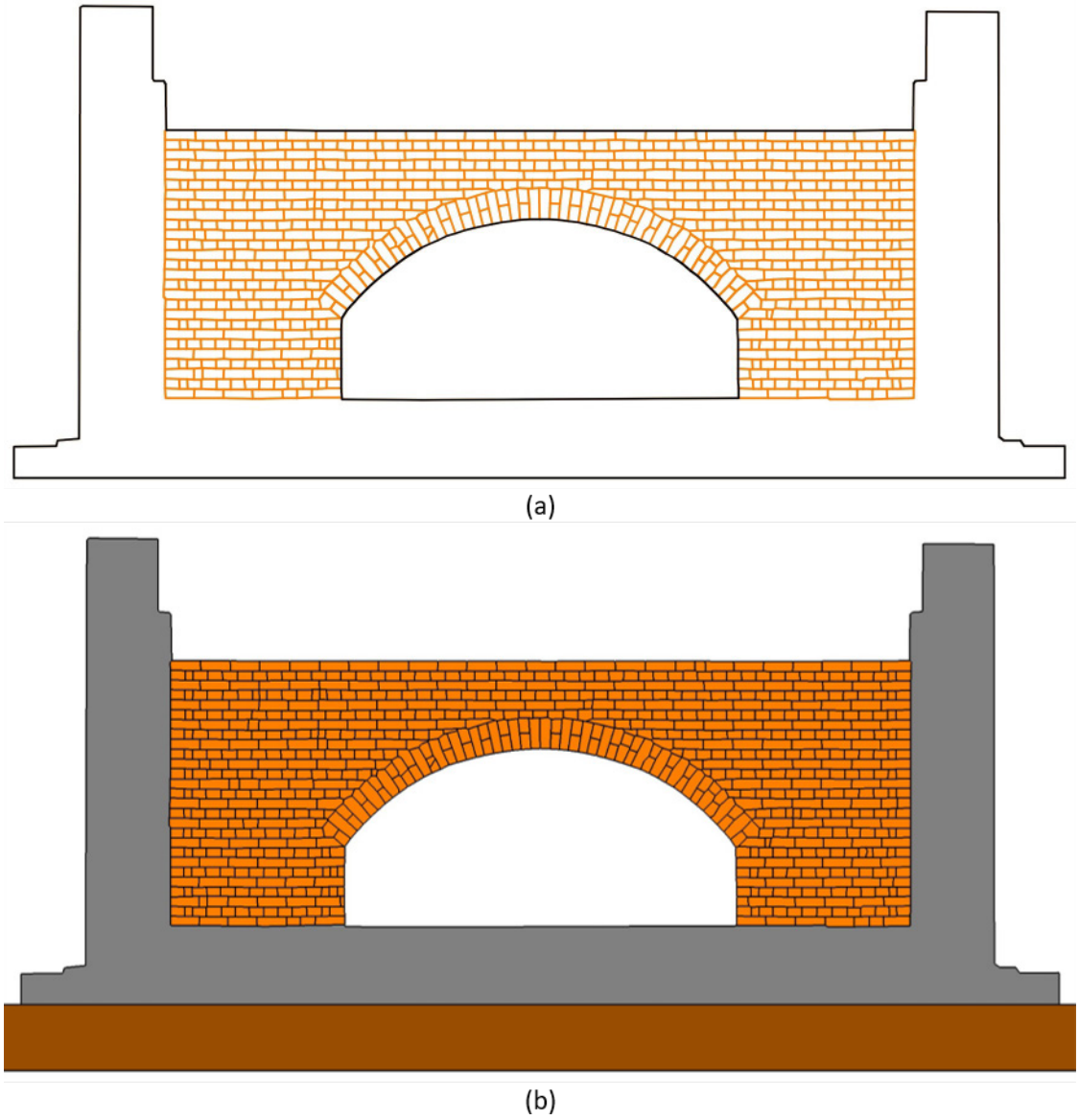


Fig. 5.10: a) Output of software - AutoCAD Drawing; b) Numerical model in UDEC (created using the supplementary program "DXF To UDEC").

5.10. Impact

Network Rail (the UK railway authority) acknowledges that that current inspection methods are highly simplified and often not capable of reliably distinguishing between structures that can sustain further loads and those which cannot (Brown, 2024). The impact of "Image2DEM" toolkit is to improve the structural analysis of our "as is" existing masonry infrastructure. "Image2DEM" allows a non-expert user to generate the geometry and mesh required for the development of high-fidelity numerical models such as the ones with DEM and FEM starting from images of the structure under consideration. The toolkit considers apart from the segmentation of masonry units, the damages in the structures such as cracks and distortions originated due to ground subsidence as well as irregularities or missing bricks. Reliable inspection of infrastructure leads to more informed maintenance schemes, and potentially reduced unnecessary repair and strengthening interventions, which contributes significantly towards the UK's "Net Zero" strategy (BEIS, 2021).

5.11. Conclusions

The “Image2DEM” software is able to harness current developments in remote surveying methods and couple them with algorithms developed in Python based on Artificial Intelligence and Machine Learning to fully automate the “*scan to structural modelling*” procedure for the efficient and accurate and detailed structural analysis of our ageing masonry infrastructure stock. According to the method, first, images captured from smartphones or DSLR cameras are uploaded into our “Image2DEM” software. Using computer vision and Artificial Intelligence (AI) techniques, it is possible to detect masonry units (e.g., bricks, blocks) and cracks automatically. The “as is” geometry of the masonry structure generated, can then be extracted in the form of simplified lines (x, y coordinates) in a DXF format. Finally, DXF files can be used in numerical analysis software for their structural assessment. The ad-hoc graphical tools developed are able to segment individual bricks in a masonry structure and mesh the mortar between them for the structural analysis. This transition, from the physical to the digital environment, has the potential to provide a better understanding of the "as-is" condition of our existing masonry infrastructure and revolutionize the way structural analysis is conducted in the industry. Using efficient and accurate estimation of the “as is” structural condition of ageing masonry infrastructure, we are able to provide detailed and accurate data that will better inform maintenance programmes and asset management decisions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The project has received partially funding from the EPSRC (EP/T001305/1). "Exploiting the resilience of masonry arch bridge infrastructure: A 3D multi-level modelling framework". The financial support is highly acknowledged.

6. Paper #5: Pixel-level block classification and crack detection from 3D reconstruction models of masonry structures using CNN.

Dimitrios Loverdos[a], Vasilis Sarhosis[a*]

[a] University of Leeds, School of Civil Engineering, Woodhouse Ln, Leeds LS2 9DY, United Kingdom

Abstract

Inspection and documentation of existing masonry-structures is a tedious and costly procedure. It usually requires Health and Safety reporting since the process usually involves an inspection engineer to access a structure that may be in close proximity of a busy railway line or road. Furthermore, it relies solely on the experience of the engineer that performs the inspection. However, nowadays, drones can be used to help engineers gather far more detail than conventional methods. Drones can be flown close to structures and get into areas that might be difficult or impossible to reach using traditional methods. The digital data gathered by drones can be used to generate very detailed 2D and 3D models of bridges using photogrammetry. This paper presents a methodology for the automatic block and crack classification and segmentation of three-dimensional models of masonry structures. The classification is acquired automatically using CNN (Convolutional Neural Networks) on images captured directly from the 3D model. Hence, achieving high-accuracy with low-effort, especially when compared to alternative-methods that consider the classification of point-clouds. At present, the available classifications are blocks, cracks, mortar, and unspecified/other-elements. However, any other-class can be incorporated as long as the CNN-model is provided, which allows the expansion of the use-case of the algorithm to encompass any structural-material and defect. Lastly, this approach allows the manual-annotation of any classification, to allow the engineer to mark areas-of-interest that were not detected by the CNN-models (i.e., cracks). Thus, enabling the industrial-use of the project, by the means of a simplified annotation tool of any structure in 3D environment. In addition, the suitability of three commercial software (i.e., Context Capture; Metashape; and Reality Capture) for photogrammetry are examined with respect to their suitability for alignment, 3D mesh, and coloured-textures.

Keywords: masonry, image processing, documentation, inspection, artificial intelligence, convolutional neural networks, reality mesh, 3D mesh.

***Corresponding author:** Prof. V. Sarhosis, University of Leeds, email: v.sarhosis@leeds.ac.uk

6.1. Introduction

Although one of the oldest construction methods, masonry is still commonly used due to its reliability and sustainability that it offers. Masonry structures consist of residential or commercial buildings and infrastructure such as masonry arch bridges. In fact, a large portion of the existing infrastructure in the UK is consisting of masonry arch-bridges (Sowden, 1990; McKibbins *et al.*, 2006), many of which were constructed well beyond the 100 years period and form part of our cultural heritage. Frequent inspection of those structures is important to ensure their serviceability, which is often a manual process (Phares *et al.*, 2004; Eaton, Edwards and Crapper, 2014). A process that is expensive in terms of workforce, financial, and time requirements. For the aforementioned reasons, improvements in automation and simplification of visual inspection have acquired large scientific and commercial interest (Spencer, Hoskere and Narazaki, 2019).

Recently, the use of terrestrial laser scanning using LiDAR and structure from motion using images have been successfully used to create accurate digital records in the form of 3D point-cloud or reality-mesh of existing structures at millimetre or even sub-millimetre level of accuracy (Altuntas, Hezer and Kırlı, 2017; Historic England, 2017, 2018; Kassotakis and Sarhosis, 2021). Furthermore, the resultant point-cloud can be used to identify the location of specific structural elements, using artificial-intelligence (e.g., (Bassier and Vergauwen, 2020) where classification of wall-objects from point-cloud data is presented). However, the manual annotation and training of machine-learning requires significant efforts due to the availability of the data and the complexity of annotation of point-clouds. Alternative methods consider the classification of defects on the reality-mesh itself (Kalfarisi, Wu and Soh, 2020). More specifically, the images that are used for the generation of the point-cloud and textures. However, this approach does not allow the manipulation of the 3-dimensional classification since it is applied on the texture of the model. Although not fully developed, those attempts have the capability to automate a large part of the visual-inspection, documentation, and assessment of existing structures. Future developments may increase the accuracy of the 3D classification with the inclusion of most structural types and materials and thus, improve applicability in commercial applications.

A simpler approach is the use of semantic-segmentation, using deep-learning, for the classification of image-data, such as CNN and FCN, that provide a reliable solution to the automatic annotation of selected targets (Ronneberger, Fischer and Brox, 2015; Chen *et al.*, 2018; Spencer, Hoskere and Narazaki, 2019). Those approaches have already been applied for the classification of the detailed geometry of masonry fabric, such the classification of masonry units (Ibrahim, Nagy and Benedek, 2019; Ergün Hatir and İnce, 2021; Loverdos and Sarhosis, 2022a) and defects in masonry (Chaiyasarn *et al.*, 2018; Ali, 2019; Brackenbury, Brilakis and Dejong, 2019; Wang *et al.*, 2019; Dais *et al.*, 2021). Due to the generation of photorealistic models from photogrammetry/LiDAR, the classification of the three-dimensional (3D) mesh may even consider the application of semantic-segmentation on image-data instead of the classification of the point-cloud (Kalfarisi, Wu and Soh, 2020). The advantage of image-based methods is mainly the abundancy of image-data and the simplicity of the generation of annotated-data for training purposes. However, the classification of 3D data requires the development of a post-processing method to transfer the detection using semantic-segmentation to the 3D object.

The classification of structures from point-cloud or reality-mesh has applications on the generation of BIM models (Volk, Stengel and Schultmann, 2014; Barazzetti *et al.*, 2015; Andriasyan *et al.*, 2020), for direct use with commercial packages. Additionally, it may be used for the generation of geometrically-accurate three-dimensional numerical models (Kassotakis and Sarhosis, 2021), which are currently limited to either continuum macro-modelling (Korumaz *et al.*, 2017; Bassier *et al.*, 2019; Rolin *et al.*, 2019; Pepi *et al.*, 2021) or discontinuum macro-modelling (Hinks *et al.*, 2013; Kassotakis *et al.*, 2020).

This paper aims to present a methodology for generating a classified point cloud and mesh elements using a textured/coloured reality mesh model of a masonry-structure as input. Also, the classification

is image-based to take advantage of the availability of image-data from various sources (i.e., drone surveys, smartphone pictures), and existing CNN models already trained to such records.

6.2. Workflow of the 3D classification

The main-concept of the workflow is to apply object-detection on images, captured from the 3D-model directly, and generate a 3D-point-cloud with classified points. The main advantage of this method is that it avoids the classification of 3D points cloud. Thus, achieving much higher accuracy with less effort. Especially when considering the scarcity of points cloud data compared to image-data, and the complexity of classifying a point-cloud in 3D. The complete workflow is provided below:

- **P1**: Load Input (3D-Mesh with textures)
- **P2**: Image Rendering from 3D-Mesh
- **P3**: Block Detection on 2D Renders (CNN)
- **P4**: Image Processing on Blocks
- **P5**: Crack Detection on 2D Renders (CNN)
- **P6**: Image Processing on masonry cracks
- **P7**: Segmentation (All Classifications)
- **P8**: 3D Point Extraction from Segmentation
- **P9**: 3D Point Filtering
- **P10**: Damage Evaluation

The input to the software can be any mesh of masonry structure, with either textures or coloured-vertices (P1). The renders are captured directly from the 3D-model by several imaginary-cameras (around the object), to cover the complete area (P2). Then, object-detection is applied on the image-renders to identify originally the location of blocks and cracks using CNN (P3 & P5). The output of the object-detection is filtered using image-processing to improve the outcome (P4 & P6). Every classification is combined to a single segmentation (P7). The final-segmentation includes blocks, cracks, mortar, and other elements (concrete, lintel, windows, etc). Then, the 3D coordinates of every classified-pixel, of the segmentation, is extracted to form the initial point-cloud (P8). The points are then filtered to improve the final output (P9). Additionally, the 3D-points that form the cracks are meshed to evaluate their 3D geometrical-properties (i.e., length, width, area, centroid; P10). The complete list of outputs is provided below:

- Output #1: Classified Point-Cloud (blocks, cracks, mortar, others)
- Output #2: Mesh of Cracks (for visualisation and measurement)
- Output #3: Crack Evaluation (spreadsheet of crack-geometry)

6.3. Input: 3D Model (P1)

The input to the sequence is any 3D model of masonry, with textures. Untextured models can also be used if they are coloured (coloured vertices). However, a good resolution texture is preferable since it provides the highest visual-quality for the file-size. A realistic and accurate 3D model of a structure can be acquired using either photogrammetry or laser-scanning with LiDAR. The 3D models, presented in this study, were developed using photogrammetry. Mainly due to the accessibility it offers, since it only requires a camera, and as a low-cost alternative to laser-scanning.

The main model was generated using 1,217 images captured with a “Xiaomi Pocophone F2 pro” with 64MP camera (Fig. 6.1). The overlap of the captured images was aimed to be at least 50%. Regarding the generation of the 3D model, 3 different photogrammetry software were tested to decide which one to be used. The photogrammetry software used in this study are: “Context Capture” from Bentley

Systems, “Metashape” from Agisoft, and “Reality Capture” from Capturing Reality. All programs provided excellent results but required different levels of user-interaction to successfully reconstruct the complete scene in 360°. The computer used for the alignment and reconstruction of the 3D mesh had I7-9750h CPU, Nvidia 2060 RTX GPU, and 2x16 GB of RAM.

Fig. 6.1: Sample images used for photogrammetry of the full-scale masonry arch bridge constructed in the laboratory.

6.3.1. Comparison of the Different Photogrammetric Software

“Context Capture” involved minimal manual-manipulation from the user to create the final-mesh, including texturing. Also, it provided an excellent mesh and decent textures of the masonry arch-bridge. A total of 4 control-points were assigned manually (in multiple images), to improve the alignment and scale the structure (Fig. 6.2a). The alignment required 66 minutes and the first reconstruction 256 minutes in medium accuracy (for a total of 322 mins, or 5:22 hrs). However, for larger models, “Context Capture” divides the final-mesh into tiles, since each tile must fit a pre-defined ram-size (i.e., 16gb by default). The specific model was divided into 115 tiles of 16 GBs ram-usage (Fig. 6.2c). Rarely, this may cause some connection issues with the mesh-tiles, which probably could be resolved with manipulation of the model-options and with repetition of the procedure (Fig. 6.2: b). However, the main issue is that since the model is divided, the merging of the individual object requires large manual-effort, and the final-result is excessive in size, especially due to the large number of texture-images. That prohibits the visualisation and usage of the model as a single entity, in a simple manner, outside of the Bentley ecosystem. Bentley programs bypass this limitation by the use of a “Level-of-Detail” tree, for visualisation and use of the complete-mesh.

“Metashape” required the highest manual-manipulation of the 3 programs to acquire the final-model. Although, it is the only program of those tested that allows the direct-manipulation of the tie-points and dense-point-cloud (i.e., deletion/filtering of points), it also provided the most tools to manipulate the output of every step of the procedure (alignment, reconstruction, etc.). The alignment also took advantage of four manual selected control points, to improve the result and scale the structure. Additionally, thirteen manual-markers were placed to provide the best alignment possible (including the 4 control-points), without of which, multiple components were generated, where none was complete. The alignment took 188 minutes in medium accuracy (112 mins matching-time, 76 mins alignment-time; Fig. 6.3a). The reconstruction lasted 132 minutes and produced 75.4 million faces (101 mins depth-map generation, 31 mins reconstruction). Finally, the texturing required 255 minutes (for a total of 575 mins, or 561 mins). The final mesh and textures have exceptional visual-quality (Fig. 6.3c), although, that comes at the cost of a large amount of additional-time/manual-labour to achieve

the presented result, mainly due to the required number of manually-placed markers and the false-surfaces it created due to small-misalignment of certain images (Fig. 6.3: b).

The last of the photogrammetry software tested is “Reality Capture”. The default options of the software are sufficient for most user cases, which makes it a simple program to use, where rarely is any special-manipulation of the software options is needed (assuming a healthy set of images is provided; i.e., good overlap, resolution, coverage). Regarding the procedure, four control points were assigned to improve the alignment and scale the structure, on multiple images. The alignment initially generated 2 components, which were merged with the help of the 4 control-points (Fig. 6.4: a). The alignment required 37 minutes and the reconstruction 145 minutes (71 mins depth-map calculation, 74 mins for meshing; Fig. 6.4b). An additional 59 minutes were necessary for the manipulation of the mesh for smoothing and simplification, which was not recorded for the other software. Colouring of the dense point-cloud and texturing of the mesh (Fig. 6.4c) lasted for 43 and 453 minutes respectively (for a total of 635 mins, or 10:35 hrs; excluding colouring and post-processing). It should be noted that the medium-accuracy for reconstruction would require around 12 to 20 hours, but never left to finish due to the excessive amount of time that was required. Instead, a modified preview-reconstruction was used, with disabled “use of sparce point-cloud” and “max vertices count” of 60 million. That produced a clean-mesh with a 27.9 million triangles, in only 145 minutes.

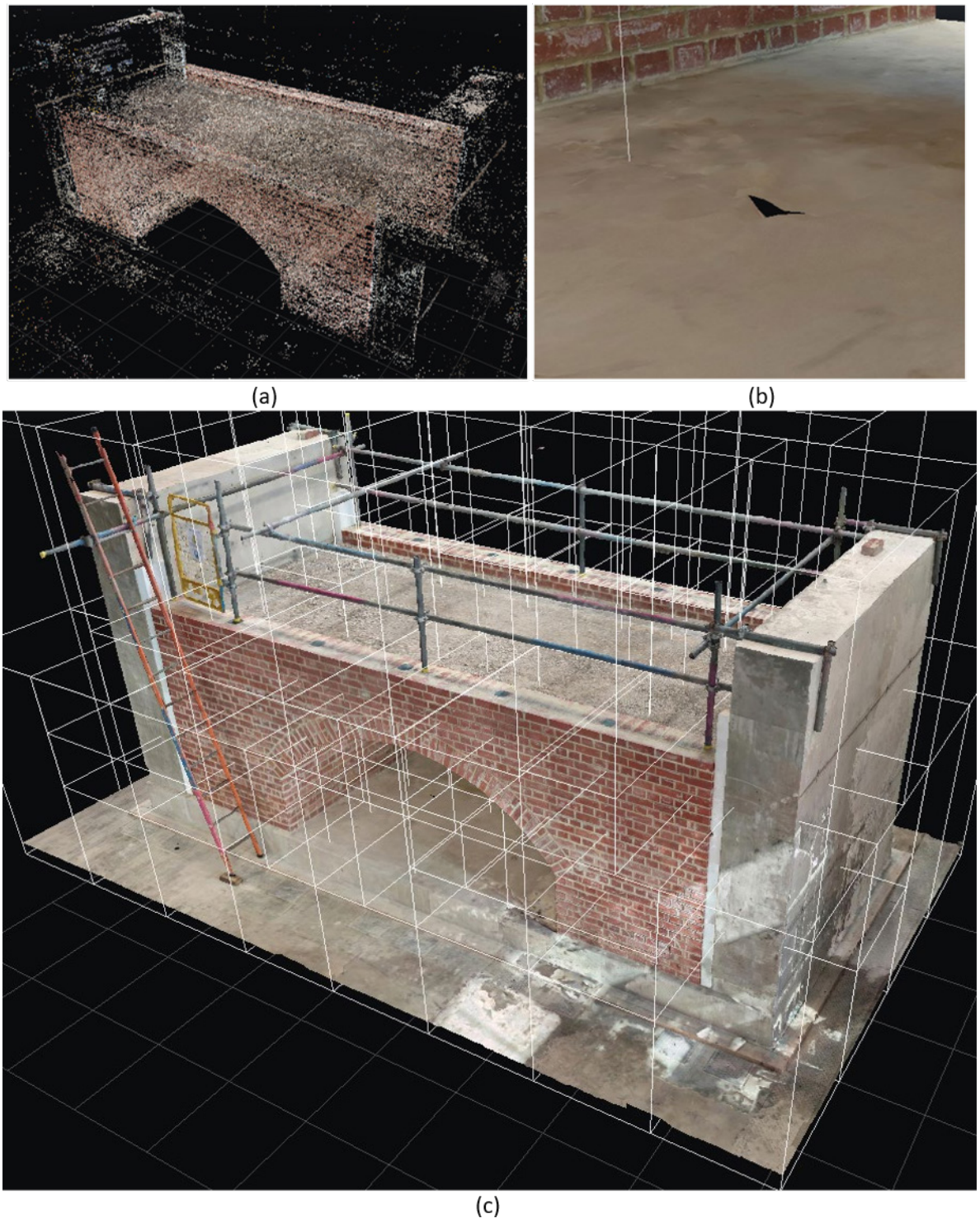


Fig. 6.2: Context Capture; a) Tie Points (alignment); b) Gap in mesh caused by the separation of tiles; c) Final textured-mesh.

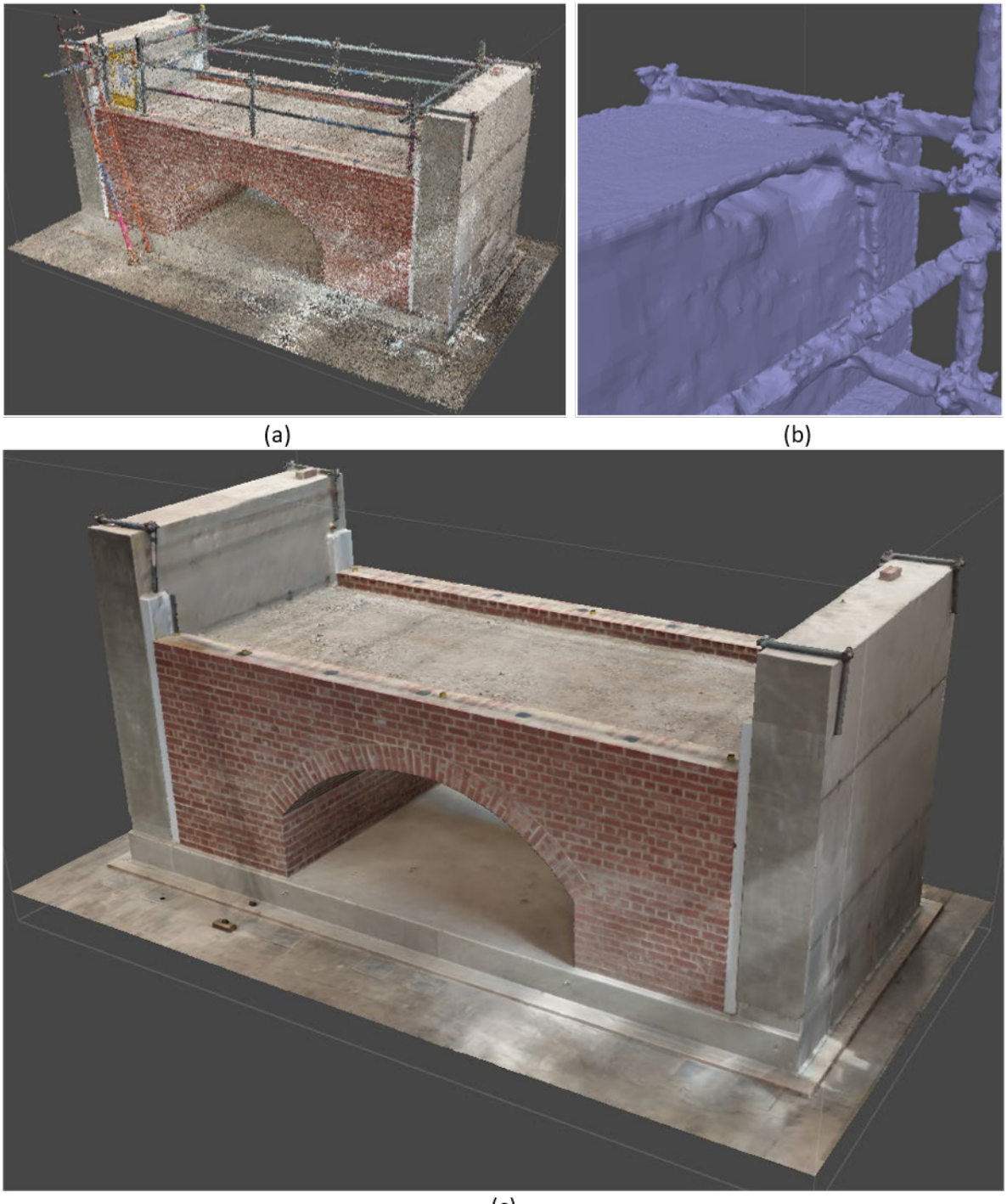


Fig. 6.3: Metashape; a) Tie-points (alignment); b) Mesh distortion due to small-degree of misalignment; c) Final textured-mesh (after smoothing).

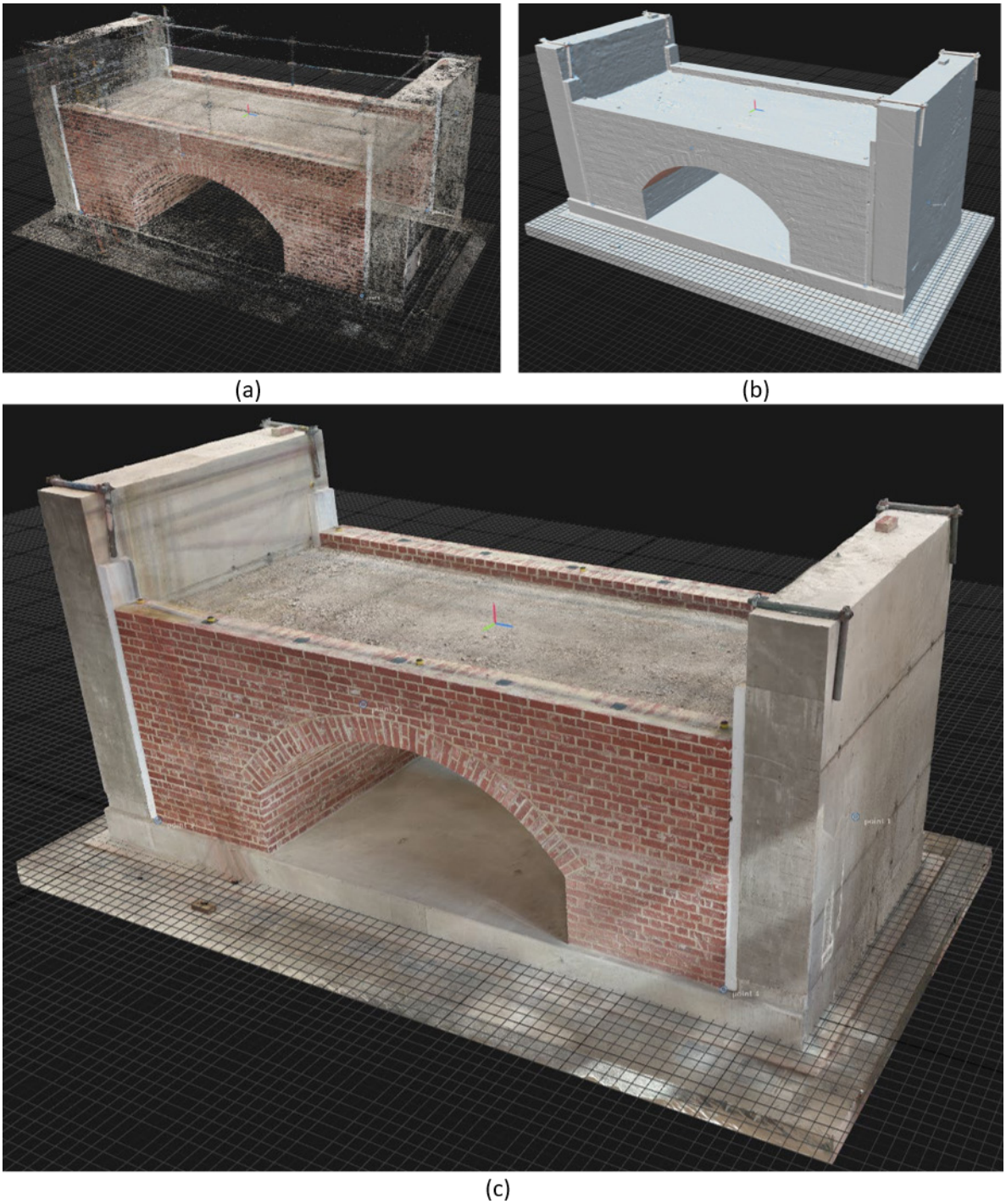


Fig. 6.4: Reality Capture; a) Tie-points (alignment); b) Solid mesh; c) Final textured-mesh (after smoothing).

6.3.2. Selection of Photogrammetry-Software

Regarding the visual-quality of the mesh, the best mesh was provided by “Reality Capture”, followed closely by “Context Capture”, and then “Metashape”. A small comparison between the software, for a complex location of the model, is provided in Fig. 6.5. From Fig. 6.5, it can be observed that “Context Capture” generated the ladder almost completely (excluding a small hole and a small area at the very top), “Reality Capture” generated most of the ladder without holes, while “Metashape” has large sections missing. Although, “Metashape” provides a better mesh in the door-area. Furthermore, it is observed that “Metashape” has the smoothest surface of the wall, followed by “Reality Capture”, with last the model produced by “Context Capture”. Additionally, “Context Capture” has small holes in the

mesh in certain areas, between tiles (Fig. 6.2: b), although that may be resolved with repetition of the procedure and adjustment of the model-configuration.

Regarding the textures, “Metashape” provided the best textures (visually), followed closely by “Reality Capture”. “Context Capture” had the lowest performance in that regard due to the division of the model in multiple tiles, which caused discrepancy between tiles. Regarding the complexity, “Context Capture” was the simplest to use followed by “Reality Capture”. Metashape required the highest manual-manipulation to provide similar results.

Finally, all 3 produced good results but only the “Metashape” and “Reality Capture” models are useable for this case (since “Context Capture” produced too many tiles to create a useable model). Between the 2, the simplicity and performance of “Reality Capture” is preferred. Thus, the “Reality Capture” model is used for the remaining of the study.

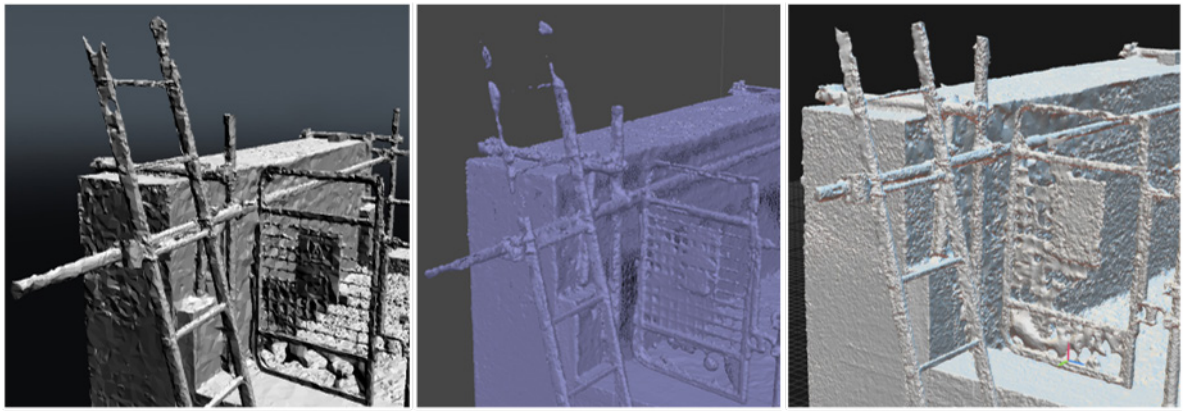


Fig. 6.5: Mesh comparison (before smoothing); a) Context-Capture; b) Metashape; c) Reality-Capture.

6.4. Image-Capture from the 3D Model (P2)

The objective of this section is to present the methodology for capturing images, surrounding the structure, directly from the 3D model. Those images will later be analysed using artificial-intelligence and image-processing. Multiple classes are derived from this procedure. Those are blocks, cracks, mortar, and unspecified/other elements. For that reason, 10 cameras are placed surrounding the structure to cover the whole surface. For most structures the 10 cameras are more than sufficient to cover the complete surface. However, since the structure is a complex one with distinctive features, 3 additional cameras were placed within the arch to classify that region as well. The 3 manual-cameras are the only part of the algorithm not fully-automated (although, limited to specific cases). The complete camera properties are provided below:

Table 6.1: Rendering-camera properties (the camera-order is important to filter the final-output of the classification).

Cam - [No]	Looking at - -	From Mid. -	Location (dx, dy, dz) [meters]	Rotation (rx, ry, rz) [degrees]	Capt. Size (x, y) [meters]	Far-Plane (z) [meters]
1	Front (South)	False	Auto	(90, 0, 0)	Auto	Auto
2	Back (North)	False	Auto	(90, 0, 180)	Auto	Auto
3	Left (West)	False	Auto	(90, 0, 270)	Auto	Auto
4	Right (East)	False	Auto	(90, 0, 90)	Auto	Auto
5	Top	False	Auto	(0, 0, 0)	Auto	Auto
6	Bottom	False	Auto	(180, 0, 0)	Auto	Auto
***7**	Inner #1 (left-up)	True	(-0.2, 0, -0.8)	(135, 0, 90)	3.0	1.5

***8**	Inner #2 (right-up)	True	(-0.2, 0, -0.8)	(135, 0, -90)	3.0	1.5
***9**	Inner #3 (bottom)	True	(0, 0, -0.5)	(0, 0, 0)	3.5	1.5
10	Diagonal #1 (SE)	False	Auto	(45, 0, 45)	Auto	Auto
11	Diagonal #2 (NE)	False	Auto	(45, 0, 135)	Auto	Auto
12	Diagonal #3 (NW)	False	Auto	(45, 0, 225)	Auto	Auto
13	Diagonal #4 (SW)	False	Auto	(45, 0, 315)	Auto	Auto

** Cameras and properties not suitable for every model.*

Certain toggles are available that simplify the process. For example, the toggle "From Middle" allows the location to be adjusted from the centroid of the model. The rotation is the camera-rotation to adjust where the camera is looking-at. The "Location" is the camera displacement from either the axis-origin or model-centroid. The "Capture Size" is the length/width of the image, converted to meters. Lastly, the "Far Plane" is the maximum rendering-distance of the camera. Almost every camera is automated and does not require further adjustments per model-case. The only exceptions are the inner-cameras (Table 6.1: Cameras 7-9), where their properties are given manually. For every case, the quality of the renders is adjustable and depends on the accuracy required (Eq. (57)). It should be noted that where the capture-size is given (i.e., cameras 7-9), the resolution/size is adjusted to ensure that the image-scale remains the same across all renders (similar to Eq. (59)).

$$ImgScale\ (default) = 0.002\ meters/pixels \tag{57}$$

6.4.1. Evaluation of the Camera Location

For the automatic-location, the camera-radius is evaluated based on the size of the structure (Eq.(58)). Where $[xmax,\ xmin,\ ymax,\ ymin,\ zmax,\ zmin]$ are the coordinates of the bounding-box of the structure. Furthermore, the resolution of the renders is also based on the size of the structure, to ensure that the whole structure will fit in the render under the given Image-scale.

$$Dist3D = \sqrt{((xmax - xmin)^2 + (ymax - ymin)^2 + (zmax - zmin)^2)} \tag{58}$$

$$Camera\ Radius:\ CamRadius = Dist3D$$

$$Test\ Image\text{-}Size:\ TempSize = Dist3D * 1.1 \tag{59}$$

$$Resolution:\ ResX = ResY = ceil(TempSize/ImgScale)$$

To find the exact-location, it is necessary to identify where the camera is looking at, given a rotation and depth (Eq. (60)). Where $XYZI$ and $XWZW$ are the image and world-coordinates respectively. K is the intrinsic-matrix (or projection-matrix), P is the extrinsic-matrix (or view-matrix). The camera is considered to be initially at the origin of the global-axis ($xyz = [dx, dy, dz] = [0,0,0]$). After which, the camera is moved so that it looks-at the centroid of the structure (Eq. (61)).

$$Image\ to\ World\ Coordinates: \tag{60}$$

$$XYZI = [XI, YI, ZI, 1] = dot(K * P^{-1} * XYZW) =>$$

$$XYZW = [XW, YW, ZW, 1] = dot(P * K^{-1} * XYZI)$$

$$Final\ Camera\ Location: \tag{61}$$

$$xyz' = -XYZW + Centroid$$

The rotation is given in Euler's angles to create the rotation matrix (Eq. (62)). The order of the rotation-matrix signifies the order of the rotation in the global-axis, where the last variable in the rotation matrix is applied first (i.e., $dot(RZ * RY * RZ) => rx \rightarrow ry \rightarrow rz$). Furthermore, the global-rotation is

equal to the inverse local rotation of the camera-axis (i.e., $RXYZ_{Global} = RZYX_{Local}$), where prior to any rotation, the global and local-axis are aligned. However, the local-axis rotation is more difficult to comprehend. Thus, only the global-axis is referenced in the equations. The rotation is combined with the translation-matrix to acquire the extrinsic-matrix. It should be noted that all matrices are appended a 4th column/row (including $XYZI$ and $XYZW$; Eq. (60)), to allow their inverse calculation (i.e., $[0,0,0,1]$; Eq. (63), (64)). This reforms all matrices to either 4x1 or 4x4 shape.

$$Extrinsic\ Matrix \sim View\ Matrix: P = dot(T * R) \tag{62}$$

$$Rotation\ Matrix\ (angles\ in\ rad): R = dot(RZ * RY * RX)$$

$$RX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(rx) & -\sin(rx) & 0 \\ 0 & \sin(\mathrm{rx}) & \cos(rx) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} <=> \quad RY = \begin{bmatrix} \cos(ry) & 0 & \sin(ry) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(ry) & 0 & \cos(\mathrm{ry}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{63}$$

$$RZ = \begin{bmatrix} \cos(rz) & -\sin(rz) & 0 & 0 \\ \sin(rz) & \cos(\mathrm{rz}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} <=> \quad T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Intrinsic\ Matrix \sim Orthorgraphic\ Projection\ Matrix: \tag{64}$$

$$K = \begin{bmatrix} 2/(r-l) & 0 & 0 & -(r+l)/(r-l) \\ 0 & 2/(\mathrm{t}-\mathrm{b}) & 0 & -(t+b)/(t-b) \\ 0 & 0 & -2/(\mathrm{f}-\mathrm{n}) & -(f+n)/(f-n) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The variables in the intrinsic-matrix define the view-area of the imaginary-camera (Eq. (64): left, right, bottom, top, near, far). Where the camera is assumed to be in the centre of the view-area (Eq. (65)). Normally the size of the view-box is equal to the scaled image-size. Additionally, the image-coordinates can consider any location in the image and are provided within the range of $[-1, +1]$. However, since only the centre is needed, the view-box can be simplified (Eq. (66)).

$$Projection\text{-}Matrix\ Properties \sim Camera\ View\text{-}Box: \tag{65}$$

$$l = -SizeX/2 \ <=> \ r = SizeX/2$$

$$b = SizeY/2 \ <=> \ t = SizeY/2$$

$$n = -SizeZ/2 \ <=> \ f = SizeZ/2$$

$$* Simplified\ Properties\ (for\ center\ only): \tag{66}$$

$$dx = dy = dz = 0$$

$$SizeX = SizeY = 2;\ SizeZ = 2 * CamRadius =>$$

$$XI = YI = 0;\ ZI = 1$$

*For the not-simplified version of the image-properties see Eq. (73) and Eq. (74)

6.4.2. Dividing the Main Render (Sub-renders)

One issue that could potentially arise, from large structures, is that the render-resolution may exceed the maximum-allowed and cause an error (Eq. (67)). Thus, if the initial-render exceeds the maximum resolution, the main-window is split into multiple-parts (Fig. 6.6, which ensures that each division is equal or less than the maximum-resolution. In this case, it is best to include an overlap to the new sub-render windows (Eq. (68); Fig. 6.6). That will increase the accuracy of the object-detection since

the divided-sections will retain some continuity. However, the overlap-area will not be considered during the point-extraction part of the workflow (section 6.7).

$$Max\ Resolution\ (default):\ MaxRes = 8{,}192\ px \tag{67}$$

$$Number\ of\ windows: \tag{68}$$
$$NX = NY = ceil(ResX/(MaxRes - 2 * Overlap))$$
$$Overlap\ (default) = ceil(MaxRes/35)$$

Moreover, the new camera-coordinates lie within the image plane. The new centres are found by repeating Eq. (60), with different image-size ($SizeX1$, $SizeY1$, $SizeZ1$; Eq. (69)) and image-coordinates ($XYZI1$; Eq. (70)). The camera rotation remains the same (rx, ry, rz; Eq. (63)). However, the main-camera position is assigned as the initial-displacement (xyz'; Eq. (70)) in the translation-matrix (T; Eq. (63)). The depth is irrelevant in this case (since $ZI1 = 0$), which is why it is given as any number ($SizeZ1$; Eq. (69)). Finally, each camera-centre can be calculated using the new image-size and coordinates (Eq. (71)).

$$New\ main\text{-}render\ size: \tag{69}$$
$$ResY1 = ResX1 = ceil(ResX/nx) * nx$$
$$SizeZ1 = SizeY1 = SizeX1 = ResX1 * ImgScale$$

$$Centers\ of\ subrenders\ (if\ the\ resolution\ exceeds\ the\ maximum): \tag{70}$$
$$XI1_i = ((nx_i + 0.5)/NX) * 2 - 1$$
$$YI1_j = 1 - ((ny_i + 0.5)/NY) * 2$$
$$ZI1_{(i,j)} = 0 \ \ <=> \ \ [\mathrm{dx1}, \mathrm{dy1}, \mathrm{dz1}] = xyz'$$
$$ny = nx = [0, \dots, NX - 1];\ Where\ [ny, nx] \in R$$

$$New\ Camera\ Locations\ (sub\text{-}renders): \tag{71}$$
$$xyz1'_{(i,j)} = XYZW1_{(i,j)} = dot(P1 * K1^{-1} * XYZI1_{(i,j)})$$

The previously calculated resolution/size is for the main-render. The sub-renders have a different size, based on the resolution and divisions of the main render (Eq. (72)). The following are the values provided to the off-screen-renderer and are not used in any other calculation.

$$New\ sub\text{-}render\ resolution/size: \tag{72}$$
$$ResY2 = ResX2 = ResX1/NX + Overlap * 2$$
$$SizeY2 = SizeX2 = ResX2 * ImgScale$$

Fig. 6.6: Sub-renders of camera #13, with overlap ($ImgScale = 0.001$).

6.4.3. Camera Renders

The cameras are using an orthographic projection (Fig. 6.7: a; Fig. 6.8), to control the scale of the image (meters/pixel). Otherwise, some elements in the image would be extremely small and avoid detection (see section 6.5). Additionally, the extraction of the 3D points, from the image, would have irregular intervals (see section 6.7). However, there are issues with the orthographic-render when the camera-limits are in contact with the object (Fig. 6.7: a). In that case, part of the internal/far regions of the model are also rendered. Those regions also provide depth, and thus will not be filtered out from the object-detection and may provide inaccurate classifications. Rendering of the internal-side of the model can be avoided by enabling "backface-culling", which improves the result but still renders the far-regions (Fig. 6.7: a). However, that would not be an issue if a perspective projection was used for the inner-renders (Fig. 6.7: b). Although, that would also mean that the image-scale would be unknown, alongside the aforementioned issues of a perspective-view.

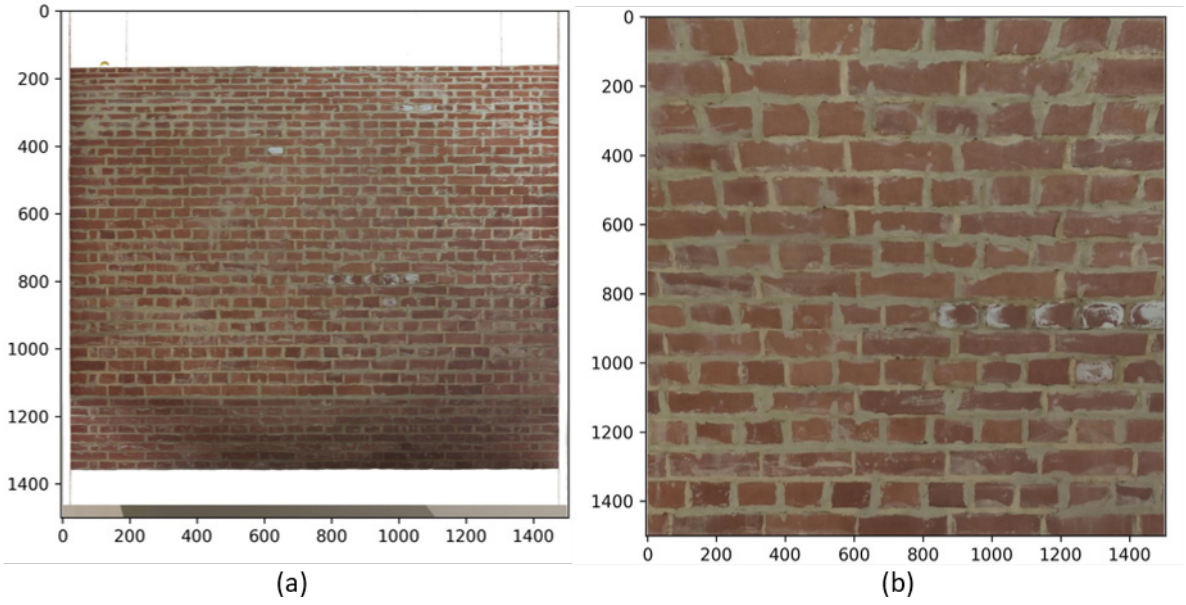


Fig. 6.7: Inner-Camera #7 (Axis in pixels); a) Orthographic render; b) Perspective render.

Additionally, the order of the cameras is important for a specific filtering method of the 3D points acquired, after the object-detection (see section 6.8), where the higher numbered-cameras have lower-priority (i.e., diagonal renders; Fig. 6.8: a2). Evidently, the front, back, left, right, top, and bottom cameras are higher in the list since they provide vertical renders of the element examined and cover the most important surfaces (Table 6.1: Cameras 1-6; Fig. 6.8: a1).

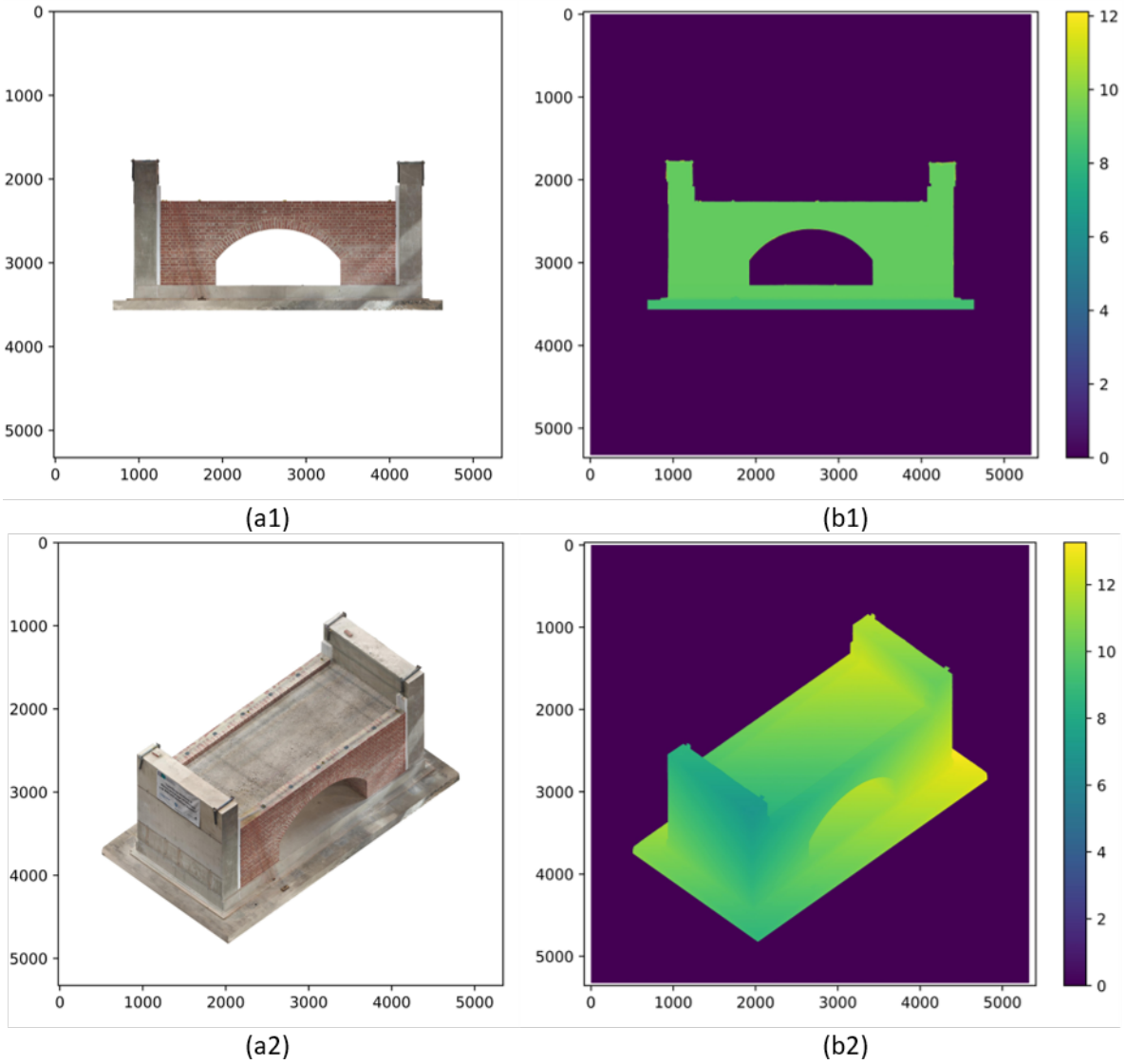


Fig. 6.8: Camera renders (Axis in pixels; Colour-bar in meters); a1) Render of camera #1; b1) Depth-map of camera #1; a2) Render of camera #13; b2) Depth-Map of camera #13.

The mesh is imported using the "Trimesh" package in python (i.e., "$trimesh.load()$"). The rendering-process is followed using the "pyrender" package (i.e., "$r = pyrender.OffscreenRenderer()$"), with a camera-pose equal to the view-matrix (P; Eq. (62)). However, "pyrender" provides incorrect depth-map (only for the orthographic-camera). That was corrected by making some minor-changes to the package and by importing the modified-package under a new name. The required-corrections are provided in (Nicastro, 2019). Any package that supports off-screen-renderer, orthographic-view, and depth-map can be used to replicate the process. However, the "trimesh-pyrender" pair was used because "trimesh" allows to import a mesh with multiple texture-files correctly. Where other packages require to re-assign the textures to the individual mesh-parts (when a single model-file has multiple texture-files).

6.5. Object Detection (P3, P5)

Each image-render is analysed to identify the location of blocks, cracks, and background. Those three classifications are enough to derive the classification of mortar and other-elements, later in the process (see Section 6.6.2). Thus, a complete segmentation will include block, cracks, mortar, other-elements, and background.

6.5.1. Automatic Annotation (P3, P5)

The background is identified automatically from the depth-map (Fig. 6.9). More specifically, where the depth is equal to zero (Fig. 6.8: b1), the background is equal to 255. This produces a binary-image where it shows the part of the render with infinite-depth. The background-mask is used to filter the remaining classifications.

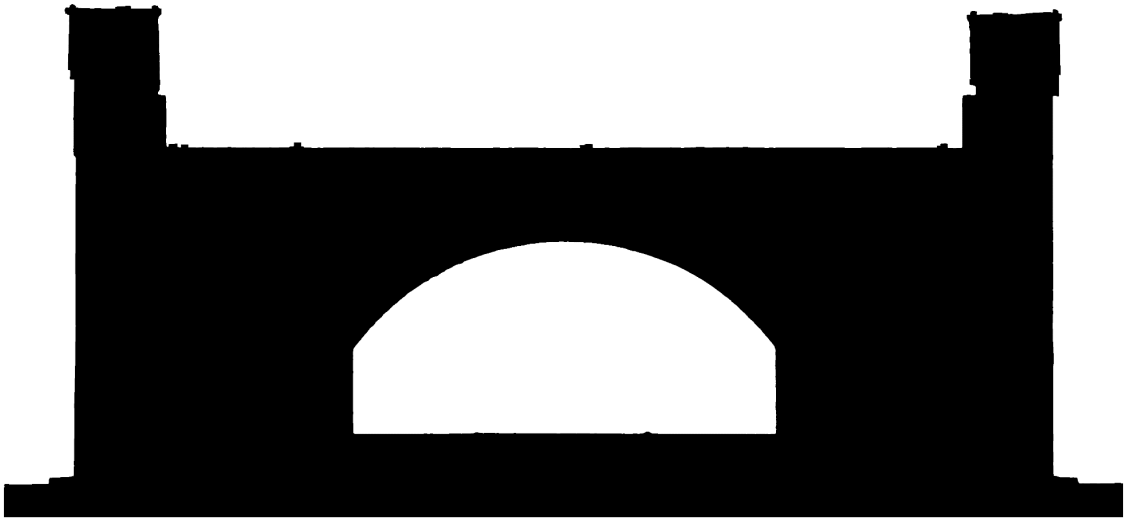


Fig. 6.9: Background detection (from depth-map).

Furthermore, the blocks and cracks are identified using semantic-segmentation through the use of CNN (Fig. 6.10). Those models have been trained to a large number of manually annotated images of blocks and cracks, with 95.66% and 79.6% validation-F1-scrore respectively. The CNN-models used in this study are described in (Dais *et al.*, 2021; Loverdos and Sarhosis, 2022a). However, since the structure has not been damaged, only the block-detection model is demonstrated.

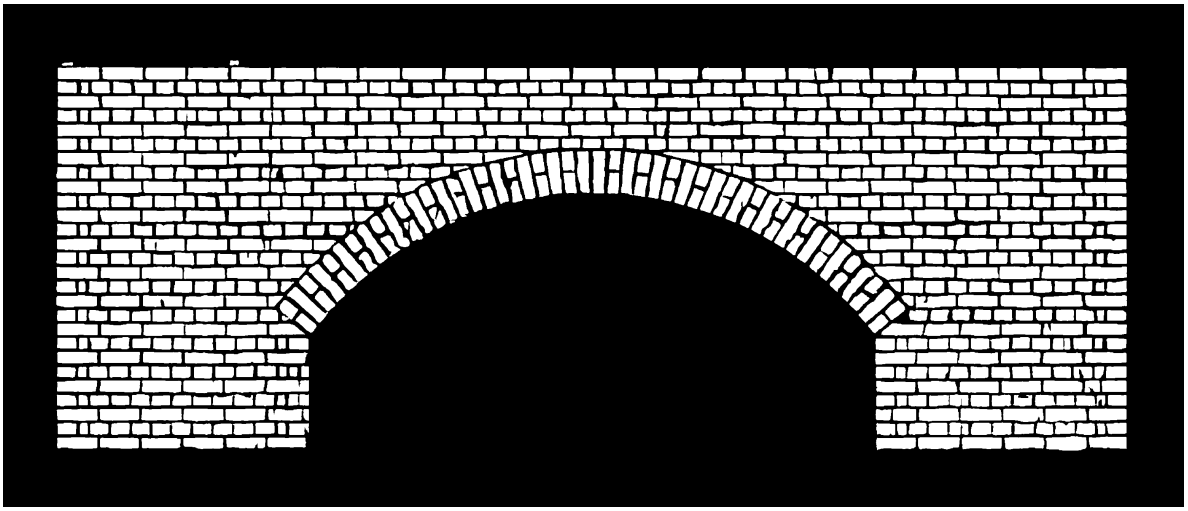


Fig. 6.10: Block-detection (using CNN model).

6.5.2. Manual Annotation (Other)

Manual-annotation has the potential to replace the automatic detection of background, blocks, and cracks. The aim is to allow the commercial use of the software for 3D-annotation, especially when the provided accuracy is not acceptable (i.e., for an inspection report), or to annotate only selected points-of-interest. Although applied only to cracks, the proposed method can extend to any classification i.e.

missing bricks, vegetation etc. The crack pattern presented here is the one observed by testing the masonry arch bridge under static conditions in the laboratory (Fig. 6.11).

Fig. 6.11: Recorded crack pattern after multiple tests on the experimental arch-bridge.

Manual-annotation is applied using a companion-program that compares up to 4-images and generates a new-mask that marks the modified-locations. The main-components are the original-render and modified-render (Fig. 6.12: a). Optionally, the background-mask and an extra-mask can be used to append to the results. The extra-mask can be an already existing-mask (i.e., result from crack-detection), used to add existing-annotations. The complete sequence is presented below:

1. Create a zero-array with size equal to the original-image (i.e., $size = [xi, yi, 1]$).
2. Where: $Modified \neq Original,\ NewMask = 255.$
3. [Optional] Where: $ExtraMask > 0,\ NewMask = 255.$
4. [Optional] Where: $Background > 0,\ NewMask = 0.$

* The comparison must consider only the first 3 layers of each image or else the New-Mask may be marked incorrectly if one image has a different number of layers. Furthermore, both images must use a lossless format (i.e., png). Otherwise, the 2-images will be different due to compression-artifacts.

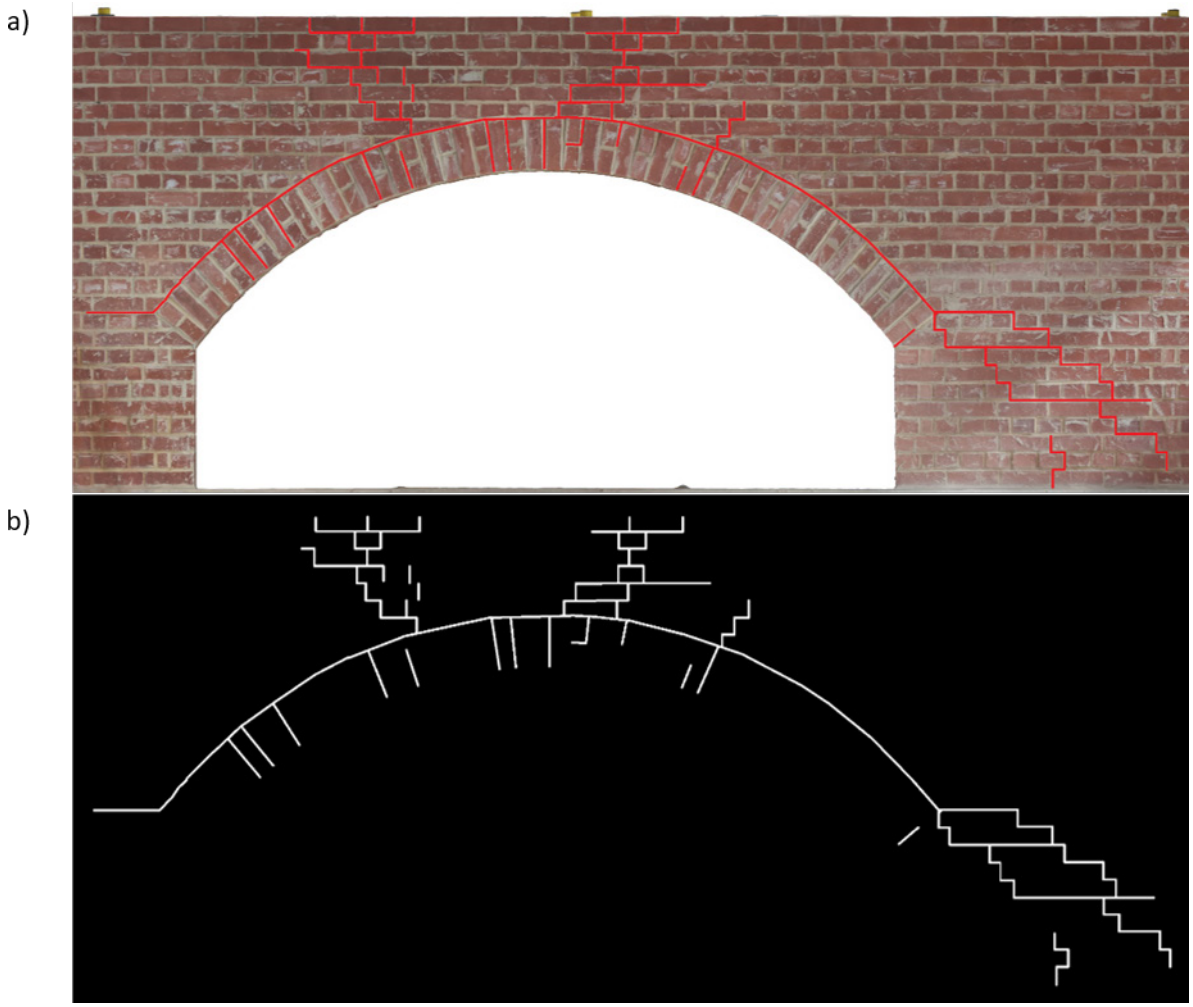


Fig. 6.12: Generation of manual-annotated mask by image-comparison (the crack-formation is recorded manually); a) Modified-render (camera #1); Generated-mask of cracks.

The adjusted-masks are added to an "Override" folder, so that they don't replace/remove existing-detections. Any image in the "Override" folder has priority over the typically-obtained masks. There is an "Override" folder assigned to every main-classification (background, blocks, cracks). Mortar and other-elements are excluded from this process since they are produced by combining the main-classifications (Section 6.6.2). By overriding the background, the operator can exclude a specific area from classification; thus, create controlled-gaps to the final point-cloud (i.e., remove vegetation; section 6.7). Optionally, the program may skip/ignore automatic-classifications and use only those present in the "Override" folder for specified classifications (i.e., cracks), allowing the use of the software as a standalone annotation tool.

6.6. Post-Processing of Masks (P4, P6, P7)

6.6.1. Image-Processing (P4, P6)

Image-processing is used to improve the output of the object-detection, by removing small-objects that are considered a leftover-artifact of the object-detection. This procedure is applied only on blocks and cracks, mostly because the remaining classifications are derived from blocks/cracks and thus if those are optimal their products will also be. The background-mask is excluded since their output is absolute and always show where a pixel targets empty-space, accurately.

The application of the image-processing is the following:

1. [Required] Binarize the mask using thresholding ($t = 0.5$).
2. [Optional] Removal of small objects below a given threshold (enabled).
   2.1. Remove small objects from the foreground ($tb1 = 0.0004;\ tc1 = 0.0001$).
   2.2. Remove small objects from the background ($tb2 = 0.0004;\ tc2 = 0$).
3. [Optional] Apply dilation/erosion/dilation in this sequence (disabled in this case).
4. [Optional] Repeat removal of small objects. Only applied if dilation/erosion/dilation has been used (disabled in this case).

The identification of the size of each object (either foreground or background) is identified by generating the watershed-markers for each object and then counting the pixels of each label. The removal of small-objects is applied on the background-objects by inverting the mask and following the same methodology as for the foreground-objects. Then the image is inverted back to restore its original polarity. The blocks and cracks are filtered with a threshold of size $tb = 0.0004m^2$ and $tc1 = 0.0001m^2$, respectively. However, the object-removal of background-cracks is not used ($tc2 = 0$), to preserve small gaps between crack-formations. On the contrary, the background-blocks are filtered in because watershed-segmentation is not applied on the blocks. If watershed-segmentation was applied on the blocks, to improve the output, the small-objects of background could be used to repair open-blocks. Additionally, the Lastly, the values provided are only suggestive and can be adjusted to suit different needs. Further information about the method can be found in (Loverdos and Sarhosis, 2023b).

6.6.2. Segmentation of All Classifications (P7)

The segmentation is the part of the algorithm that combines all the classifications into a single array. This part also introduces the mortar/other classifications. The methodology follows a prespecified priority to ensure the merging of all classifications. This process is followed for every render (and sub-render) individually. The complete procedure is provided below:

1. [Masonry] Create the masonry-mask by applying image-closing on the blocks (dilation/erosion).
   1.1. [Corrections] Where: $Blocks > 0, Masonry = 255;$
   1.2. [Corrections] Where: $Background > 0, Masonry = 0.$
2. [Segmentation] Create a zeros-array equal in size to the original-render (i.e., $size = [xi, yi, 1]$), as the segmentation-image.
   2.1. [Mortar class] Where: $Segmentation = 0, Segmentation = 3.$
   2.2. [Structural class] Where: $Masonry = 0, Segmentation = 1.$
   2.3. [Blocks class] Where: $Blocks > 0, Segmentation = 2.$
   2.4. [Cracks class] Where: $Cracks > 0, Segmentation = 4.$
   2.5. [Background class] Where: $Background = 0, Segmentation = 0.$

The overall-performance of the block-detection/segmentation process provides better results with vertical-cameras (Fig. 6.13), as opposed to diagonal-cameras (Fig. 6.14). However, both will be used for the final point-cloud. Parts of the diagonal-cameras will cover small-gaps in the classified point-cloud (section 6.7), while their inaccuracies will be filtered out (section 6.8).

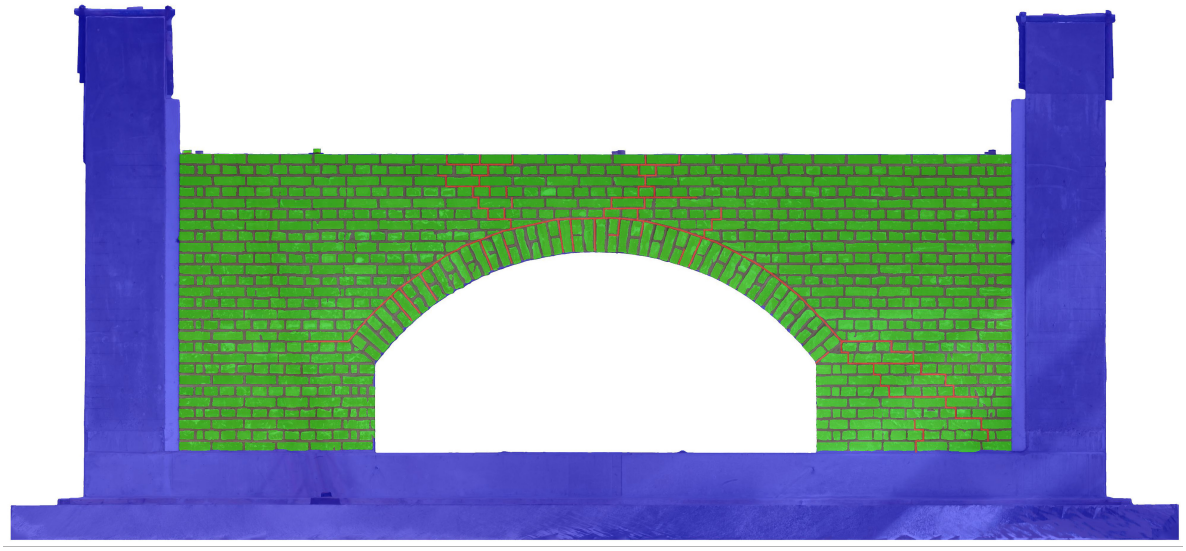


Fig. 6.13: Segmentation overlay in 2D (Camera #1); Green: Blocks; Grey: Mortar; Red: Cracks; Blue: Others.

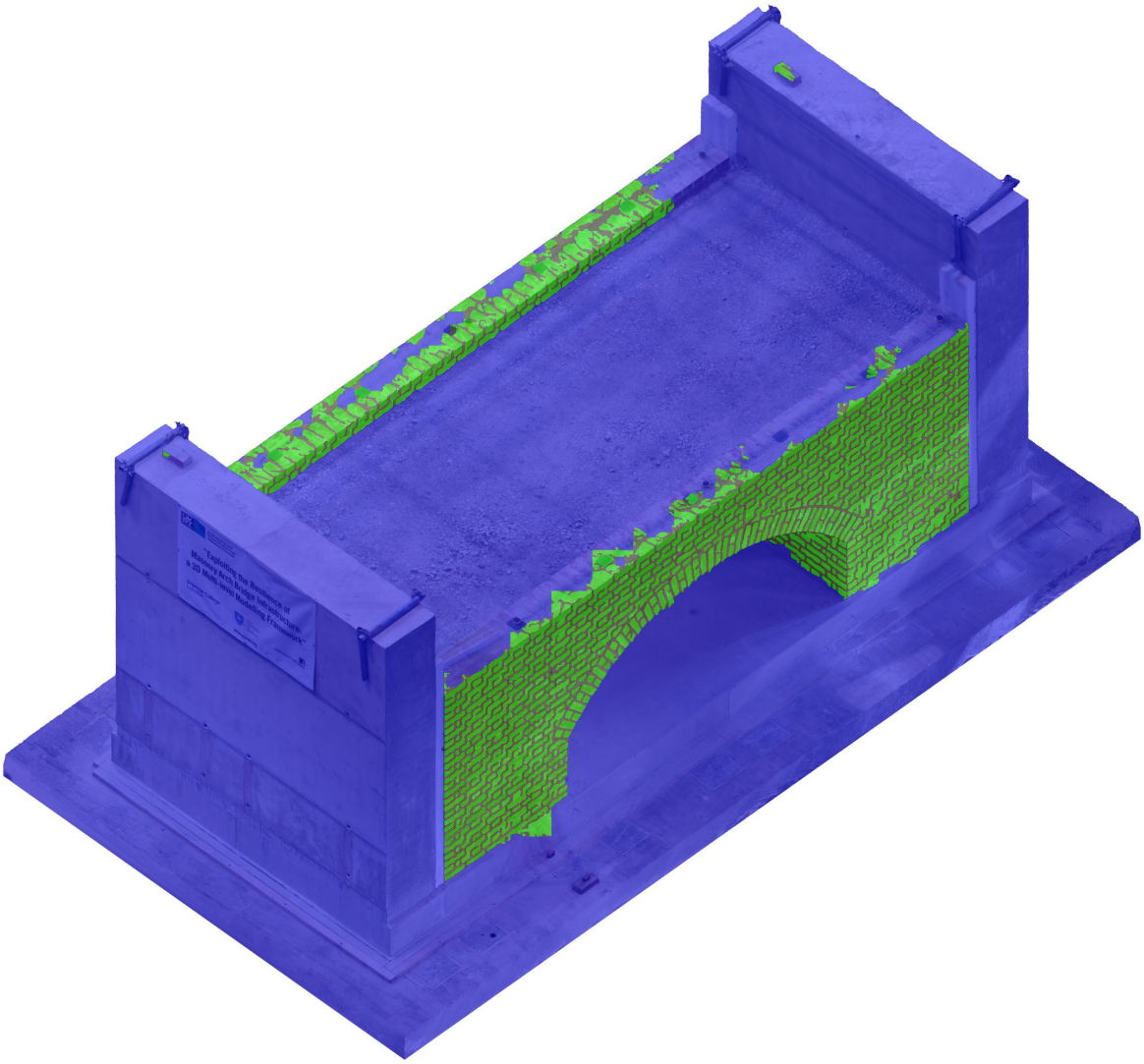


Fig. 6.14: Segmentation overlay in 2D (Camera #13); Green: Blocks; Grey: Mortar; Blue: Others.

6.7. Point Extraction from Images (P8)

The purpose of this section is to present the methodology used to extract the classified-pixels of the segmentation (Section 6.6.2) into 3D coordinates. The method is similar to the one followed to identify the world-coordinates of the camera-centre (Eq. (60)). Compared to the previous use-case, there are two differences. Firstly, the whole image-array is analysed. Thus, the image-size used must be representable of the real size (i.e., no simplification is applied). Secondly, the world-coordinates, acquired from the formula, are not modified. Where previously, the location was adjusted to ensure that the camera is looking at the model-centroid (Eq. (61)).

The renders have the same XY-size to simplify the process (i.e., $ResX = ResY$; Eq. (73)). The size of the image refers to the scaled-size, converted to meters (i.e., SizeX, SizeY, SizeZ). The size is acquired from the resolution. Since the initial image-size was modified to ensure that the resolution is an integer, while retaining constant image-scale. Each pixel has its own image-coordinates, where the image-coordinates are the pixel-coordinates converted to $[-1, +1]$ range (Eq. (74)). The image-axis starts from the top-left corner of the image. Thus, the sign of the Y-Y axis is reversed to account for that. Additionally, the image-pixels are given using the python convention, where the first row/column is the number 0 (i.e., $XP_0 = 0, YP_0 = 0$). Each coordinate acquired is located in the middle of the pixel. Thus, the pixel value is adjusted to add half-a-pixel to the image-coordinates (i.e., $XP + 0.5$; Eq. (74)). The image-coordinates are given in an array that contains all the coordinates of the individual-image, which improves the calculation speed significantly (i.e., $Shape_{XYZI} = 4 * n;\ n = ResX * ResY$). Finally, the global-coordinates are determined using the same formula as previously (Eq. (75)), where the camera intrinsic/extrinsic properties (Eq. (62), (64)) are calculated using the new size/displacement values (Eq. (73)), but with the same camera-rotation used for the render (RX, RY, RZ; Eq. (63)).

$$\begin{gathered} Scaled\ Image\text{-}Size: \\ SizeX = SizeY = ResX * ImgScale => \\ Scaled\ Depth:\ SizeZ = \max(Depth) * 2 \\ Camera\ Displacement:\ [dx, dy, dz, 1] = xyz' \end{gathered} \tag{73}$$

$$\begin{gathered} Image\ Coordinates\ (XYZI): \\ [XI, YI, ZI] \in [-1, \dots, +1] => \\ XI = 2 * (XP + 0.5)/ResX - 1 \\ YI = 1 - 2 * (YP + 0.5)/ResY \\ ZI = 2 * Depth_{(i,j)}/SizeZ \end{gathered} \tag{74}$$

$$XYZW_{(i,j)} = dot\left(P * K^{-1} * XYZI_{(i,j)}\right) \tag{75}$$

The final-output of the point-extraction is a collection of classified 3D points, that can be used as an overlay on top of the original model (Fig. 6.15). Using one of the many 3D modelling-tools (i.e., AutoCAD, Rhino), the generated point-cloud can be used for convenient visualisation of the structure under investigation. In this way, it is possible to assign different classifications to individual-layers and toggle any group to enhance visibility (i.e., by turning-off all layers but cracks), or even change the colour of the layer. Regarding the quality of the output, the unfiltered point-cloud demonstrates overlapping classifications (Fig. 6.15: b). This is caused by the classification of common areas between multiple renders. If the object-detection was ideal, the overlapping-classification would be minimal. However, the angular-renders have lower accuracy (compared to planar), causing the aforementioned issue. This is resolved almost fully in the following section (section 6.8), by filtering the points using simple-methods.

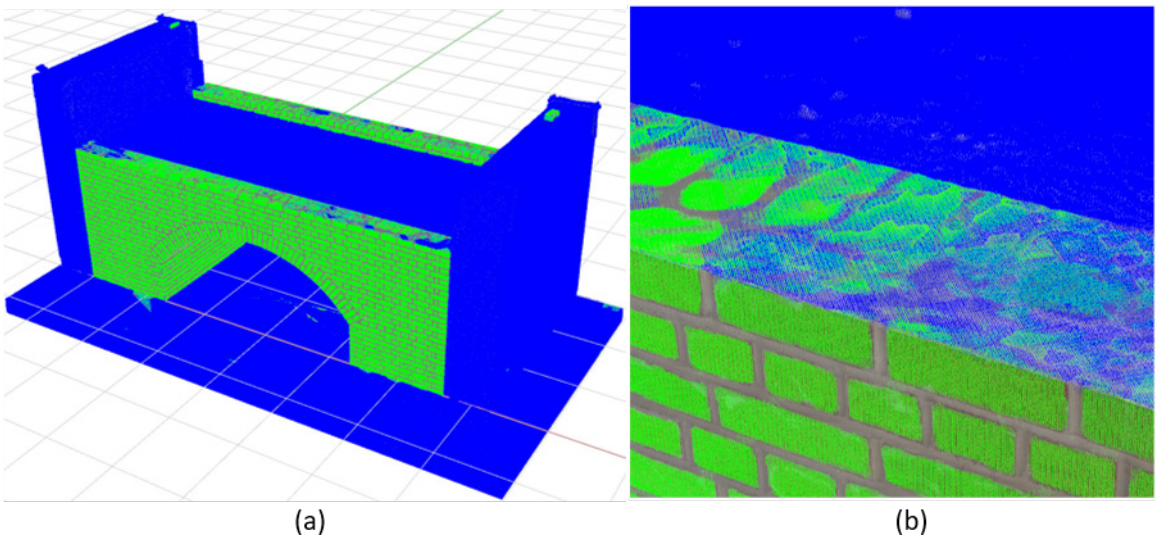


Fig. 6.15: Visualisation of extracted-points in Rhino7 (unfiltered; excluding damage); a) Overall view; b) Zoomed view.

6.8. Point Filtering (P9)

There are two steps in the filtering process. The first step is based on the camera-priority while the second one is based on classification-priority. The camera-priority filtering adds the points of the individual renders to a combined-pool but ignores points that are in close-proximity to an existing-point already within the pool. Thus, each collection of extracted-points, from individual renders, is added to the pool one by one. The process starts with the first-camera and continues with the same-order as the camera-order (section 6.4.1). There are 2 threshold values to evaluate the distance between existing and test-points. The first threshold-value compares the distance between the closest existing-point of a different-label to the test-point. The second threshold-value compares the closest existing-point of the same-label as the test-point. Each point in the collection is tested against every point in the pool.

$$Filtering\ Method\ \#1: Camera\text{-}priority\ (default\ values) \tag{76}$$

$$Threshold\ \#1\ (Different\ Label): Thld1 = 0.99 * ImgScale$$

$$Threshold\ \#2\ (Same\ Label): Thld2 = 0.49 * ImgScale$$

After the points have been merged there may still be overlapping-classifications (depending on the threshold values used). The second filtering method is filtering the merged point-cloud, removing points in close-proximity to any point of a higher-priority label (Eq. (77)). The classification-priority is based on the expected accuracy of each classification. The output of the manual-annotation is considered the most accurate (cracks), followed by the output of the CNN model (blocks). The remaining classifications are acquired using simple array-operations (mortar, structural), and thus, are last in the priority list.

$$Filtering\ Method\ \#2: Label\text{-}priority\ (default\ values) \tag{77}$$

$$Threshold\ \#3\ (Higher\text{-}Priority\ Label): Thld3 = 0.99 * ImgScale$$

$$Priority\ Order: [Cracks \rightarrow Blocks \rightarrow Mortar \rightarrow Others]$$

Different threshold values were tested for all 3 thresholds. However, the ones suggested provided the most accurate results. For example, if only the label-priority is used (i.e., by using $Thld1 = 0$), the less accurate renders will force overestimation of the higher-priority labels (Fig. 6.16: a). Furthermore, those areas will contain both classifications, which is detrimental for visualisation. Even if the camera-priority filtering-method is not applied (Eq. (76)), the same-label filtering should always be used to control the point-density of areas with the equal classification in multiple renders. When a lower

same-label threshold is used (i.e., $Thld2 < Thld1$; Eq. (76)), areas with equal label in multiple segmentations will have higher density. Thus, providing a visual difference of areas with higher classification-certainty. Higher density of higher certainty areas could be advantageous for more sophisticated filtering-methods. In general, if both filtering-methods are used, then the Label-priority filtering will not provide any substantial difference to the final-result (Eq. (77)). Except if the second filtering-method used a threshold smaller than any of the thresholds of the previous filtering-method (i.e., $Thld3 < Thld1$ or $Thld3 < Thld2$). Otherwise, all points will already satisfy the thresholds of the camera-priority filtering-method (Eq. (76)).

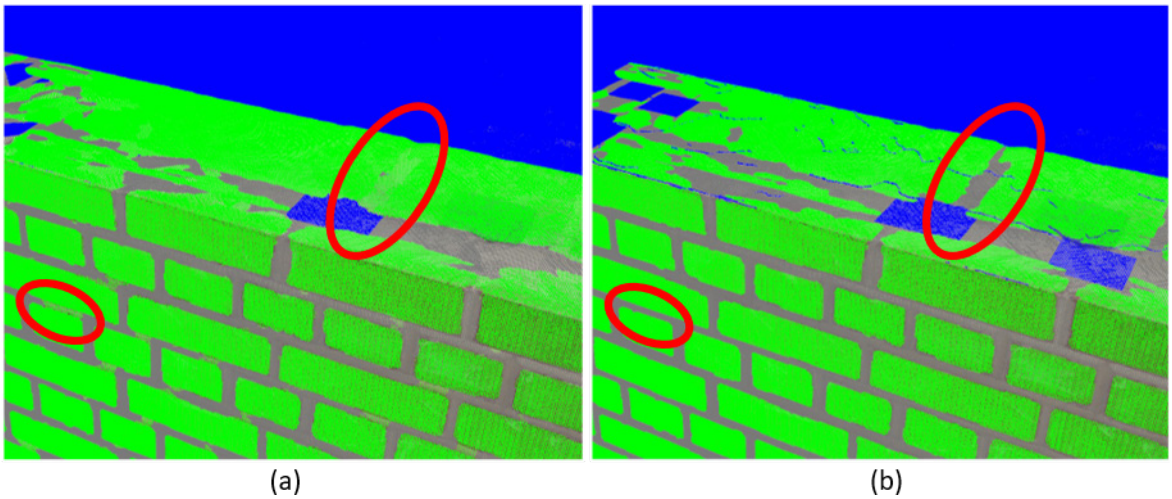


Fig. 6.16: Comparison of filtering methods; a) Label-priority only ($Thld = [0,0.49,0.99]$); b) Camera & label-priority ($Thld = [0.99,0.49,0.99]$).

The main disadvantage of this method is that locations near completely vertical-corners of the object (as seen by the render), that are classified incorrectly, may drag incorrectly classified-points to deeper areas (blue lines in Fig. 6.16: b). For example, the blue-lines formed (in Fig. 6.16: b) are caused by the segmentation of the camera #1 (Fig. 6.13). Where the top-side of the object is classified as structural instead of blocks. Those, incorrectly classified-pixels, generated incorrectly-classified points to the distance acquired by the depth-map. However, it is expected that a more advance filtering-method can resolve the aforementioned issue with ease, such as by adjusting the label of points that describe very thin objects (since the incorrect regions are thin/small).

Nonetheless, the final-result, using both filtering-methods (i.e., $Thld = [0.99,0.49,0.99]$), is accurate (Fig. 6.17 and Fig. 6.18). It can be observed that even the inner-region of the arch was classified appropriately (Fig. 6.18: Left). The same is valid for the overall surface of the structure. Where it is observed that the block-classification follows the shape of the brick accurately (Fig. 6.18: Right). Regarding the presentation of the results, the original-model can be included in the final-output to use the classified point-cloud as a type of 3d overlay (Fig. 6.17). Furthermore, any classified-group can be disabled (Fig. 6.17) or change its colour (Fig. 6.18), to enhance visualisation.

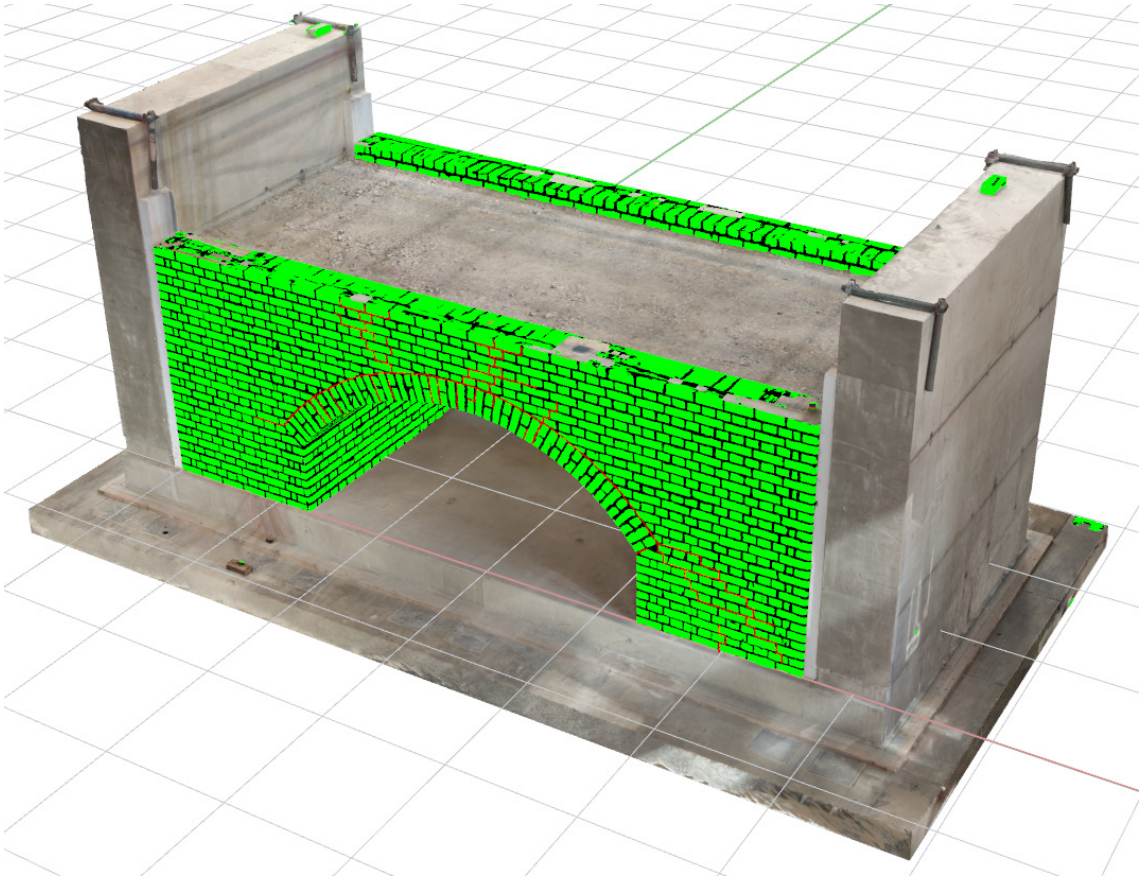


Fig. 6.17: Final classified point-cloud, excluding the structural-classification.

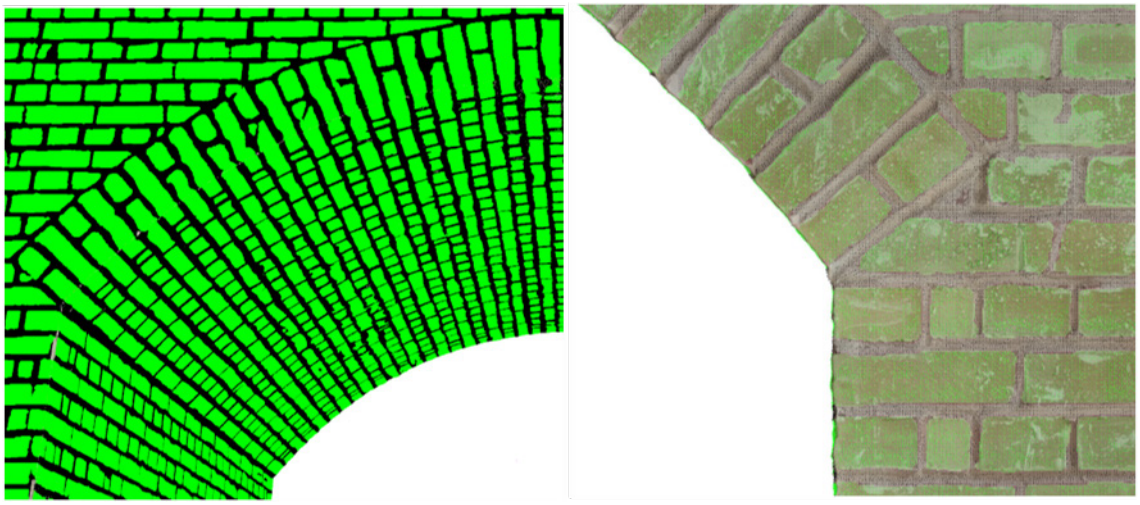


Fig. 6.18: Detailing of classified point-cloud, excluding structural and cracks classifications.

Lastly, the distance between points was tested using the "KDTree().query()" of the "SciPy" package, which provides the closest-point between 2 groups of points (testing and verified-points). The specific package was selected mainly because it allowed to ignore points further than a pre-specified distance (i.e., $MaxDist = Thld * 1.1$). Thus, not even providing an output for a large number of points. Furthermore, it allowed to limit the output to a single result per testing-point. Both features reduced the ram-requirements of the process and improved the computational-speed substantially, compared to other solutions (i.e., to minutes instead of hours). Especially important since every rendered-camera was increasing the computational-time logarithmically.

6.9. Damage Evaluation (P10)

The final-step of the proposed workflow creates the mesh of the classified point-cloud of cracks (Fig. 6.19) and measures their geometric properties (Table 6.2; Table 6.3). The main-purpose, of the damage evaluation, is to assist with the automation of visual inspection in 3d space. It should be noted that the crack-classification shown here is provided manually (section 6.5.2). The CNN model was not used primarily because the structure did not exhibit noticeable damage but also to illustrate the simplicity of manual annotation for detecting cracks in 3D models. However, a crack-detection model is easily implementable (similar to the block-detection model).

Initially, all points of cracks are converted to mesh using the ball-pivoting algorithm of "Open3D" ("o3d.geometry.TriangleMesh.create_from_point_cloud_ball_pivoting()"; Eq. (78)). Then the trimesh repair command is used to close holes of a single triangle element ("trimesh.repair.fill_holes()"), in case the mesh has holes from the ball-pivoting algorithm. This is combined with the "pymeshlab" command to close holes of larger openings ("ms.meshing_close_holes(maxholesize=10)"). The face-normal and vertex-normals are also calculated using the "pymeshlab" (i.e., "ms.current_mesh().face_normal_matrix()").

$$Ball\text{-}Pivoting\ properties: \tag{78}$$

$$Radii = [1 * Scale, 2 * Scale, 3 * Scale]$$

$$Scale = 1.25 * ImageScale$$

Fig. 6.19: Classified point-cloud and mesh of cracks.

Due to the radii-size of the ball-pivoting algorithm (Eq. (78)), which depends on the image-scale, each crack remains isolated. Thus, the bounding-box and area of each crack is easily identifiable (Table 6.2; Table 6.3). For the length, width, and inner-centroid, each crack-mesh is skeletonize using the "skeletor" package (). In more detail, the skeleton is used to acquire the vertices of the polyline of the skeleton and evaluate the length of the crack and thus, the average width (i.e., W=A/L; Table 6.3). The centroid is acquired from the skeleton-vertex closest to the centre of the bounding-box of the crack-mesh. The area of each crack is compared to the area of all cracks and of that of the model, to acquire the coverage (Table 6.3). Thus, providing an important metric to evaluate the current condition of the structure.

Table 6.2: Initial geometric-properties of cracks.

Crack	Centroid			Boundaries					
No	xc	yc	zc	xmin	xmax	ymin	ymax	zmin	zmax
-	m	m	m	m	m	m	m	m	m
1	2.65	1.328	1.251	0.854	5.464	-0.005	2.916	0.066	2
2	3.38	1.567	1.258	3.18	3.566	0	2.903	1.187	1.364
3	4.302	1.439	0.595	4.293	4.386	-0.002	2.899	0.584	0.67
4	2.766	1.481	1.344	2.659	2.967	1.345	1.621	1.338	1.352
5	5.042	2.904	0.388	4.69	5.416	2.901	2.907	0.212	0.586
6	1.626	0.893	0.917	1.492	1.746	0.833	1.013	0.797	1.017
7	4.722	2.905	0.102	4.682	4.752	2.903	2.909	-0.012	0.218
8	2.424	1.57	1.303	2.322	2.523	1.511	1.629	1.279	1.324
9	4.991	-0.001	0.103	4.962	5.028	-0.014	0.002	-0.01	0.216
10	2.528	1.548	1.324	2.469	2.589	1.511	1.583	1.313	1.336
11	2.509	1.169	1.323	2.465	2.529	1.131	1.185	1.314	1.331
12	4.345	2.903	0.62	4.306	4.384	2.899	2.905	0.59	0.65
13	2.208	0.001	1.749	2.204	2.212	-0.001	0.001	1.71	1.788
14	2.246	0.001	1.676	2.242	2.25	-0.001	0.002	1.638	1.714
15	2.705	1.175	1.343	2.668	2.742	1.171	1.179	1.34	1.346

Table 6.3: Processed geometric-properties of cracks.

Crack	Basic Properties			Coverage		Inner Location		
No	Area	Length	Width	Cracks	Model	xin	yin	zin
-	m^2	m	m	%	%	m	m	m
1	0.405	49.38	0.008	79.847	0.268	2.646	1.295	1.34
2	0.036	4.463	0.008	7.17	0.024	3.372	1.569	1.26
3	0.027	3.187	0.009	5.386	0.018	4.3	1.439	0.593
4	0.01	1.172	0.009	2.041	0.007	2.767	1.519	1.348
5	0.009	1.225	0.007	1.78	0.006	5.046	2.905	0.435
6	0.005	0.687	0.008	1.019	0.003	1.633	0.898	0.924
7	0.003	0.394	0.007	0.581	0.002	4.686	2.905	0.102
8	0.003	0.362	0.008	0.567	0.002	2.424	1.567	1.305
9	0.003	0.374	0.008	0.561	0.002	5.023	0.001	0.1
10	0.002	0.203	0.008	0.328	0.001	2.526	1.549	1.324
11	0.001	0.126	0.008	0.195	0.001	2.507	1.181	1.323
12	0.001	0.098	0.009	0.168	0.001	4.345	2.903	0.62
13	0.001	0.085	0.007	0.123	0	2.208	0.001	1.749
14	0.001	0.084	0.007	0.121	0	2.245	0.001	1.678
15	0.001	0.074	0.008	0.115	0	2.705	1.176	1.343
Total	0.507	61.916	0.119	100	0.336	**Model Area (m^2):**		150.884

6.10. Conclusions

This paper presents the development of a simple but efficient workflow for the classification of 3D models of masonry structures using semantic-segmentation on renders captured by the program directly. This process offers a simplified approach compared to alternatives (i.e., classification of the point-cloud), with very high-accuracy (as seen by the results), where the software is used to classify any structural-material (if the appropriate CNN model is provided) and measure the geometric properties of detected-defects.

The program developed can have applications for enhanced visualisation of existing-structures (by the means of discretisation of different materials), or automatic generation of computer graphics. Another application is the generation of discretized numerical-models for improved assessment of the structural condition. However, the main application of the developed software is to improve the manual process of visual inspection of heritage buildings and assist with the decision-making regarding renovation and rehabilitation methods.

The main disadvantage of the proposed process is the inability of the software to identify all required cameras for complete classification (i.e., inside tunnels). In which case, the manual incorporation of additional camera-locations is a simple process, that could be improved. A lesser disadvantage is the incorrect-classification of a small number of points, as a leftover of the filtering processes. This is mainly due to the simplicity of the filtering process. Lastly, the detection-accuracy is dependent on the accuracy of the CNN model and every main material-classification requires its own independent model.

Future work can include improvements to the process. For example, a dedicated algorithm to identify large openings on the model and incorporate the additional required cameras, automatically. Another possible improvement is the integration of a more sophisticated algorithm for the filtering of classified points, improving the final output. Lastly, since the classification process is relying on CNN models, the increase of training-data will improve the classification accuracy directly.

6.11. Acknowledgements

This work was funded by the EPSRC project ‘Exploiting the resilience of masonry arch bridge infrastructure: a 3D multi-level modelling framework’ (ref. EP/T001348/1). The financial contribution is very much appreciated.

7. Discussion

The discussion chapter delves into a thorough examination of the results and findings derived from the methodologies presented in the previous chapters. These methodologies have facilitated the integration of computer vision, deep learning, and structural engineering, marking a significant advancement in the field of visualisation, inspection, and assessment of masonry structures. The journey outlined in this thesis spans from the creation of pioneering image processing-based frameworks for analysing 2D masonry structures to the expansion of these techniques into a 3D context.

7.1. Artificial Intelligence (Publication #2)

The aim of the initial application of artificial intelligence was to improve the detection of the micro-geometry of masonry using either thresholding or canny-edge detection. In that regard, a CNN model for the block detection was developed and another model of crack detection was acquired from external research. In general, the accuracy of the detection improved highly, especially in the block-detection model. Thus, the artificial intelligence models were used in all subsequent publications, after their development.

7.1.1. Block Detection

To create the block detection model, a dataset consisting of 107 annotated images of typical brickwork was employed for training and validation. This dataset yielded 2,814 image slices, each measuring 224x224 pixels, with a 50-pixel overlap between slices. The overlap ensured that the accuracy of the edges will be equal to the inner areas, thus improving the result. The dataset was divided into training and validation sets with a ratio of 75% and 25%, respectively. Various CNN architectures (including UNET, LinkNET, FPN, and DLV3+), loss functions (such as WCE, FL, F1L, and BCE), optimizers (including SGD, RMSP, and Adam), and hyperparameters (e.g., Learning Rate and Decay) were tested, to identify the optimal combination. It was observed that the use of Focal-Loss demonstrated exceedingly high loss, but it is assumed to be a bug in the package itself. Mostly because the use of focal-loss didn't hinder training and provided moderate accuracy in most cases.

Among the models evaluated, the DLV3+ model with the RMSP optimizer, coupled with either the F1L or BCE loss functions, demonstrated the highest accuracy, approximately 96.86% for F1L and nearly 96.87% for BCE (Validation Accuracy). Generally, the F1L model is favoured due to its ability to produce cleaner results with less noise, while the BCE model exhibited moderate overfitting in the accuracy/loss graph. Although combining the two models could potentially enhance accuracy, this approach has not yet been tested.

The selection of architectural designs, optimizers, loss functions, and hyperparameters was influenced by existing literature in the field, as well as their availability. The choice of the validation accuracy metric is primarily driven by the fact that both marked (blocks) and unmarked areas (i.e., background, mortar, openings, structural) occupy a large portion of the image, rendering the F1 score less relevant. Additionally, precision and recall were less critical considerations, as the research aims to visualize precise geometry, assigning equal importance to marked and unmarked areas. Consequently, validation accuracy is deemed the most suitable metric.

The quality of the output, of the block-detection model, was tested using a similar approach to IoU (Intersection over Union). However, instead of considering the intersection over the union (as in IoU), the new metrics consider the Intersection over each objects area. That change allows the evaluation of the objects separately (ground-truth vs detected), and thus filtering out objects based on user preference. In general, both were set to retain only objects that have both ratios equal to 80% and higher. The final metric used, to evaluate the quality of the shape, is the area-error, which was given as the ratio-difference of the ground-truth-area over the detected-area and was set to 20% maximum.

All metrics are used to test how many objects have proper shape, removing improper objects, aiming to compare the CNN method with the Image-processing methods (i.e., thresholding). The results demonstrated, that even in ideal photographs of masonry structures (i.e., visible change between materials), the CNN output is slightly more reliable. That difference is increasing, in favour to CNN method, with less ideal images of masonry.

7.1.2. Crack Detection

The original model's training involved 351 images with cracks and 118 images without cracks, resulting in 4,057 slices with cracks and 7,434 slices without cracks (slices without cracks were selected randomly from the pool). The final crack detection model was constructed utilizing the U-NET-MobileNet architecture, the Adam optimizer, and the WCE loss function. The chosen performance metric for the crack detection model is the F1-score, primarily since marked areas (cracks) constitute only a small portion of the image.

It's noteworthy that the accuracy of the crack detection model is moderate and has potential for significant improvement. However, generating sufficient annotated data for cracks presents challenges, as cracks cover a relatively small area of the image, and their shapes tend to be more irregular compared to typical bricks. However, the crack-detection model has not been evaluated or improved since it was externally sourced.

7.2. 2D Workflow (Publications #1, 3, 4)

The 2d workflow is used to generate geometrical and numerical models of masonry structures from a single 2d image. The output of the software also includes the generation of convex mesh, to investigate crack propagation (during the numerical analysis), and the geometric measurement of detected-defects. The inclusion of scaling to the manual parameters automated the completed workflow (other than the scaling itself; publication #3). The optimal input to the developed workflow is an orthorectified image (i.e., generated using photogrammetry) to ensure the accurate dimensions of the final-output. The 2D workflow was improved in every subsequent publication and the final result is a robust and reliable method for the precise documentation of masonry structures.

7.2.1. Feature Detection (2D)

Initially, the accuracy was limited since the detection was using either thresholding or edge-detection. In multiple cases, the application of the software was impossible since the images were not appropriate for the detection of blocks/cracks using image processing. However, this was improved highly with the incorporation of artificial intelligence for the detection of blocks and cracks (developed in publication #2 and applied in publication #3; chapter 3; section 7.1). The remaining classifications (mortar, structural, openings, background) were identified using image processing on the output of the CNN models. The accuracy of the remaining classifications is excellent. However, the separation of other structural elements (i.e., concrete, steel) with the openings/background requires a filtered image to identify the background area individually (i.e., white background colour). White background is generally acceptable since the output from photogrammetric applications can conveniently include white background, and thus, facilitate the identification of other structural elements.

7.2.2. Mesh Generation (2D)

The internal generation of mesh (section 4.11) allowed the investigation of crack-propagation and removed the necessity to create the mesh externally. The mesh generated is also always convex, which avoids expected issues with interlocking elements. The mesh-size tested for validation was equal to 20mm and in the later subchapter 100mm (section 7.4). The smaller mesh-size is expected to provide more accuracy, since it allows for more complex paths of crack-propagation during the numerical analysis.

7.2.3. Damage Measurement (2D)

The damage detected using CNN is measured to provide a reliable tool for the evaluation of damage in a structure (section 4.8). Those include the location, area, length, average width, and coverage of damage in the image provided. The coverage is providing the ratio of the damage area compared to the whole masonry area, structure, and image. The accuracy of the crack measurements is equal to ±1 pixel, assuming an ideal detection from the CNN model. The damage-metrics can be used for the evaluation of damage on masonry structures since they denote the severity of damage and can identify important cracks individually (i.e., if the length, width, or area exceed a certain threshold provided). The output is given in .CSV format for convenience.

7.2.4. Shape Adjustments (2D)

The research used a generalization algorithm, developed specifically for the purpose of improving the workflow (section 2.3.2). The generalization algorithm developed reduces the number of vertices used to describe a detected object, while trying to retain an accurate representation of the object. Thus, improving the useability of the output for numerical analysis by reducing the complexity of the input. The difference between the developed and existing generalization algorithm (RDP), is that the developed algorithm can retain vertices that satisfy large horizontal-thresholds and not only vertical-thresholds. Thus, retaining an accurate depiction of a small number of complex shapes (rarely appearing but may be important). Furthermore, the shape of the simplified models (mortar as a zero thickness interfaces) was improved by the development of an algorithm that checks and relocates vertices to improve the shape of the surface of connected objects (section 2.3.3), when they are expected to be straight (i.e., the upper surface of two connected blocks). The programming logic, behind the algorithm for the surface adjustments, adjusts only small angles. Since most masonry structures use typical brickwork (i.e., not rubble, with horizontal/vertical surfaces), the algorithm improves the output significantly.

7.2.5. Computational time (2D)

The 2D workflow provides extremely fast results, ranging from several seconds to several minutes (i.e., ~151s for the model in section 7.4), depending on image resolution, number of elements, selection of program options, and computational power (i.e., laptop with: i7-9750h CPU, Nvidia RTX 2060, 32GB Ram). The parts that require the most time are the: Block detection (due to the DLV3+ complexity; ~26s); crack detection (Due to the CNN model; ~14s); segmentation adjustments (mainly due to segmentation corrections; ~33s); contour detection (mainly due to the point-detection; ~27s); and model generation (mainly due to mesh generation, but for finer meshes; ~6s). Although the external packages used may include multiprocessing capabilities, the developed algorithms do not currently incorporate multiprocessing. Therefore, the computational time can be significantly reduced if multiprocessing capabilities are integrated into the developed algorithms. Additionally, some processes may be programmatically optimized to improve their computational efficiency, especially since the programs were primarily designed for research purposes rather than commercial use.

7.2.6. Improving traditional Inspection and Assessment Workflows (2D)

The workflow developed can separate between materials and generate the accurate geometry of the target. Although there are defects/materials not identified explicitly, those can be added directly to workflow (i.e., in the form of additional CNN models) to provide a complete visualization of the structure. Since the materials can be identified and generate their boundaries (i.e., in the form of polylines), their metrics can be evaluated to identify locations of interest (such as the example of measurement of detected cracks; section 4.8). Thus, allowing for automatic and unbiased inspection of structures (i.e., by evaluating the length of cracks compared to the area of the structure). Lastly, the workflow can be used to automatically generate the numerical model of the structure (i.e., using a stationary camera), assess the structural integrity, and warn about potential issues. Thus, providing an automatic method for the assessment of masonry.

7.3. 3D Workflow (Publication #5)

The 3D workflow generates the classified point cloud of the structure (blocks, mortar, structural, cracks), taking as input the textured/coloured 3D reality-mesh of a masonry structure. The workflow exploits previously developed algorithms for the classification, while translating the classified pixels to 3D points. Additionally, the 3D workflow has the ability to measure detected defects. The combination of the features allows the use of the developed workflow to improve automation in the inspection oof structures, but not numerical assessment since it cannot generate the geometrical/numerical model.

7.3.1. Feature Detection (3D)

The feature detection in the 3d workflow is applied using the CNN models developed in previous research (chapter 3), to identify blocks and cracks. The remaining classifications are detected using image-processing. The classification is applied on renders around the structure, captured automatically. The classified pixels are converted to classified points using transformation matrices. The 2D approach (for the classification) has multiple advantages compare to the alternative method (i.e., classification of point-clouds). Mainly, the acquisition of image-samples is faster and simpler than the acquisition of point-clouds of structures. Additionally, the generation of training/validation data is simpler since it is applied on the 2D plane. Those advantages allowed the developed workflow to provide a highly accurate result, with generally small effort (regarding the training/validation of the CNN models).

7.3.2. Damage Measurements (3D)

The detected cracks area meshed to measure their geometric properties, using a similar approach to the 2D workflow but with completely different algorithmic implementation (section 6.9). In this case, the 2D algorithms could not be applied on 3D, but the newly developed algorithms follow a similar notion. Initially the detected cracks are meshed with the aim to measure them. The area and location are the first metrics acquired (since they do not need any further action). Then, each individual mesh of cracks is linearised to measure their length and calculate their average width. Since the mesh generated is in 3D space, the reliability of the results is better than the 2D approach, since it considers cracks expanding to multiple planes.

7.3.3. Computational time (3D)

The computational time of the 3D workflow varies greatly (i.e., 6,640s for the model in section 6.8, with 13x cameras), depending on model size, resolution, selection of program options, and computational power. The parts of the workflow that take the longest time are the rendering of image captures (for 13x cameras; ~619s), block detection (using CNN; ~1,573s), crack detection (if CNN is used; ~2,016s), and point-combination (including camera-filtering but not point-filtering; ~1,703s). Similar to the 2D workflow, the developed algorithms do not incorporate multiprocessing. Thus, incorporating multiprocessing will reduce the computational time significantly. Moreover, certain processes can be programmatically optimized to enhance their computational efficiency, and consequently reduce computational time even further.

7.3.4. Automating Traditional Inspection and Assessment Workflows (3D)

Similar to the 2D approach, the workflow has the capability to assist with the visual inspection aspect, since the individual materials/defects can be identified and measured individually. However, not all materials/defects are detected. Although the addition of more materials/defects can easily be integrated within the workflow in the form of CNN models trained to identify them. That way, the workflow can provide a complete and unbiased inspection of structures. However, since the geometrical model is not generated, the workflow cannot assist with the numerical assessment of the structure to ensure its structural integrity. Future work may improve the workflow to add geometrical model generation (i.e., similar to crack meshing in section 6.9).

7.4. Validation of Image2DEM (2D)

The final version of the 2D workflow has not been validated during the evolution of the procedure. Although the original workflow was verified in the first publication, that doesn't include the detection using CNN and the internal mesh-generation. Thus, the complete process is tested in this section, to verify the applicability of the proposed workflow.

7.4.1. Geometry

The geometry selected, to investigate the workflow, is the same used during the validation in the 1st publication (section 2.5). The specific geometry was selected since it includes both mortar and damage, to verify all aspects of the workflow. The validation compares the idealised (created manually) against the generated geometry (generated by the software automatically). The geometry of the idealised model is the same as in the original publication, with the exception of the mortar-mesh. The mesh was modified to resemble the mesh produced for the generated model. The seed for the mesh is equal to 100mm. That way the analysis is considerably faster (than using 10 or 20mm seed), at the cost of a small reduction in accuracy due to the large size of the mesh-seed (since finer mesh would produce more possible crack-propagation paths). The geometry was simplified using the dynamic generalisation-thresholds, with values equal to: $Vth = Hth = Lth/2 = 0.01$ (Eq. (55)); section 4.10). The generalisation produced a simplified model, while still retaining high geometrical-accuracy. Additionally, the generated cracks-mask was substituted with the original cracks-mask, to ensure equal damage and enable a more accurate comparison. The main reason, for not utilizing the output of the CNN model, is that the crack-detection model is not a part of the broader research scope and therefore does not necessitate validation.

a)

b)

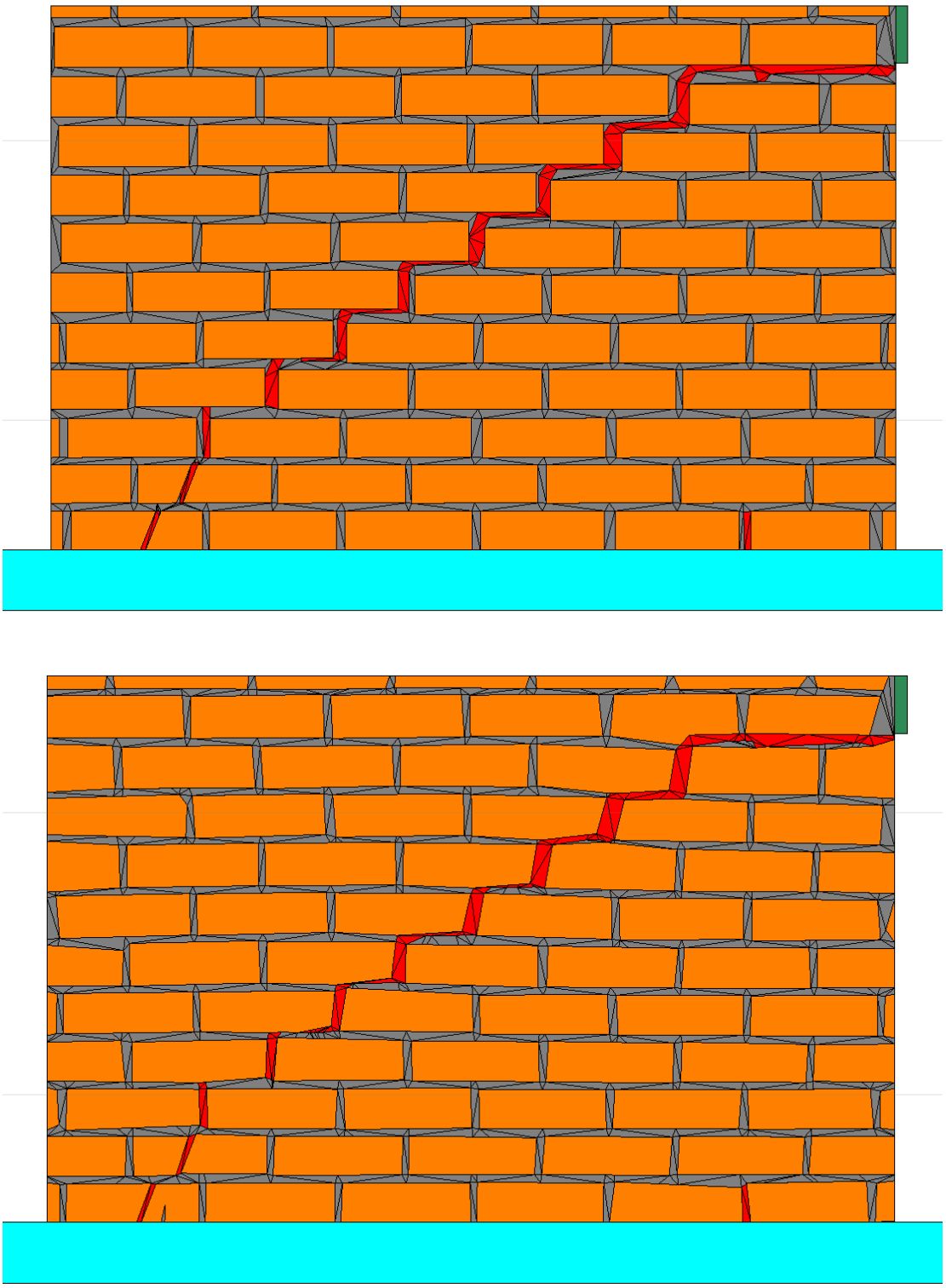


Fig. 7.1: Numerical model in UDEC7; a) Idealised Geometry; b) Generated Geometry.

7.4.2. Numerical Analysis

Similar to the numerical validation of the 1st publication, the damaged areas were removed completely to simulate the loss of material (Fig. 7.1; Fig. 7.3). As mentioned previously, the material, joint, and other properties of the model are the same as in the original work. The application of the

force was provided as a horizontal velocity of -0.01 m/s, acting at the green-plate on the top-right corner of the wall.

Regarding the analysis itself, the model ran in total for 1sec mechanical time (or 0.1m). The force-displacement graph, of the idealised and generated cases, shows very similar reaction to the applied force. In fact, the graphs are in general parallel, showing similar peak and reduction. The maximum force recorded is equal to 24.556kN (@1.01mm) and 26.729kN (@1.17mm; +8.8% force), for the idealised and generated cases respectively (Fig. 7.2). The difference in maximum load is small, although higher than the original comparison. However, the shape of the mortar-mesh is slightly different between the 2 cases (the idealised-mesh was designed by hand, to replicate the generated mesh). Additionally, the mesh is larger than the original model, which is assumed that it reduces the accuracy of the model. Nonetheless, the change in peak-load, between the models, is relatively small.

The deformed-geometry, at 10mm displacement, demonstrates the same reaction to the applied force (Fig. 7.3). The formation of cracks is appearing at the same locations, for both cases. The type of failure is also the same between the 2 models, at multiple locations (i.e., rotation in the bottom-left and top-mid corners; sliding in the mid-left side). The only slight difference is the exact mesh-element separated, although the mesh is not the same between the 2 models anyway.

The similarity between the 2 numerical models (of the idealized and generated geometries), showcases that the complete workflow can be used for the reliable assessment of masonry structures. Although a small difference in peak load was noted (+8.8% force), it is expected that the difference would be less significant if a finer mesh would be used.

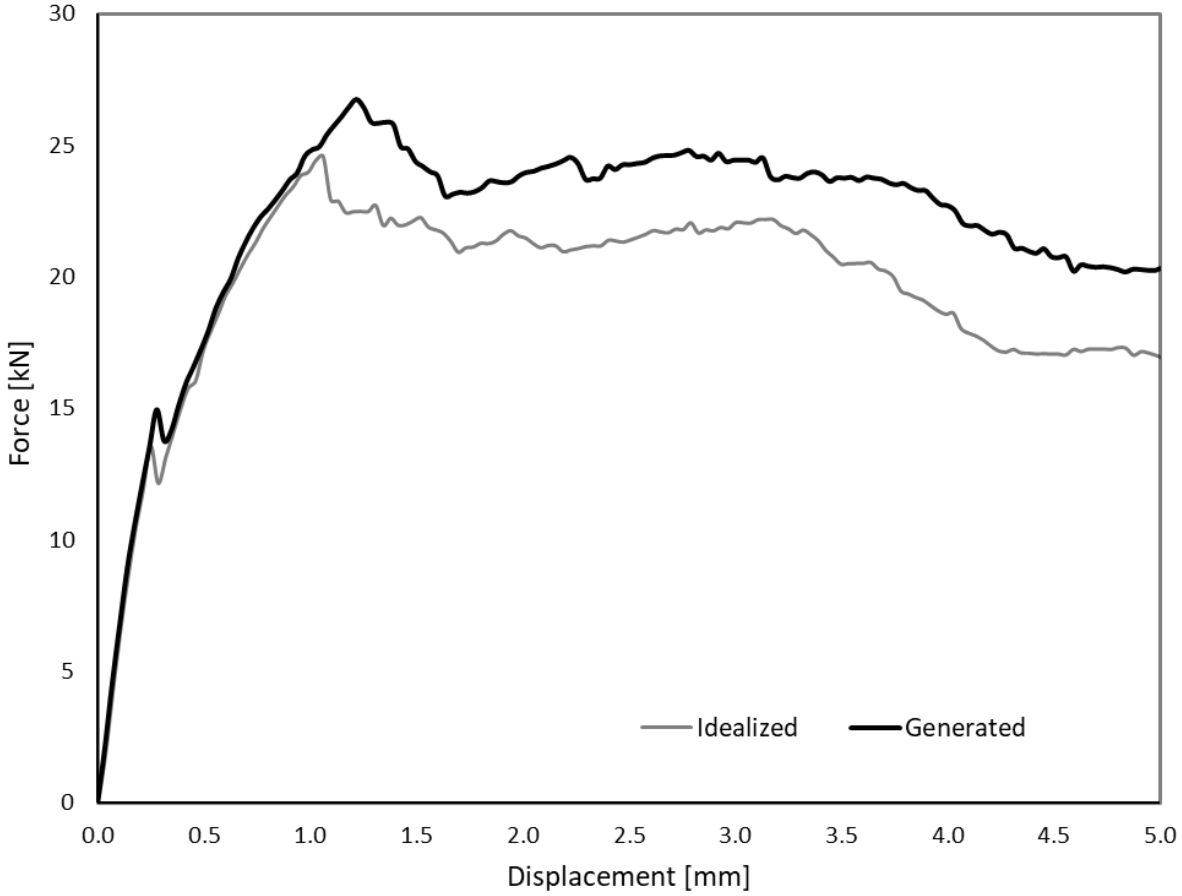
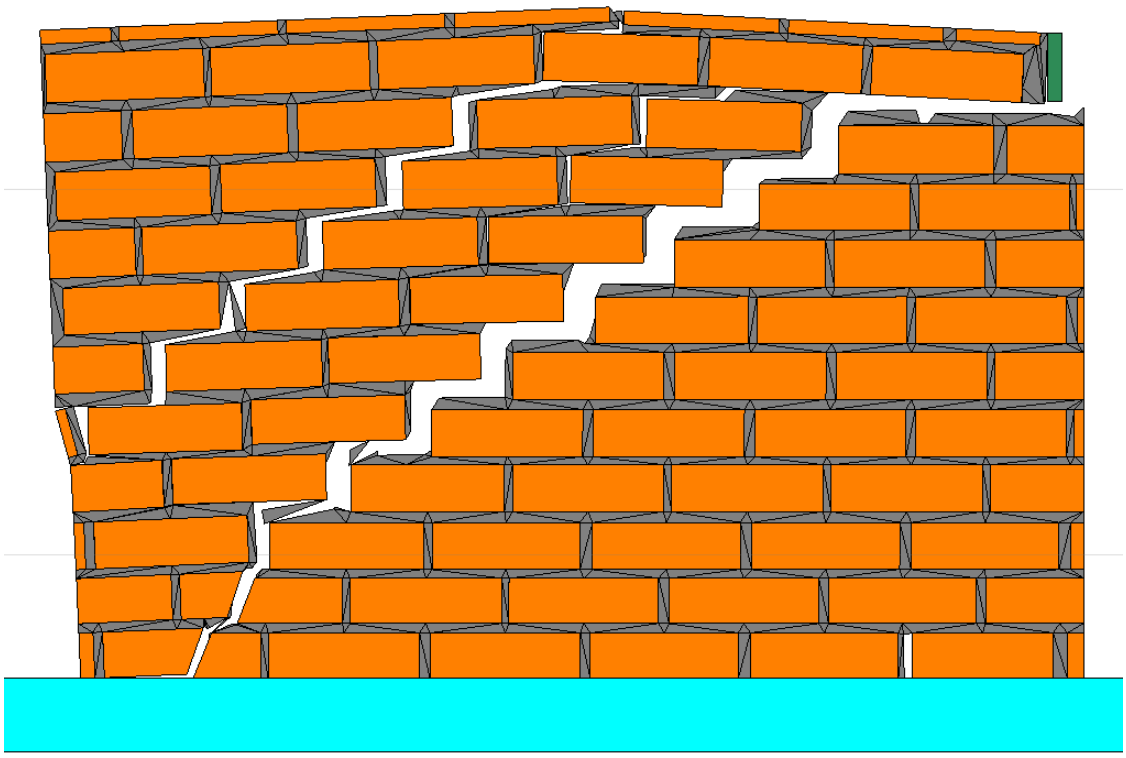


Fig. 7.2: Force-Displacement graph at 0.005 meters displacement (5secs).

a)



b)

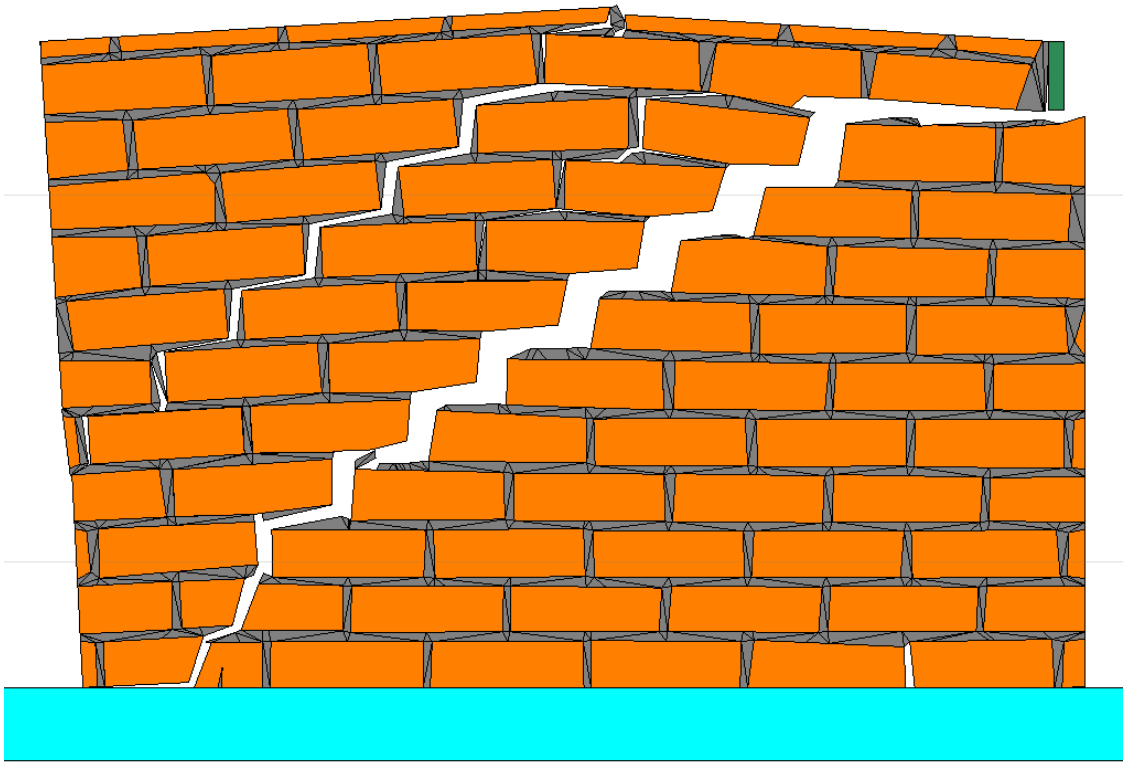


Fig. 7.3: Deformations after 10mm displacement (x5 deformation factor); a) Idealised Model; b) Generated Model.

8. Conclusions

The work presented in this thesis demonstrates efforts to automate typical inspection and assessment workflows, by incorporating an innovative approach that utilizes visual data of masonry structures. The novelty of the approach includes the identification and development of complex algorithms, with simplified user-application, to ensure their applicability in actual engineering conditions. Additionally, the study also investigates the use of the accurate geometry for the numerical assessment of masonry.

The initial research focuses on generating CAD drawings and numerical models from 2D images of masonry structures (Chapter 2; Objective #1). Feature detection is applied using image processing operations, such as edge detection and thresholding, which are then converted into a collection of polylines. However, it was noticed that the detection of the micro-geometry of masonry is unreliable with image processing due to variations in colour, illumination, and brightness in the images. Additionally, it was observed that the CAD-generated mesh could cause issues during numerical analysis due to concavity in certain regions. The complete workflow was validated to ensure that the generated models could be used for numerical analysis, with exceptional results.

The following research investigates the use of artificial intelligence for the detection of blocks and cracks using CNN (Chapter 3; objective #2; objective #4). The detection rate, compared to image processing operations, was found to be more accurate and reliable, thus addressing the major issue identified in the initial research (objective #3). The evaluation of detected cracks was also examined, but the acquired metrics for cracks were approximate. The CNN models, either generated during the research or acquired from external sources, were incorporated into the original algorithm (Chapter 4; objective #5), improving the accuracy of geometry generation from input images. The remaining classifications (mortar, structural, background) were estimated using image-processing on the CNN classifications (objective #6). However, the estimation of the structural elements is not always accurate since it does not separate between common structural materials (i.e., concrete, steel). Furthermore, the updated software included a new mesh generation that ensured the convexity of every mesh element, thereby enhancing accuracy during numerical analysis. The updated software also featured a more precise evaluation of the geometry of detected cracks (objective 7), by measuring the linearized cracks. Most user options were simplified with the inclusion of a scale parameter, which automated most software variables. The later publication incorporated a complete GUI layout (Chapter 5), further simplifying user input, with the only required input being the scale of the image (provided in the GUI). The quality of the geometry in the numerical model was validated in the supplementary chapter (see Chapter 7.4). The final/updated workflow resulted in a fast, reliable, and convenient method for generating 2D geometrical/numerical models of masonry structures from image-data.

Furthermore, additional efforts were made to extend the application of the algorithms from 2D to a 3D environment (Chapter 6; objective #8). An additional program was developed that leveraged deep learning and image processing techniques to identify the micro-geometry of masonry in a 3D environment. Since feature detection is applied using CNN on image data, the generation of training data is straightforward, accessible, and rapid compared to alternatives, such as the classification of point clouds. The final output is the generation of a classified point cloud of masonry structures, identifying the locations of masonry units, mortar, damage, and other structural elements. Similar to the 2D approach, damage is also measured to provide information about the structure's current condition (objective #9). As the geometric properties of damage are estimated in a 3D environment, the measurement is considered more accurate, accounting for continuity between different capture planes, assuming accurate detection of damaged areas. The methodology also includes a simple point filtering method to address common classification issues. However, the filtering method allows for a small number of mis-classified pixels, incorrectly classified due to proximity to a classified body, creating incorrect thin-lines (by dragging the classification to deeper depths than the classified body).

8.1. Possible Applications

The applications of the developed algorithms are multiple and range from engineering to digital-visualisation. The main purpose, of the initial work, is to generate high-accuracy numerical models with little effort. Mainly, to ensure the accurate assessment of the structural-integrity, in a timely manner. A secondary application of the algorithms is to assist engineers with the visual-inspection of heritage-structures by identifying the severity of damage. That can improve maintenance workflows in terms of physical labour, time-requirements, and monetary-costs. The proposed algorithms can also be used for documentation of the structure over specified time-intervals (for comparison between different time-periods). The documentation over specified-intervals can be fully automated with the use of drones (i.e., with programmable fly-routes) or stationary cameras, and thus, providing the means to generate a digital-twin of the structure. Other applications of the algorithm include the geometry generation for realistic visualisation on-demand (i.e., CAD, videogames, presentations).

8.2. Advantages and Disadvantages

The main advantage of the proposed algorithms is the speed and reliability that they offer for their applications. Since the feature-detection is applied using semantic segmentation with deep-learning, the acquisition of training data is simple, and thus, the improvement of the detection-accuracy. Acquiring training data is particularly straightforward in the case of the final publication (chapter 6), whereas the alternative approach necessitates obtaining and manually-annotating 3D point-clouds, which is both more challenging to acquire and more complex to annotate. Furthermore, the process has been highly automated and requires minimal manual input in both cases (2D and 3D algorithms).

The main limitation of the algorithms proposed is the accuracy of the CNN models since they are the main source of feature-detection (blocks and cracks). The remaining classifications (background, openings, and other-structural-materials) are a result of image-processing that rely heavily on the initial classifications. More specifically, it has been noted that the trained models have reduced accuracy on identifying bricks on substantial discoloured regions, especially of dark colour. The inaccuracy of the model is reduced on discolouration of brighter colour. Additionally, the trained models cannot distinguish elements behind foreground-objects (i.e., labels, pipes). Furthermore, only blocks and cracks have been included on the possible classifications, and thus, the documentation cannot assign the regions of other materials (i.e., concrete, steel) and defects (i.e., discolouration, spalling). Lastly, the block-detection model has been trained mainly to typical brickwork, and thus, the detection of other types of masonry (i.e., stonework, rubble) is inaccurate.

8.3. Future Work and Improvements

The algorithms developed during the research can be improved, mainly in terms of output quality, useability, reliability, and performance. A minor improvement is the optimisation of the 2D shape extracted from blocks. Although an optimisation to the shape has already been applied, it considers only one pair of parallel lines (usually the longest side). Major improvements include the addition of more classification of defects (i.e., discolouration, spalling, vegetation), masonry types (i.e., stonework), and structural elements (i.e., concrete, steel). The inclusion of common materials and defects, to the feature-detection, would enhance the overall visualization of the structure and its structural condition. Furthermore, the 3D classification of the masonry structure can also consider the generation of a 3-dimensional numerical model, which would highly improve the workflow of assessment of masonry structures. Lastly, the point-filtering of the point-cloud may be improved with the application of a more complex filtering-method (i.e., by considering the neighbour classifications).

9. References

Albertz, J. (2007) 'A Look Back; 140 Years of Photogrammetry', 73, pp. 504–506. Available at: https://doi.org/https://doi.org/10.1080/10095020.2020.1843375.

Alessandri, C. *et al.* (2015) 'Crack patterns induced by foundation settlements: Integrated analysis on a renaissance masonry palace in Italy', *International Journal of Architectural Heritage*, 9(2), pp. 111–129. Available at: https://doi.org/10.1080/15583058.2014.951795.

Ali, L. (2019) 'Damage Detection and Localization in Masonry Structure Using Faster Region Convolutional Networks', *International Journal of GEOMATE*, 17(59), pp. 98–105. Available at: https://doi.org/10.21660/2019.59.8272.

Altuntas, C., Hezer, S. and Kırlı, S. (2017) 'IMAGE BASED METHODS FOR SURVEYING HERITAGE OF MASONRY ARCH BRIDGE WITH THE EXAMPLE OF DOKUZUNHAN IN KONYA , TURKEY', *SCIENTIFIC CULTURE*, 3(2), pp. 13–20. Available at: https://doi.org/10.5281/zenodo.438183.

Andriasyan, M. *et al.* (2020) 'From point cloud data to Building Information Modelling: An automatic parametric workflow for heritage', *Remote Sensing*, 12(7), p. 1094. Available at: https://doi.org/10.3390/rs12071094.

Angelillo, M. (2019) 'The model of Heyman and the statical and kinematical problems for masonry structures', *International Journal of Masonry Research and Innovation*, 4(1–2), pp. 14–21. Available at: https://doi.org/10.1504/IJMRI.2019.096820.

Arbeláez, P. *et al.* (2011) 'Contour detection and hierarchical image segmentation', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5), pp. 898–916. Available at: https://doi.org/10.1109/TPAMI.2010.161.

Asteris, P.G. *et al.* (2015) 'Numerical modeling of historic masonry structures', in B. DeMarco, Austin; Wolfe, Kayla; Henning, Christina; Carbaugh (ed.) *Handbook of Research on Seismic Assessment and Rehabilitation of Historic Structures*. United States of America: Engineering Science Reference (an imprint of IGI Global), pp. 213–256. Available at: https://doi.org/10.4018/978-1-4666-8286-3.ch007.

Bagińska, M., Srokosz, P. and Srokosz, P.E. (2019) 'The Optimal ANN Model for Predicting Bearing Capacity of Shallow Foundations Trained on Scarce Data', pp. 130–137. Available at: https://doi.org/https://doi.org/10.1007/s12205-018-2636-4.

Bal, İ.E. *et al.* (2021) 'Novel invisible markers for monitoring cracks on masonry structures', *Construction and Building Materials*, 300, p. 124013. Available at: https://doi.org/10.1016/j.conbuildmat.2021.124013.

Barazzetti, L. *et al.* (2015) 'Cloud-to-BIM-to-FEM: Structural simulation with accurate historic BIM from laser scans', *Simulation Modelling Practice and Theory*, 57, pp. 71–87. Available at: https://doi.org/10.1016/j.simpat.2015.06.004.

Bassier, M. *et al.* (2019) 'Semi-automated Creation of Accurate FE Meshes of Heritage Masonry Walls from Point Cloud Data', *RILEM Bookseries*, 18, pp. 305–314. Available at: https://doi.org/10.1007/978-3-319-99441-3_32.

Bassier, M. and Vergauwen, M. (2020) 'Unsupervised reconstruction of Building Information Modeling wall objects from point cloud data', *Automation in Construction*. Elsevier B.V., p. 103338. Available at: https://doi.org/10.1016/j.autcon.2020.103338.

BEIS (2021) *Net Zero Strategy: Build Back Greener*, *Gov.Uk*. Available at: https://www.gov.uk/government/publications/net-zero-strategy.

Beucher, S.; Meyer, F. (1993) 'Advances of mathematical morphology in image processing', in *Mathematical Morphology in Image Processing*. New York: Marcel Dekker Inc, pp. 433–481. Available at: https://doi.org/10.1201/9781482277234-12.

Bora, D.J. (2017) 'A Novel Approach for Color Image Edge Detection Using Multidirectional Sobel International Journal of Computer Sciences and Engineering Open Access A Novel Approach for Color Image Edge Detection Using Multidirectional Sobel Filter on HSV Color Space', *International Journal of Computer Sciences and Engineering*, 5(2), pp. 154–159. Available at: https://doi.org/10.6084/m9.figshare.4732951.

Brackenbury, D., Brilakis, I. and Dejong, M. (2019) 'Automated defect detection for masonry arch bridges', *International Conference on Smart Infrastructure and Construction 2019, ICSIC 2019: Driving Data-Informed Decision-Making*, 2019(1), pp. 3–10. Available at: https://doi.org/10.1680/icsic.64669.003.

Brackenbury, D. and Dejong, M. (2018) 'Mapping Mortar Joints in Image Textured 3D Models to Enable Automatic Damage Detection of Masonry Arch Bridges', in *17th International Conference on Computing in Civil and Building Engineering*. Tampere, Finland, pp. 173: 530-545. Available at: http://programme.exordo.com/icccbe2018/delegates/presentation/344/ (Accessed: 27 January 2021).

Brown, D. (2024) 'Open letter to launch PR23', pp. 1–8. Available at: https://www.orr.gov.uk/sites/default/files/2021-06/2021-06-17-pr23-launch-letter.pdf.

Cabaleiro, M. *et al.* (2017) 'Algorithm for automatic detection and analysis of cracks in timber beams from LiDAR data', *Construction and Building Materials*, 130, pp. 41–53. Available at: https://doi.org/10.1016/j.conbuildmat.2016.11.032.

Caliò, I., Marletta, M. and Pantò, B. (2012) 'A new discrete element model for the evaluation of the seismic behaviour of unreinforced masonry buildings', *Engineering Structures*, 40, pp. 327–338. Available at: https://doi.org/10.1016/j.engstruct.2012.02.039.

Canny, J. (1986) 'A Computational Approach to Edge Detection', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), pp. 679–698. Available at: https://doi.org/10.1109/TPAMI.1986.4767851.

Chaiyasarn, K. *et al.* (2018) 'Crack detection in historical structures based on Convolutional Neural Network', *International Journal of GEOMATE*, 15(51), pp. 240–251. Available at: https://doi.org/10.21660/2018.51.35376.

Chaurasia, A. and Culurciello, E. (2018) 'LinkNet: Exploiting encoder representations for efficient semantic segmentation', *2017 IEEE Visual Communications and Image Processing, VCIP 2017*, 2018-Janua(April), pp. 1–4. Available at: https://doi.org/10.1109/VCIP.2017.8305148.

Chen, L.C. *et al.* (2015) 'Deeplab: Semantic image segmentation with deep convolutional nets and fully connected CRFs', *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 40(4), pp. 834–848. Available at: https://doi.org/https://doi.org/10.48550/arXiv.1412.7062.

Chen, L.C. *et al.* (2018) 'DeepLabv3+: Encoder-decoder with atrous separable convolution for semantic image segmentation', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11211 LNCS, pp. 833–851. Available at: https://doi.org/10.1007/978-3-030-01234-2_49.

Cluni, F. *et al.* (2015) 'Enhancement of thermographic images as tool for structural analysis in earthquake engineering', *NDT and E International*, 70(2015), pp. 60–72. Available at: https://doi.org/10.1016/j.ndteint.2014.10.001.

Cundall, P.A. (1971) 'A computer model for simulating progressive large-scale movements in blocky rock systems', in *Proocedings of the Symposium of the International Society of Rock Mechanics*. Nancy, France, p. Vol. 1., Paper No. II-8. Available at: https://www.scirp.org/(S(i43dyn45teexjx455qlt3d2q))/reference/ReferencesPapers.aspx?ReferenceID=1230197 (Accessed: 27 January 2021).

Dais, D. *et al.* (2021) 'Automatic crack classification and segmentation on masonry surfaces using convolutional neural networks and transfer learning', *Automation in Construction*, 125, p. 103606. Available at: https://doi.org/10.1016/j.autcon.2021.103606.

D'Altri, A.M. *et al.* (2018) 'A 3D detailed micro-model for the in-plane and out-of-plane numerical analysis of masonry panels', *Computers and Structures*, 206, pp. 18–30. Available at: https://doi.org/10.1016/j.compstruc.2018.06.007.

D'Altri, A.M. *et al.* (2020) 'Modeling Strategies for the Computational Analysis of Unreinforced Masonry Structures: Review and Classification', *Archives of Computational Methods in Engineering*, 27(4), pp. 1153–1185. Available at: https://doi.org/10.1007/s11831-019-09351-x.

Douglas, D.H. and Peucker, T.K. (1973) 'ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE', *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2), pp. 112–122. Available at: https://doi.org/10.3138/fm57-6770-u75u-7727.

Eaton, J., Edwards, M. and Crapper, M. (2014) *Heritage Railway Association: The Inspection and Maintenance of Civil Engineering Assets*. Edinburgh: Heritage Railway Association (HRA). Available at: https://static1.squarespace.com/static/59f1c5ec51a58457c01eaed0/t/5a12c2e253450af6e5f4cdd5/1511179012245/HGR-A0701-Is01+-+Inspection+and+Maintenance+of+Civil+Engineering+Assets+S.pdf (Accessed: 12 August 2021).

Erdogmus, E. *et al.* (2019) 'Analysis of the Last Standing Arch of the Roman Aqueduct at Blaundos', in P.B. Dillion and F.S. Fonseca (eds) *13th North American Masonry Conference*. Salt Lake City, Utah, pp. 483–493. Available at: https://www.researchgate.net/publication/334001391_Analysis_of_the_Last_Standing_Arch_of_the_Roman_Aqueduct_at_Blaundos (Accessed: 27 January 2021).

Erdogmus, E. *et al.* (2020) 'Reverse Engineering a Fully Collapsed Ancient Roman Temple through Geoarchaeology and DEM', *International Journal of Architectural Heritage*, 00(00), pp. 1–21. Available at: https://doi.org/10.1080/15583058.2020.1728593.

Ergün Hatir, M. and İnce, İ. (2021) 'Lithology mapping of stone heritage via state-of-the-art computer vision', *Journal of Building Engineering*, 34(November 2020), p. 101921 Contents. Available at: https://doi.org/10.1016/j.jobe.2020.101921.

Ferrante, A. *et al.* (2021) 'Discontinuous approaches for nonlinear dynamic analyses of an ancient masonry tower', *Engineering Structures*, 230(November 2020), p. 111626. Available at: https://doi.org/10.1016/j.engstruct.2020.111626.

Forgács, T., Sarhosis, V. and Bagi, K. (2017) 'Minimum thickness of semi-circular skewed masonry arches', *Engineering Structures*, 140(2009), pp. 317–336. Available at: https://doi.org/10.1016/j.engstruct.2017.02.036.

Forgács, T., Sarhosis, V. and Bagi, K. (2018) 'Influence of construction method on the load bearing capacity of skew masonry arches', *Engineering Structures*, 168(March), pp. 612–627. Available at: https://doi.org/10.1016/j.engstruct.2018.05.005.

Funari, M.F. *et al.* (2021) 'A parametric scan-to-FEM framework for the digital twin generation of historic masonry structures', *Sustainability (Switzerland)*, 13(19), p. 11088. Available at: https://doi.org/10.3390/su131911088.

Garcia-Garcia, A. *et al.* (2017) 'A Review on Deep Learning Techniques Applied to Semantic Segmentation', pp. 1–23. Available at: https://doi.org/https://doi.org/10.48550/arXiv.1704.06857.

Ghaboussi, J. and Barbosa, R. (1990) 'Three-dimensional discrete element method for granular materials', *International Journal for Numerical and Analytical Methods in Geomechanics*, 14, pp. 451–472. Available at: https://doi.org/10.1002/nag.1610140702.

Heyman, J. (1966) 'The stone skeleton', *International Journal of Solids and Structures*, 2(2), pp. 249–256, IN1–IN4, 257–264, IN5–IN12, 265–279. Available at: https://doi.org/10.1016/0020-7683(66)90018-7.

Hinks, T. *et al.* (2013) 'Point Cloud Data Conversion into Solid Models via Point-Based Voxelization', *Journal of Surveying Engineering*, 139(2), pp. 72–83. Available at: https://doi.org/10.1061/(asce)su.1943-5428.0000097.

Historic England (2017) 'Photogrammetric Applications for Cultural Heritage', *Guidance for Good Practice*, pp. 1–124. Available at: https://historicengland.org.uk/images-books/publications/photogrammetric-applications-for-cultural-heritage/ (Accessed: 31 August 2021).

Historic England (2018) '3D Laser Scanning for Heritage', *Advice and Guidance on the Use of Laser Scanning in Archaeology and Architecture*, pp. 1–113. Available at: https://historicengland.org.uk/images-books/publications/3d-laser-scanning-heritage/heag155-3d-laser-scanning/ (Accessed: 31 August 2021).

Hussain, M., Bird, J.J. and Faria, D.R. (2019) 'A study on CNN transfer learning for image classification', *Advances in Intelligent Systems and Computing*, 840(June), pp. 191–202. Available at: https://doi.org/10.1007/978-3-319-97982-3_16.

Iannuzzo, A. *et al.* (2018) 'Crack patterns identification in masonry structures with a C° displacement energy method', *Int. J. Masonry Research and Innovation*, 3(3), pp. 295–323. Available at: https://doi.org/10.1504/IJMRI.2018.093490.

Ibrahim, Y., Nagy, B. and Benedek, C. (2019) 'Cnn-based watershed marker extraction for brick segmentation in masonry walls', in A. Karray, F.; Campilho, A.; Yu (ed.) *Image Analysis and Recognition. ICIAR 2019. Lecture Notes in Computer Science*. Waterloo, ON, Canada: Springer, Cham, pp. 332–344. Available at: https://doi.org/10.1007/978-3-030-27202-9_30.

Itasca (2019) *ITASCA Consulting Limited*. Available at: https://www.itasca.co.uk/ (Accessed: 22 September 2020).

Kalfarisi, R., Wu, Z.Y. and Soh, K. (2020) 'Crack Detection and Segmentation Using Deep Learning with 3D Reality Mesh Model for Quantitative Assessment and Integrated Visualization', *Journal of Computing in Civil Engineering*, 34(3), p. 04020010. Available at: https://doi.org/10.1061/(asce)cp.1943-5487.0000890.

Karagianni, A. *et al.* (2010) 'ELASTIC PROPERTIES OF ROCKS', *Bulletin of the Geological Society of Greece*, 43(3), pp. 1165–1168. Available at: https://doi.org/10.12681/bgsg.11291.

Kassotakis, N. *et al.* (2020) 'Three-dimensional discrete element modelling of rubble masonry structures from dense point clouds', *Automation in Construction*, 119(February), p. 103365. Available at: https://doi.org/10.1016/j.autcon.2020.103365.

Kassotakis, N. *et al.* (2021) 'Quantifying the effect of geometric uncertainty on the structural behaviour of arches developed from direct measurement and Structure-from-Motion (SfM) photogrammetry', *Engineering Structures*, 230(September 2020), p. 111710. Available at: https://doi.org/10.1016/j.engstruct.2020.111710.

Kassotakis, N. and Sarhosis, V. (2021) 'Employing non-contact sensing techniques for improving efficiency and automation in numerical modelling of existing masonry structures: A critical literature review', *Structures*. Elsevier Ltd, pp. 1777–1797. Available at: https://doi.org/10.1016/j.istruc.2021.03.111.

Kornilov, A.S. and Safonov, I. V. (2018) 'An overview of watershed algorithm implementations in open source libraries', *Journal of Imaging*, 4(10), p. 123. Available at: https://doi.org/10.3390/jimaging4100123.

Korumaz, M. *et al.* (2017) 'An integrated Terrestrial Laser Scanner (TLS), Deviation Analysis (DA) and Finite Element (FE) approach for health assessment of historical structures. A minaret case study', *Engineering Structures*, 153(October), pp. 224–238. Available at: https://doi.org/10.1016/j.engstruct.2017.10.026.

Lin, T.Y. *et al.* (2017) 'Feature pyramid networks for object detection', *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua, pp. 936–944. Available at: https://doi.org/10.1109/CVPR.2017.106.

Long, J., Shelhamer, E. and Darrell, T. (2015) 'Fully convolutional networks for semantic segmentation', *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June, pp. 431–440. Available at: https://doi.org/10.1109/CVPR.2015.7298965.

Lourenço, P.B. (1996) *Computational strategies for masonry structures*, *PhD Thesis*. Available at: http://www.narcis.nl/publication/RecordID/oai:tudelft.nl:uuid:4f5a2c6c-d5b7-4043-9d06-8c0b7b9f1f6f (Accessed: 27 January 2021).

Lourenço, P.B. (2013) 'Computational strategies for masonry structures: multi-scale modeling, dynamics, engineering applications and other challenges', in *Congreso de Métodos Numéricos en Ingeniería*, pp. 451–472. Available at: http://repositorium.sdum.uminho.pt/handle/1822/26547.

Loverdos, D. *et al.* (2021a) 'An innovative image processing-based framework for the numerical modelling of cracked masonry structures', *Automation in Construction*, 125, p. 103633. Available at: https://doi.org/10.1016/j.autcon.2021.103633.

Loverdos, D. *et al.* (2021b) 'An innovative image processing-based framework for the numerical modelling of cracked masonry structures', *Automation in Construction*, 125, p. 103633. Available at: https://doi.org/10.1016/j.autcon.2021.103633.

Loverdos, D. and Sarhosis, V. (2022a) 'Automatic image-based brick segmentation and crack detection of masonry walls using machine learning', *Automation in Construction*, 140, p. 104389. Available at: https://doi.org/10.1016/j.autcon.2022.104389.

Loverdos, D. and Sarhosis, V. (2022b) 'Automatic image-based brick segmentation and crack detection of masonry walls using machine learning', *Automation in Construction*, 140, p. 104389. Available at: https://doi.org/10.1016/j.autcon.2022.104389.

Loverdos, D. and Sarhosis, V. (2023a) 'Automation in Documentation of Ageing Masonry Infrastructure Through Image-Based Techniques and Machine Learning', in *European Workshop on Structural Health Monitoring*. Palermo, pp. 727–735. Available at: https://doi.org/10.1007/978-3-031-07322-9_73.

Loverdos, D. and Sarhosis, V. (2023b) 'Geometrical digital twins of masonry structures for documentation and structural assessment using machine learning', *Engineering Structures*, 275(PA), p. 115256. Available at: https://doi.org/10.1016/j.engstruct.2022.115256.

Loverdos, D. and Sarhosis, V. (2023c) 'Image2DEM: A geometrical digital twin generator for the detailed structural analysis of existing masonry infrastructure stock', *SoftwareX*, 22. Available at: https://doi.org/10.1016/j.softx.2023.101323.

Loverdos, D. and Sarhosis, V. (2023d) 'Pixel-level block classification and crack detection from 3D reconstruction models of masonry structures using Convolutional Neural Networks', *Engineering Structures* [Preprint].

Martin, D.R., Fowlkes, C.C. and Malik, J. (2003) 'Learning to detect natural image boundaries using brightness and texture', *Advances in Neural Information Processing Systems*, 26(5), pp. 530–549. Available at: https://doi.org/10.1109/TPAMI.2004.1273918.

Martin, D.R., Fowlkes, C.C. and Malik, J. (2004) 'Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues', *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENC*, 26(5), pp. 530–549.

McKibbins, L. *et al.* (2006) *Masonry arch bridges: condition appraisal and remedial treatment (C656)*. London: CIRIA. Available at: https://www.ciria.org/ItemDetail?iProductCode=C656&Category=BOOK&WebsiteKey=3f18c87a-d62b-4eca-8ef4-9b09309c1c91 (Accessed: 1 June 2021).

Morer, P., de Arteaga, I. and Ortueta, A. (2013) 'A low-cost photogrammetric methodology to obtain geometrical data of masonry arch bridges', *Journal of Architectural Conservation*, 19(3), pp. 246–264. Available at: https://doi.org/10.1080/13556207.2013.869974.

Muhit, B.I. *et al.* (2023) 'A Framework for Digital Twinning of Masonry Arch Bridges', in *Conference: Eighth International Symposium on Life-Cycle Civil Engineering (IALCCE2023)*. Milan, Italy.

Napolitano, R. and Glisic, B. (2019) 'Methodology for diagnosing crack patterns in masonry structures using photogrammetry and distinct element modeling', *Engineering Structures*, 181(15), pp. 519–528. Available at: https://doi.org/10.1016/j.engstruct.2018.12.036.

Nicastro, A. (2019) *fixed recovery of metric depth for orthographic cameras #40*. Available at: https://github.com/mmatl/pyrender/pull/40/files (Accessed: 3 February 2023).

Oses, N., Dornaika, F. and Moujahid, A. (2014) 'Image-based delineation and classification of built heritage masonry', *Remote Sensing*, 6(3), pp. 1863–1889. Available at: https://doi.org/10.3390/rs6031863.

Pepi, C. *et al.* (2021) 'An integrated approach for the numerical modeling of severely damaged historic structures: Application to a masonry bridge', *Advances in Engineering Software*, 151(November 2020), p. 102935. Available at: https://doi.org/10.1016/j.advengsoft.2020.102935.

Phares, B.M. *et al.* (2004) 'Routine Highway Bridge Inspection Condition Documentation Accuracy and Reliability', *Journal of Bridge Engineering*, 9(4), pp. 403–413. Available at: https://doi.org/10.1061/(asce)1084-0702(2004)9:4(403).

Ramer, U. (1972) 'An iterative procedure for the polygonal approximation of plane curves', *Computer Graphics and Image Processing*, 1(3), pp. 244–256. Available at: https://doi.org/10.1016/S0146-664X(72)80017-0.

Rolin, R. *et al.* (2019) 'From point cloud data to structural analysis through a geometrical hBIM-oriented model', *Journal on Computing and Cultural Heritage*, 12(2), pp. 1–26. Available at: https://doi.org/10.1145/3242901.

Ronneberger, O., Fischer, P. and Brox, T. (2015) 'U-Net: Convolutional Networks for Biomedical Image Segmentation', *arXiv*, pp. 1–8. Available at: https://doi.org/https://doi.org/10.48550/arXiv.1505.04597.

Sarhosis, V. *et al.* (2014) 'The effect of skew angle on the mechanical behaviour of masonry arches', *Mechanics Research Communications*, 61, pp. 53–59. Available at: https://doi.org/10.1016/j.mechrescom.2014.07.008.

Sarhosis, V., Forgács, T. and Lemos, J. V. (2019) 'A discrete approach for modelling backfill material in masonry arch bridges', *Computers and Structures*, 224, p. 106108. Available at: https://doi.org/10.1016/j.compstruc.2019.106108.

Sarhosis, V., Garrity, S.W. and Sheng, Y. (2015) 'Influence of brick-mortar interface on the mechanical behaviour of low bond strength masonry brickwork lintels', *Engineering Structures*, 88(1), pp. 1–11. Available at: https://doi.org/10.1016/j.engstruct.2014.12.014.

Sarhosis, V. and Lemos, J. V. (2018) 'A detailed micro-modelling approach for the structural analysis of masonry assemblages', *Computers and Structures*, 206, pp. 66–81. Available at: https://doi.org/10.1016/j.compstruc.2018.06.003.

Sarhosis, V. and Sheng, Y. (2014) 'Identification of material parameters for low bond strength masonry', *Engineering Structures*, 60, pp. 100–110. Available at: https://doi.org/10.1016/j.engstruct.2013.12.013.

Schmid, K. *et al.* (2012) *Lidar 101: An Introduction to Lidar Technology, Data, and Applications*. Available at: https://coast.noaa.gov/data/digitalcoast/pdf/lidar-101.pdf (Accessed: 10 April 2023).

Segura, J. *et al.* (2021) 'Experimental and numerical insights on the diagonal compression test for the shear characterisation of masonry', *Construction and Building Materials*, 287, p. 122964. Available at: https://doi.org/10.1016/j.conbuildmat.2021.122964.

Shen, Y. *et al.* (2018) 'Extracting individual bricks from a laser scan point cloud of an unorganized pile of bricks', *Remote Sensing*, 10(11). Available at: https://doi.org/10.3390/rs10111709.

Sithole, G. (2008) 'Detection of Bricks in a Masonry Wall', in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 567–572. Available at: https://www.isprs.org/proceedings/XXXVII/congress/5_pdf/99.pdf (Accessed: 27 January 2021).

Sowden, A.M. (1990) *The Maintenance of Brick and Stone Masonry Structures*. 1st edn. Edited by A.M. Sowden. London: CRC Press. Available at: https://doi.org/https://doi.org/10.1201/9781003062066.

Spencer, B.F., Hoskere, V. and Narazaki, Y. (2019) 'Advances in Computer Vision-Based Civil Infrastructure Inspection and Monitoring', *Engineering*, 5(2), pp. 199–222. Available at: https://doi.org/10.1016/j.eng.2018.11.030.

Stockdale, G., Yuan, Y. and Milani, G. (2022) 'The behavior mapping of masonry arches subjected to lumped deformations', *Construction and Building Materials*, 319. Available at: https://doi.org/10.1016/j.conbuildmat.2021.126069.

Suzuki, S. and Abe, Keiichi. (1985) 'Topological structural analysis of digitized binary images by border following', *Computer Vision, Graphics and Image Processing*, 30(1), pp. 32–46. Available at: https://doi.org/10.1016/0734-189X(85)90016-7.

Tiberti, S. *et al.* (2020) 'A Genetic Algorithm adaptive homogeneous approach for evaluating settlement-induced cracks in masonry walls', *Engineering Structures*, 221(April), p. 111073. Available at: https://doi.org/10.1016/j.engstruct.2020.111073.

Tiberti, S. and Milani, G. (2019) '2D pixel homogenized limit analysis of non-periodic masonry walls', *Computers and Structures*, 219, pp. 16–57. Available at: https://doi.org/10.1016/j.compstruc.2019.04.002.

Tiberti, S. and Milani, G. (2020a) '3D homogenized limit analysis of non-periodic multi-leaf masonry walls', *Computers and Structures*, 234, p. 106253. Available at: https://doi.org/10.1016/j.compstruc.2020.106253.

Tiberti, S. and Milani, G. (2020b) '3D voxel homogenized limit analysis of single-leaf non-periodic masonry', *Computers and Structures*, 229, p. 106186. Available at: https://doi.org/10.1016/j.compstruc.2019.106186.

UNESCO Institute for Statistics (2021) *UN Sustainable Development Goal 11.4: Strengthen efforts to protect and safeguard the world's cultural and natural heritage - SDG Indicator Metadata*. Montreal. Available at: https://unstats.un.org/sdgs/metadata/files/Metadata-11-04-01.pdf (Accessed: 2 February 2022).

Valero, E. *et al.* (2019) 'Automated defect detection and classification in ashlar masonry walls using machine learning', *Automation in Construction*, 106(May), p. 102846. Available at: https://doi.org/10.1016/j.autcon.2019.102846.

Valero, E., Bosché, F. and Forster, A. (2018) 'Automatic segmentation of 3D point clouds of rubble masonry walls, and its application to building surveying, repair and maintenance', *Automation in*

Construction, 96(October 2017), pp. 29–39. Available at: https://doi.org/10.1016/j.autcon.2018.08.018.

Vandenabeele, L. *et al.* (2023) 'Deep Learning for the Segmentation of Large- Scale Surveys of Historic Masonry : A New Tool for Building Archaeology Applied at the Basilica of St Anthony in Padua Deep Learning for the Segmentation of Large-Scale Surveys of Historic Masonry : A New Tool ', *International Journal of Architectural Heritage*, 00(00), pp. 1–13. Available at: https://doi.org/10.1080/15583058.2023.2260771.

Vincent, L. and Soille, P. (1991) 'Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6), pp. 583–598. Available at: https://doi.org/10.1109/34.87344.

Volk, R., Stengel, J. and Schultmann, F. (2014) 'Building Information Modeling (BIM) for existing buildings - Literature review and future needs', *Automation in Construction*, pp. 109–127. Available at: https://doi.org/10.1016/j.autcon.2013.10.023.

Wang, N. *et al.* (2019) 'Automatic damage detection of historic masonry buildings based on mobile deep learning', *Automation in Construction*, 103(February), pp. 53–66. Available at: https://doi.org/10.1016/j.autcon.2019.03.003.

Zhang, Y., Macorini, L. and Izzuddin, B.A. (2018) 'Numerical investigation of arches in brick-masonry bridges', *Structure and Infrastructure Engineering*, 14(1), pp. 14–32. Available at: https://doi.org/10.1080/15732479.2017.1324883.