COGITO

CONSTRUCTION PHASE DIGITAL TWIN MODEL

cogito-project.eu

D5.1 – Innovative Scan-vs-BIM-based Geometric QC component v1

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310.

D5.1 – Innovative Scan-vs-BIM- based Geometric QC component v1

Dissemination Level:	Public
Deliverable Type:	Demonstrator
Lead Partner:	UEDIN
Contributing Partners:	UEDIN, UCL, CERTH, RSRG
Due date:	31/01/2022
Actual submission date:	31/01/2022

Authors

Name	Beneficiary	Email
Martín Bueno	UEDIN	martin.bueno@ed.ac.uk
Frédéric Bosché	UEDIN	f.bosche@ed.ac.uk
Kyriakos Katsigarakis	UCL	k.katsigarakis@ucl.ac.uk
Georgios Lilis	UCL	g.lilis@ucl.ac.uk
Thanos Tsakiris	CERTH	atsakir@iti.gr
Elias Bruno Meusburger	RSRG	Elias.Meusburger@rsrg.com

Reviewers

Name	Beneficiary	Email
Jochen Teizer	AU	teizer@cae.au.dk
Raúl García-Castro	UPM	rgarcia@fi.upm.es

Version History

Version	Editors	Date	Comment
0.1	Martín Bueno, Frédéric Bosché	17.11.2021	ToC
0.2	Martín Bueno, Frédéric Bosché	15.12.2021	First draft ready for internal review
0.5	Jochen Teizer, Raúl García-Castro	20.01.2022	First draft internal review
0.7	Martín Bueno, Frédéric Bosché	24.01.2022	Second version of document
0.8	Martín Bueno	26.01.2022	Cleaning of document
0.9	Martín Bueno, Giorgos Giannakis	27.01.2022	Quality Check
1.0	Martín Bueno, Giorgos Giannakis	31.01.2022	Submission to the EC

Disclaimer

Executive Summary

The COGITO Deliverable D5.1 “Innovative Scan-vs-BIM-based Geometric QC component v1” documents the COGITO Scan-vs-BIM solution and presents the first iteration of the development activities in T5.1 “Scan-vs-BIM Geometric Quality Control”. Overall, the Scan-vs-BIM solution aims to obtain the matching between the as-built data (laser scanned point clouds) and as-designed data (object in the project BIM model). The output matched data is stored to be used as input for the subsequent processes of the GeometricQC tool, developed in T5.3 “BIM-based Standard Test Methods for Geometric Quality Control”. As such, the Scan-vs-BIM solution lies at the start of the overall COGITO geometric quality control process.

The Scan-vs-BIM solution, designed to be part of the GeometricQC tool, is a set of modular open-source tools and components that can be included in any project. It is delivered as a service that can be executed from the command line, included in any operating system. The Scan-vs-BIM solution is composed of two sub-components, the main one being the *Point Cloud Matching and Segmentation*, and the second one the *BIM Element Manager*. The *Point Cloud Matching and Segmentation* sub-component aims to perform the point cloud matching process of the laser scanned geometric data acquired on the construction site (which represents the actual geometry of the constructed building/asset components) and the as designed geometry data. The *BIM Element Manager* sub-component aims to be the interpreter and manager of the BIM model components (such as walls, slabs, columns, beams, etc.) stored in the IFC files to facilitate the subsequent data processing and GeometricQC tool execution.

The present documentation of the COGITO Scan-vs-BIM solution, along with its sub-components, is oriented towards the functionalities they broadly deliver, the technology stacks they build upon, the inputs, outputs and APIs they expose, the installation instructions, the assumptions and restrictions, the applications examples, the development and integration status, and the requirements coverage. In this first release, the COGITO Scan-vs-BIM solution implements a set of the envisaged functionalities (i.e. IFC file interpreter, point cloud to mesh matching) and its usage is illustrated and evaluated with examples obtained during the development of the sub-components using artificial and real case examples.

Table of contents

List of Figures

List of Tables

List of Acronyms

Term	Description
API	Application Programming Interface
BIM	Building Information Modelling
B-Rep	Boundary Representation
CAD	Computer-Aided Design
CAE	Computer-Aided Engineering
CAM	Computer-Aided Manufacturing
COGITO	Construction Phase diGItal Twin mOdel
IFC	Industry Foundation Classes
OBJ	Object file
QC	Quality control
STL	Standard Triangle/Tessellation Language
TLS	Terrestrial Laser Scanning
UID	Unique Identifier

1 Introduction

1.1 Scope and Objectives of the Deliverable

This deliverable reports on the work conducted between M6 and M13 on the Scan-vs-BIM solution being developed as part of T5.1. Scan-vs-BIM. It can be seen as a main step in the geometric Quality Control (QC) process proposed by the COGITO consortium. The scope of the Scan-vs-BIM solution is to enable the GeometricQC tool to perform automatic QC compliance, and report its results back to the relevant stakeholders, such as the quality managers and the project managers. Having the QC reports at hand, these stakeholders can decide on whether the already built components require any remedial works or they satisfy the standard specifications.

More specifically, this deliverable reports on the development of the first release of the Scan-vs-BIM solution, focusing on its two main sub-components listed below:

- **BIM Element Manager sub-component**: the BIM model data pre-processing, interpreter and manager tool. This sub-component takes as input the as-designed data (using the IFC schema file format) and pre-processes the geometric information of the relevant elements (such as walls, columns, slabs, beams) to be later stored in an internal structure, facilitating the subsequent data queries and management for the overall QC process. The tool employs automated processes only.
- **Point Cloud Matching and Segmentation sub-component**: the as-built data matching and segmentation tool. This sub-component takes as input the acquired and registered laser scanned data (point cloud) and the geometries from the as-designed data (triangle mesh). The latter data is obtained from the COGITO Digital Twin Platform with the help of the BIM Element Manager sub-component. The tool employs automated processes only.

These two sub-components provide the core functionalities of the Scan-vs-BIM solution that is integrated with the GeometricQC tool.

1.2 Relation to other Tasks and Deliverables

This deliverable reports the first version (v1) of the Scan-vs-BIM solution. A second version (v2) of the Scan-vs-BIM solution will be reported at the end of M24 in *"D5.2: Innovative Scan-vs-BIM- based Geometric QC component v2"* including new functionalities and improvements to the components presented in this deliverable.

1.2.1 Relation to other COGITO tools and components

The Scan-vs-BIM solution serves as a data provider to the GeometricQC tool. In alignment with COGITO's system architecture, it does not interact with other components outside the GeometricQC tool.

1.2.2 Relation to other Deliverables

The components presented in this deliverable are part of the GeometricQC tool which is being prototyped based on the User Requirements defined in "*D2.1: Stakeholder requirements for the COGITO system"*, the functional and non-functional requirements introduced in "*D2.4: COGITO system architecture v1"* and will be presented in *"D5.5: BIM-based Standard Test Methods for Geometric Quality Control v1"*.

The Scan-vs-BIM solution is principally used to support the Use Case 2.1 (UC2.1).

1.3 Structure of the Deliverable

The deliverable is organised as follows: Section 2 presents an overview of the Scan-vs-BIM solution, placing the sub-components in the overall context of the solution and the GeometricQC tool processing pipeline. Sections 3 and 4 describe the *BIM Element Manager* and *Point Cloud Matching and Segmentation* sub-components, including the technology stack, implementation tools, the input/output data, and API documentation, application examples, licensing, installation instructions, development and integration

status, requirements coverage, and assumption and restrictions of each sub-component. Section 5 concludes the deliverable.

2 Overview of the Scan-vs-BIM solution

The aim of the proposed Scan-vs-BIM solution is to automatically match the as-built data acquired during the project's construction phase against the as-designed data. The Scan-vs-BIM solution is designed to use open formats (i.e. OBJ, STL, PLY, E57, etc), thus increasing its interoperability, accessibility, and data sharing among third parties. Concerning the as-designed data, the input information is extracted from the as-designed BIM model of the construction project in an IFC format, where information is stored about the project's geometry and additional data (e.g. components materials) that are relevant for the project and the Scan-vs-BIM solution. The as-built data is point cloud data obtained using Terrestrial Laser Scanning (TLS) devices, already registered (by the surveyor) in the project global coordinate system. Various point cloud open formats are supported by the Scan-vs-BIM solution, such as E57, PLY, or PTS.

The Scan-vs-BIM solution is part of the GeometricQC tool, defined in "*D2.1: Stakeholder requirements for the COGITO system"* and "*D2.4: COGITO system architecture v1"*, and as such, it was designed to be a back-end set of classes, part of the GeometricQC tool pipeline, as can be seen in Figure 1, which will be explained in detail in *"D5.5: BIM-based Standard Test Methods for Geometric Quality Control v1"*. The Scan-vs-BIM solution consist of two (2) main sub-components:

- **BIM Element Manager sub-component**: this sub-component takes the as-designed data in IFC format and generates a structure containing all the elements which require a geometry quality control and all the information that will be needed for the GeometricQC tool execution. The different information stored in this structure contains the original as-designed geometry, location, unique identifiers (UID), and the associated as-built segmented point cloud after matching is completed.
- **Point Cloud Matching and Segmentation sub-component**: this sub-component takes the scanned as-built data in its entirety and matches all the points to their corresponding BIM element as extracted by the BIM Element Manager.

As explained above, the Scan-vs-BIM solution is part of an overall GeometricQC tool, hence some of its sub-components were designed to be utilised by other sub-components of the GeometricQC tool. A high-level diagram of the GeometricQC tool is illustrated in Figure 1, highlighting (green dashed lines) the positioning of the Scan-vs-BIM sub-components in the whole tool's pipeline.

The above two sub-components are presented in Sections 3, and 4, respectively.

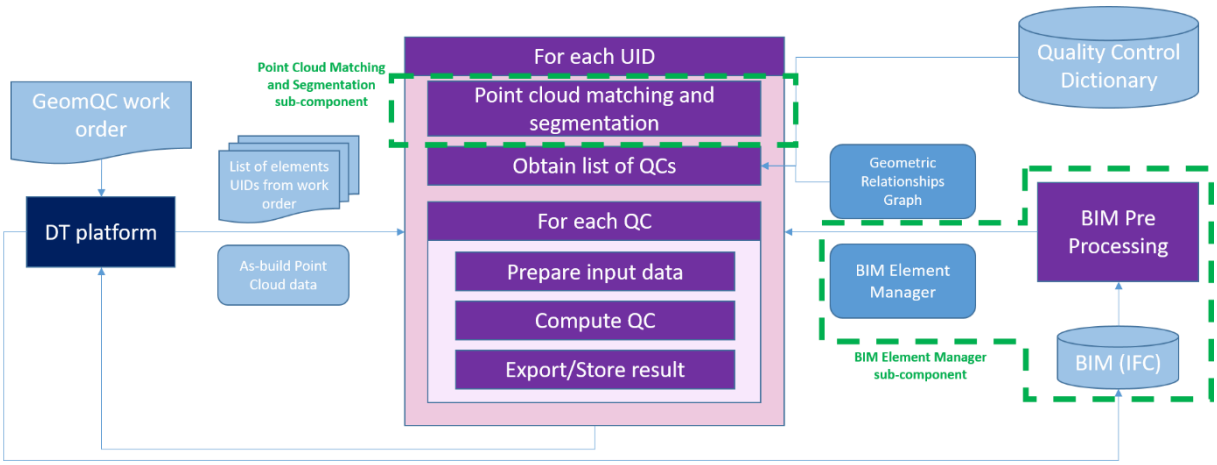


Figure 1: GeometricQC tool pipeline overview

3 BIM Element Manager sub-component

In this section, we describe the BIM Element Manager sub-component. This component is key for the GeometricQC tool as it is the component to pre-process the IFC data, and to store the processed data in the desired format for the tool, avoiding multiple queries and re-processing of the BIM information (IFC). Hence, the BIM Element Manager mainly serves as the BIM data management layer of the GeometricQC tool providing methods to obtain and store BIM data in order to be used by the rest of the GeometricQC tool. The following sub-sections describe in more detail the BIM Element Manager sub-component and its technical aspects and requirements.

3.1 Prototype Overview

The objective of the BIM Element Manager sub-component is to allow the GeometricQC tool to load, interpret, and manage the construction's project as-designed data. For its development, the BIM Element Manager utilises the IFC schema to process the as-designed BIM data. Despite being just a component of the entire GeometricQC tool's sequential process, it is delivered as a key module, obtaining the relevant geometric information, as well as its identification labels, and storing it so that the GeometricQC tool can be executed without requiring re-processing of the same information at every call.

The BIM Element Manager sub-component is quite simple to use and include in any project. It only requires the file path of the IFC file as input parameter, or the BIM Manager file path. Note here that, as will be explained in more detail in D5.5, the GeometricQC tool requires several inputs, but this sub-component requires only the IFC file path.

As can be seen in Figure 2, the BIM Element Manager sub-component delivers five object classes that are interconnected:

- **BIMElement class**: this class is used to describe a BIMElement object populated with all the relevant information extracted from the IFC file. It records all the relevant BIM identifiers, such as the UID, the IFC entity ID, and the human interpretable label. It also contains the element type and the material type of the component, as well as the mesh and segmented point cloud file paths. Additionally, it contains geometric information extracted from the IFC file, such as the global space location, and the global space vertices coordinates from the B-rep. The class is also prepared with the gets and sets methods to modify and query information relative to the BIM component. This class can be easily extended (as it will likely be) to contain more information that is required to conduct the whole geometric QC process.
- **BIMElementManager class**: main class of the sub-component to access the entire geometric information stored in the BIM model. In a nutshell, this class is the heart of the GeometricQC tool, storing all the relevant geometric information. The object contains the information of the IFC file path where the data comes from, and a mapped list of all the BIM components of interest, indexed by the UID of them, to facilitate the access, search, and queries by the GeometricQC tool and its sub-components. Additionally, it contains the necessary methods to add elements, either a single one or a set of set, save the information into a file and load it, and to obtain and process the queried data.
- **ElementType class**: this class only contains a list of possible BIM components types (such as walls, columns, slabs, etc) that are of interest for the GeometricQC tool. It also contains the parser methods to be able to read and save that information into the internal project files and data models. This class is likely to be extended to include more element types as the development of the tools progresses.
- **MaterialType class**: this class, similarly to the previous one, contains the material types of the BIM components. Since the work in WP5 is focused on the standard specifications of concrete [1] and steel [2], this class only contains these main materials. However, it could naturally also be extended if needed.
- **IFCManager class**: this class is an IfcOpenShell library wrapper, created to ease the information retrieval from the IFC file and its complex structure. The only required parameter to use this object is to set the IFC file path at object declaration. The class contains the necessary methods to retrieve

the BIM components and its geometric information from the IFC file. The output of the *getElement* and *getElemensList* methods serve as input for the **BIMElementManager** class. This class can also be extended (as it is likely to) and include additional functionalities that would be useful for the GeometricQC tool.

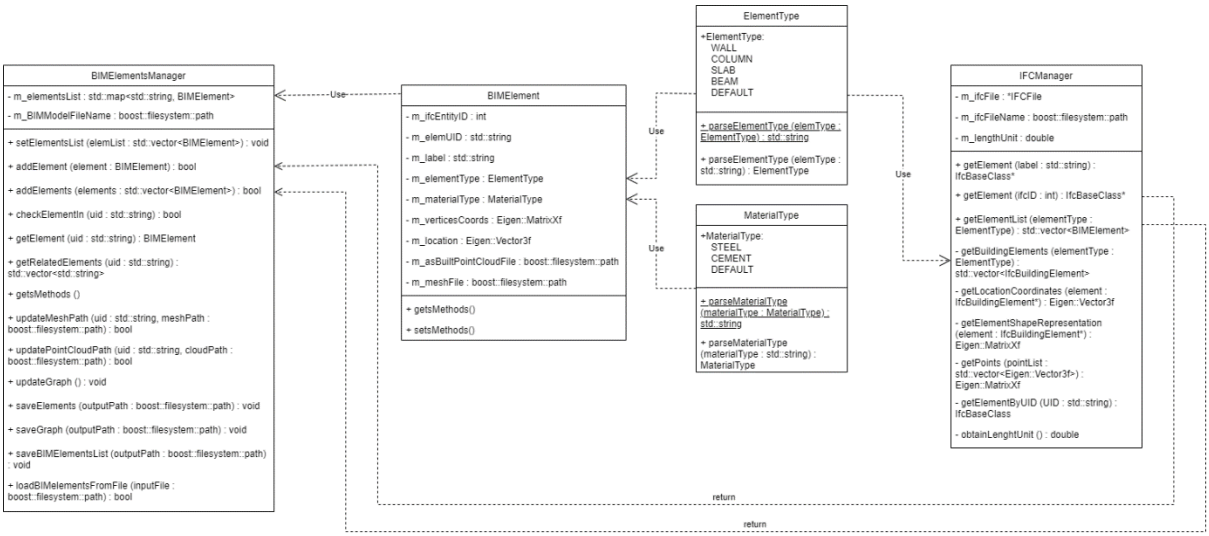


Figure 2: BIM Element Manager sub-component classes and its relationships

3.2 Technology Stack and Implementation Tools

The BIM Element Manager sub-component is a set of headers only libraries that can be included in any project as any other libraries. The classes are written in C++ v14 for compatibility purposes. In order to be able to facilitate the development of the sub-component, the BIM Element Manager sub-component, and in its extension the GeometricQC tool, use the following open source C++ libraries:

- **Eigen**: high-level C++ library of template headers for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms. It is used to handle the matrices and vectors of point coordinates, and as an algebra toolbox.
- **Boost**: set of libraries for the C++ programming language that provides support for tasks and structures such as multithreading, regular expressions, unit testing, etc., and improves the capabilities of the C++ language. Boost contains over 150 individual libraries. Boost is used in a number of places within the GeometricQC tool, mainly to deal with JSON formats, system files and directories management and speed up processes.
- **IfcOpenShell**: open-source library designed to enable software developers to work with the IFC file format. It is the main library to read and pre-process BIM IFC files, and extract all the relevant information of each component.
- **OpenCascade**: open-source library designed to help software developers to work with 3D CAD, CAM, CAE, B-Rep, etc. It is used by IfcOpenShell to extract and work with the geometry of each BIM component from the IFC file.

Table 1: Libraries and Technologies used in the BIM Element Manager sub-component

Library/Technology Name	Version	License
Eigen	3.3.9	MPL2
Boost	1.76.0	Boost Software License 1.0
IfcOpenShell	0.6.0	LGPL-3.0 License
OpenCascade	7.5.0	LGPL-2.1 License

3.3 Input, Output and API Documentation

The BIM Element Manager sub-component can be demonstrated following 2 different options: (1) as a standalone object class that can be included in any project, or (2) as part of the GeometricQC tool.

The first option requires the following input parameters:

- **IFC file path**: the BIM file path, in IFC file format, of the project at object creation. The class will be able to load the IFC file and extract all the BIM components from it and store them in the manager.
- **BIM Element Manager file path**: As a second option, the object can be created using a pre stored BIM Element Manager object. In this case, the class requires the file path to the JSON file of the BIM Element Manager file. This option is useful to avoid the pre-processing step every time the GeometricQC tool is run as part of the COGITO solution.

For the second option, the GeometricQC tool would require as an input the following parameters:

- **IFC file path**: BIM file in IFC file format of the project. The GeometricQC tool will create a BIM Element Manager object using the IFC file path as input parameter as in the standalone case.
- **File with the list of elements to be quality-controlled**: text file containing the UIDs of the elements that need to be checked at that point in time. In this case, the BIM Element Manager sub-component does not require to use this input, but the input data is linked to the one that is stored in the sub-component.
- **As-built point cloud data file path**: file path to the input point cloud of the as-built data that needs to be analysed.
- **Output folder path**: folder path where to store all the outputs that are generated by the tool. In the case of the BIM Element Manager sub-component is the output path where to store the pre-processing data, such as the BIM Element Manager file, and also where to store the point clouds and mesh files that are part of each BIM Element.

In the case that the BIM Element Manager sub-component is used as part of the GeometricQC tool, the entire solution can run from the command prompt. In its second release, to be consistent with the rest of the COGITO solution, the GeometricQC tool will expose an execution interaction with the Digital Twin Platform.

3.4 Application Example

The BIM Element Manager sub-component is essentially used to extract the geometric information from the BIM file. For such data extraction, it only requires as input the IFC file path to function properly. In this section, we will show the outputs generated using two different example IFC files.

3.4.1 COGITO Basic Artificial Test BIM model

The COGITO Basic Artificial Test BIM model is an IFC file, representing a very basic test building with all the basic components that are required in this first stage. This BIM model has been used for the development of all the classes and algorithms integrated in the first release of this component, allowing to progress with the GeometricQC tool without the need of acquiring real construction projects' datasets. Figure 3 shows the geometry of this BIM model.

Once the input file has been set, the BIM Element Manager uses the **IFCManager** class to query the elements from the IFC file using the appropriate *get()* method, and adds them into the mapped structure for storage using the appropriate *add()* method. Figure 4 depicts the logs of the BIM Element Manager after loading the IFC data. We can see that, in this case, there is a set of elements loaded by type, and that the BIM Element Manager JSON file has been saved for its reuse in future runs in the same project. Figure 5 illustrates the content of this JSON file, where we can see the list of components in the file, the file of origin, and the list of them, with all the fields from the **BIMElement** class.

In subsequent calls to the GeometricQC tool under the same construction project, the BIM Element Manager sub-component will load the BIM Element Manager JSON file instead of having to read and pre-process all

the information from the IFC file, saving time, and resources. Figure 6 presents the output after loading the BIM Elements Manager JSON file, showing the correct list of elements loaded from it.

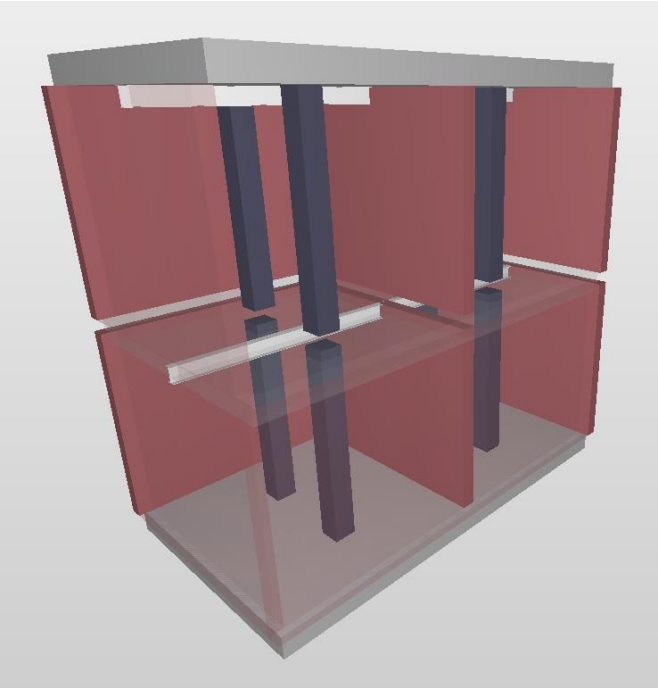


Figure 3: COGITO Basic Artificial Test BIM model. Front walls and intermediate slab have been removed for visualisation purposes

```
**************************************************
COGITO_GeomQC tool application v0.0.1
**************************************************

Loading information from IFC file C:\COGITO_Repos\COGITO_GeomQC\COGITO-DUMMY.ifc
Loaded 10 walls from the IFC file
Loaded 3 slabs from the IFC file
Loaded 8 columns from the IFC file
Loaded 4 beams from the IFC file
Saving BIM elements list info at C:\COGITO_Repos\COGITO_GeomQC\build/Output\BIMElementsList.json
```

Figure 4: COGITO Basic Artificial Test file loaded by the BIM Element Manager sub-component

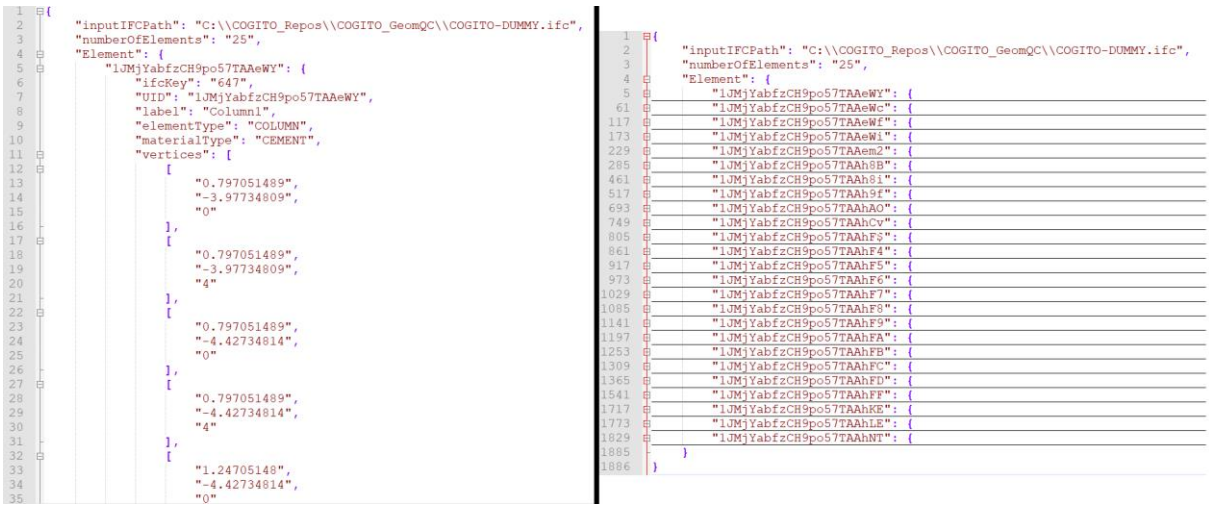


Figure 5: BIM Element Manager JSON file example after saving using the COGITO Basic Artificial Test file

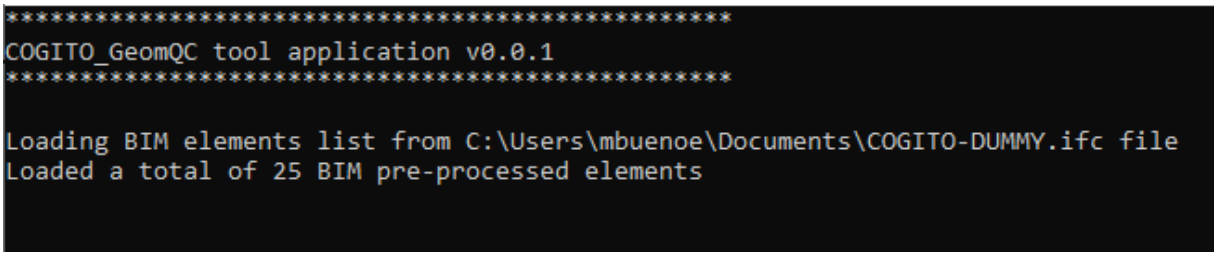


Figure 6: BIM Elements Manager JSON file of COGITO Basic Artificial Test file loaded by the BIM Element Manager sub-component

3.5 Licensing

The Scan-vs-BIM solution is free software; you can redistribute it and/or modify it under the terms of the **GNU General Public License Version 3** as published by the Free Software Foundation (https://www.gnu.org/licenses/gpl-3.0.en.html).

3.6 Installation Instructions

No complex installation is required. Binaries will be provided in the Cyberbuild Datashare collections. BIM Element Manager sub-component is a set of header-only libraries that can be included in any project just by adding the header files to the includes list.

3.7 Development and integration status

The BIM Element Manager sub-component is currently under development. It can be considered to be 70% completed, having the main basic functionalities already in place and ready to use. It is planned that additional functionalities would be introduced by time of publication of D5.5, integrating the BIM Element Manager with the rest of the GeometricQC tool, allowing new interactions with the quality control component, such as all the interactions relative to the geometric relationships graph and the QC compliances that are required for each BIM component.

The BIM Element Manager does not interact with any external component, all the interactions are within the GeometricQC tool.

3.8 Requirements Coverage

Despite the fact that the BIM Element Manager sub-component is just a back-end part of the GeometricQC tool, it already covers a couple of the requirements defined in D2.4 and D2.1.

The functional and non-functional requirements that have introduced in the COGITO System Architecture and that have already been covered by this version of the BIM Element manager are presented in Table 2. Functional requirement Req-1.1 is partially covered using this sub-component, thanks to the IFC file loader and interpreter, however, despite storing the matched point cloud data file path into each BIM component, the BIM Element Manager does not have the capability to load the point cloud data. That part is covered in Section 4. Thanks to the C++ property of using different objects in header only classes, Req-2.1, Req-2.2, and Req-2.3 are well covered within this sub-component. As previously mentioned, this set of classes (as part of the full GeometricQC tool library) can be used autonomously in any other project, or as part of the full GeometricQC tool. In addition, its scalability is well handled using object oriented programming, which enables straightforward addition of new functionalities and properties to each of the components.

Table 2: BIM Element Manager sub-component requirements coverage from D2.4

ID	Description	Type	Status
Req-1.1	Loading as planned 4D BIM and point cloud data	Functional	Partially achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Reusability	Non-Functional	Achieved

Req-2.3	Interoperability	Non-Functional	Achieved

Table 3 presents the stakeholders requirements that have been documented in D2.1 and are relevant to this component. COGI-CS-1 and COGI-CS-4 are covered simply by the programming language that has been selected for the development of the GeometricQC tool. With regards to COGI-QC-11, the BIM Element Manager allows to load and interpret the as-designed data from the BIM model, thus enabling the interconnection with other sub-components to automate the forward coming quality control activities.

Table 3: BIM Element Manager sub-component stakeholders requirements coverage from D2.1

ID	Description	Type	Priority	Status
COGI-CS-1	Runs on desktop or laptop PC	• Operational	Must	Achieved
COGI-CS-4	Runs on Windows	• Operational	Must	Achieved
COGI-CS-5	Runs on Mac	• Operational	Could	Achieved
COGI-QC-11	Automates QC-related activities	• Functional • Operational	Must	Achieved

3.9 Assumptions and Restrictions

The first version of the BIM Element Manager sub-component has been delivered under certain assumptions and restrictions, listed below:

- It has been developed to be part of the GeometricQC tool, so no standalone programme is supposed to be executed for this sub-component.
- It currently supports IFC files of version 4.0.
- It requires the input IFC file to be correct (free of geometric errors) and consistent with the IFC 4.0 schema.
- BIM elements are expected to have the vertical along the +Z axis using the right-hand side coordinate frame.
- The elements' reference coordinate frame within the entire GeometricQC tool uses global coordinates; hence the generated output files contain coordinates information in the same global space.
- The measures are stored using the international metric system, and the distance measurement unit used in the sub-component is the metre [m].

4 Point Cloud Matching and Segmentation sub-component

In this section, we describe the *Point Cloud Matching and Segmentation* sub-component. This sub-component is the key part of the Scan-vs-BIM solution, responsible for performing the matching of the as-built point cloud data to the as-designed BIM components and storing the result in the *BIM Element Manager* sub-component for further processing within the *GeometricQC* tool. The following sub-sections describe in more detail the *Point Cloud Matching and Segmentation* sub-component and its technical aspects and requisites.

4.1 Prototype Overview

The objective of the *Point Cloud Matching and Segmentation* sub-component is to geometrically match the as-built point cloud data to the as-designed BIM components. The as-built data comes (as the result of the survey working order) in the form of a point cloud already registered in the project coordinate system (i.e. the same coordinate system as the as-designed BIM model), while the as-designed data is provided by reinterpreting the geometry of each component in the as-designed BIM model into a triangular mesh. The Scan-vs-BIM component (like the rest of the GeometricQC tool) utilises well know open formats, PLY and E57 for the point clouds, and OBJ or STL for the meshes.

The Scan-vs-BIM solution is a key component of the GeometricQC tool. Indeed, being able to accurately segment the as-built data of each BIM component is critical to obtain reliable and accurate geometric control results.

As with the majority of the sub-components that are developed in the GeometricQC tool, the *Point Cloud Matching and Segmentation* sub-component is a modular component, quite simple to use and include in any other project. As a standalone object class, it only requires the file path of the input point cloud and a list of file paths to the BIM component meshes. Note that, as will be explained in more detail in D5.5, the GeometricQC tool requires several inputs, but this sub-component requires only the above-listed file paths.

As illustrated in Figure 7, the *Point Cloud Matching and Segmentation* sub-component comprises six object classes that are interconnected:

- **PointCloudMatching**: the main class of the Scan-vs-BIM solution. This class allows to pass all the input parameters necessary to perform the matching and segmentation of the point cloud to the meshes. It only contains three different fields, the file path to the input point cloud, the list of file paths to the BIM component meshes, and the output path where to export the matched and segmented point clouds. The matching process has an input indicating which one of the point matching algorithms to use. Its generated output is then passed to the main GeomericQC tool for further processing.
- **Open3DMatching**: this class is in charge of the heavy computational work, calling some of the other classes and methods to obtain and process the relevant data. As its name indicates, the point cloud matching is based on point cloud and triangle mesh structures of Open3D (see Section 4.2). All the relevant fields of this class are filled automatically when one of the matching methods is called.
- **VoxelGrid**: this class is in charge of processing the point cloud into a voxel grid. Everything that is required from the voxel grid can be processed and queried using this class. The input point cloud and the voxel size are the only parameter required as input to this class. As most of the other classes, it can be used independently of the project and be included in any other C++ project.
- **PointCloudUtils**: this class is designed as a modular point cloud utilities toolbox. Inside, we can find all the point cloud processing utilities used by the GeometricQC tool. By doing so, we can guarantee that this toolbox can be used by any other project that requires to work with point clouds.
- **PointCloudSegmentation**: this class contains all the point cloud segmentation methods, complementing the PointCloudUtils toolbox class.
- **MathUtils**: this class is another handy toolbox containing arithmetic operations and checks using mostly (but not only) vectors and matrices.

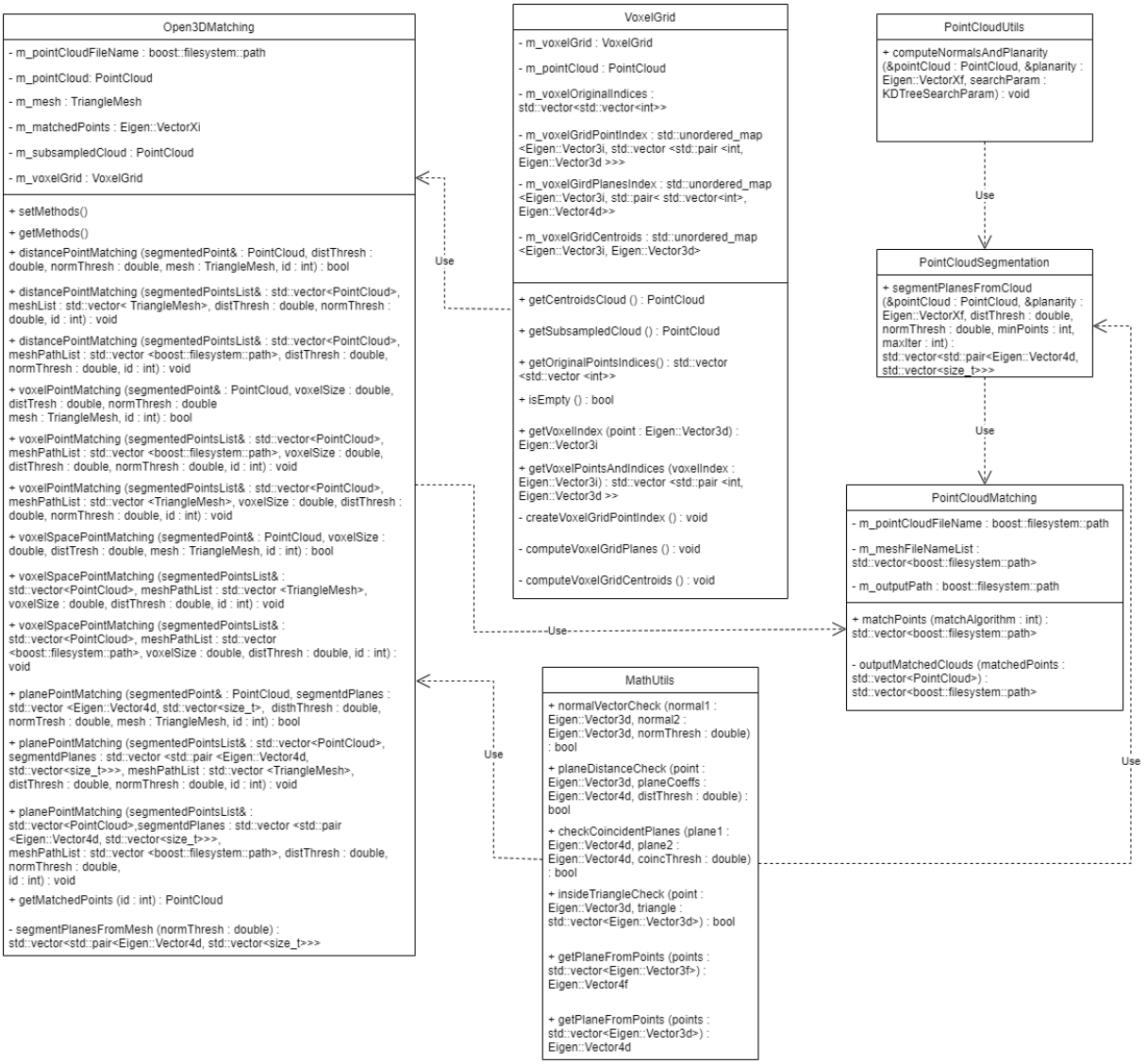


Figure 7: Point Cloud Matching and Segmentation sub-component classes and its relationships

4.1.1 Point cloud matching methods

The Point Cloud Matching and Segmentation sub-component contains four different matching methods implemented for its first release. As will be detailed in Section 4.4, these methods already pave a good foundation of results that contain the right quality for the geometric quality control process. These methods are briefly described below.

- **Point matching by distance and normal vector similarity**: this is the most basic, "brute force" method. It basically iterates through every unmatched point in the point cloud and check if (1) its normal vector (pre-computed if not provided) is similar to a given mesh's triangle normal vector, (2) its orthogonal distance to the plane of the triangle is within a given threshold, and (3) its orthogonal projection on the plane of the triangle falls within the triangle. If all these checks are satisfactory, the point is assigned a label as matched to that mesh (that triangle), and the algorithm continues with the rest of the cloud points. This method is referred to as the "brute force" method, because it iterates through all the points in the point cloud and triangles in the mesh. Consequently, it is not the most resource efficient.
- **Point matching by distance and normal vector similarity using a voxel subsample**: this method is very similar to the previous one, but instead of iterating through the entire point cloud, the point cloud is first subsampled using a voxel grid, reducing the number of points to be initially

checked. The iteration is performed using the centroid of each voxel first, and if the three checks define in the "brute-force" method are satisfactory, then algorithm recovers all the points from the original point cloud that lie inside that matched voxel and checks them one by one. This method is much faster than the previous ones, as it reduces the number of iterations and checks, with only minor reduction in the quality of the matching expected.

- **Point matching using a voxel space**: this method relies on checking if the mesh triangles lie inside an occupied voxel from the voxel space and perform the same checks for the points inside that occupied voxel. As we will see later, this method produces high quality results and it is fast, sometimes faster than the previous method.
- **Plane point matching**: this method's original idea is to segment the point cloud into planar surfaces and then match those planar surfaces at once. By doing so, the amount of data to be checked is significantly reduced. The algorithm segments planes using the RANSAC algorithm [3]. Once all the planes are segmented, and the original point indices mapped to them, the method can try to find a matching plane in the mesh, by checking whether the mesh's plane equation is coincident, within a threshold, with the segmented plane equation. If a coincident plane is found, the algorithm still must check if the points that belong to that plane also are inside the boundaries of the triangles defining the mesh plane, to reduce false positives matches. As we will see, this turns out to be the slowest of the four methods implemented so far, since it requires computing the normal vectors and segmenting the different planes from the point cloud, prior to start the point matching, but the matching step of the algorithm is actually the fastest. The results so far show this method is the less reliable, since it is very sensitive to incorrect point cloud plane segmentation. Its effective application requires the input point cloud to contain a low amount of noise.

4.2 Technology Stack and Implementation Tools

The Point Cloud Matching and Segmentation sub-component is a set of headers-only libraries that can be included in any project as any other libraries. The classes are written in C++ v14 for compatibility purposes. In order to be able to facilitate the development of the sub-component, the Point Cloud Matching and Segmentation sub-component, and in its extension the Scan-vs-BIM solution and the GeometricQC tool, use the following open source C++ libraries:

- **Eigen**: high-level open-source C++ library of template headers for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms. It is used to handle the matrices and vectors of point coordinates, and as an algebra toolbox.
- **Boost**: set of libraries for the C++ programming language that provides support for tasks and structures such as multithreading, regular expressions, unit testing, etc., and improves the capabilities of the C++ language. Boost contains over 150 individual libraries. Boost is used in many places within the GeometricQC tool, mainly to deal with JSON formats, system files and directories management and speed up processes.
- **Open3D**: open-source library designed to help the software developers and users to work with three-dimensional data, such as point clouds and meshes. It is the main library to read and process the point cloud and mesh data.

Table 4: Libraries and Technologies used in the Point Cloud Matching and Segmentation sub-component

Library/ Technology Name	Version	License
Eigen	3.3.9	MPL2
Boost	1.76.0	Boost Software License 1.0
Open3D	0.13	MPL2

4.3 Input, Output and API Documentation

Similar to the BIM Element manager, the Point Cloud Matching and Segmentation sub-component provides two option for demonstration: (1) running as a standalone object class that can be included in any project, or (2) running as an object part of the GeometricQC tool.

The first option requires the following input parameters:

- **pointCloudFileName**: file path to the input point cloud of the as-built data that is to be matched to the input object meshes.
- **meshFileNameList**: list of files path of the mesh geometries to which the input point cloud is to be matched. The list is using the C++ standard library format. The mesh can be of OBJ or STL type. In the future, it will be extended to accept a list of mesh files already loaded in memory, but for the current use, and within the GeometricQC tool, the mesh file path is stored and handled by the BIM Element Manager, so there is no need to keep elements in memory.
- **outputPath**: (optional) output path where to store the segmented point clouds. Each point cloud would be named using the same UID as the input BIM element UID.

For the second option, the GeometricQC tool would require as an input the following parameters:

- **As-built point cloud data file name**: file path to the input point cloud of the as-built data that is to be matched to the input object meshes.
- **IFC file path**: BIM file in IFC file format of the project. This file is not used by the Point Cloud Matching and Segmentation sub-component explicitly, but it contains information that is necessary for the rest of the GeometricQC tool to work.
- **File with the list of elements to be checked**: text file containing the UIDs of the elements (as defined in the as-designed 3D model, e.g. IFC file) that the input point cloud is to be matched with. The GeometricQC tool is in charge of reading this information, obtaining, and preparing the necessary data in the appropriate data format for the Point Cloud Matching and Segmentation sub-component class.
- **Output folder path**: folder path where to store all the outputs that are generated by the tool. In the case of the Point Cloud Matching and Segmentation sub-component, it is the output path where to store the segmented point clouds.

In the case that the Point Cloud Matching and Segmentation sub-component is used as part of the GeometricQC tool, the entire solution can run from the command prompt. In its second release, to be consistent with the rest of the COGITO solution, the GeometricQC tool will expose an execution interaction with the Digital Twin Platform

4.4 Application Example

The Point Cloud Matching and Segmentation sub-component includes all the main methods required to perform Scan-vs-BIM. It is used to load the point cloud data and run the matching algorithms to have linked both the as-built and as-designed of the BIM components data, serving as input for the geometric quality control. To work properly, the Point Cloud Matching and Segmentation sub-component only requires the point cloud file path and the list of the triangle meshes of the as-designed BIM components (this input can come by connecting and using the BIM Element Manager sub-component). In this section, we will show the different outputs obtained with the implemented matching algorithms for a laboratory test set and a real case scenario.

4.4.1 COGITO Basic Artificial Test BIM model and point cloud

As already explained in Section 3.4.1, the COGITO Basic Artificial Test dataset is a synthetic laboratory dataset that facilitates development and testing. As discussed below, it also allows to create different unsatisfactory conditions, where the GeometricQC tool can detect discrepancies between the as-designed and the as-built data.

During the development test, two versions of the point clouds have been created, both containing 1 million points (Figure 8). The first one does not contain any noise, and all the points belong to a surface of the as-designed geometry. The second version is the same point cloud but with the addition of 2 cm standard deviation noise to each of the points, making a more challenging and realistic scenario for the matching algorithms.

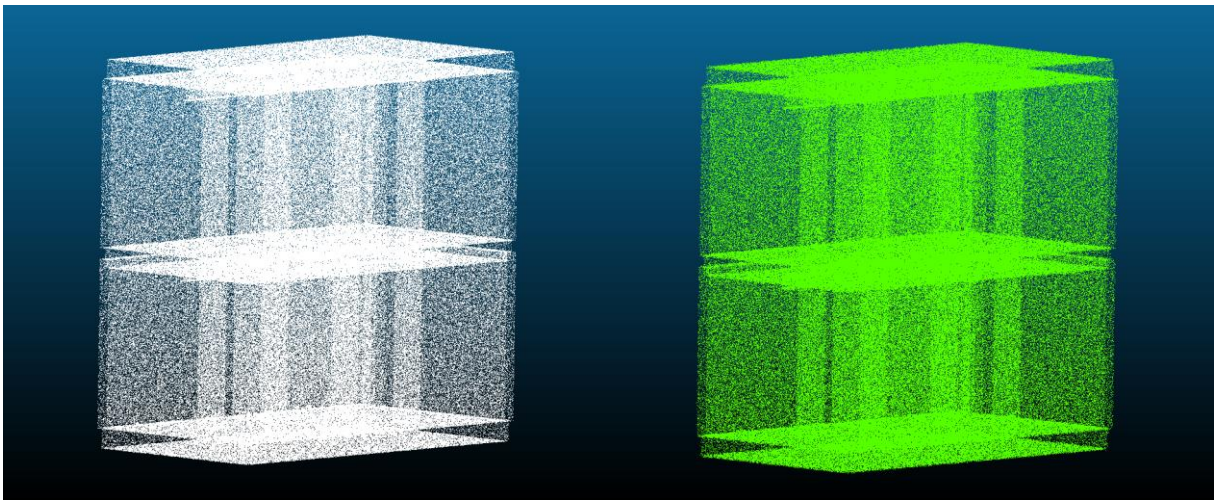


Figure 8: COGITO Basic Artificial Test datasets. (Left) Point cloud generated from the as-designed triangle mesh. (Right) Point cloud generated from the as-designed triangle mesh with additional 2cm standard deviation noise.

Two different tests have been conducted for both point clouds versions. For the first test, we compare the processing time it takes each algorithm to match and segment 18 BIM components from the IFC file, including walls, columns and slabs. Results are shown in Table 5.

The Brute-Force (Distance + normal vector similarity) algorithm is a slow matching algorithm, but the simplest one in both implementation and logic, that also gives useful results, allowing it to be used as a benchmark method. The results indicate that the two fastest approaches are the ones that use a voxel structure. A more detailed analysis can be performed in the plane matching algorithm. The algorithm takes a lot of time in pre-processed the input point cloud, computing the normal and segmenting the planes, but the matching step is actually very fast (23.0 secs from the overall time for the original dataset, and 14.2 secs for the noisy one). Arguably, the point cloud would be pre-processed only once, and the extra features obtained from it would be useful in other sub-components. However, as can be seen in the second test, the quality of the results are not as good as expected.

Table 5: Comparison of the processing time (in seconds) for each of the matching algorithms in the COGITO Basic Artificial Test dataset

	Distance + normal vector similarity	Voxel subsample	Voxel Space	Plane Matching
COGITO Basic Artificial Test	78.2	40.1	41.0	1,483.0
COGITO Basic Artificial Test + 2 cm std noise	89.0	42.1	38.7	1,616.3

For the second test, the quality of the segmented object was compared. A total of 18 BIM components were matched and segmented, and the total number of segmented points have been collected in Table 6. As can be seen, the first 3 algorithms produce high numbers of matching points, obtaining a good geometric quality in the matching, which produces a suitable input for the sub-sequent geometry quality control processing (see Figure 9 to Figure 12). The quality of the results of both algorithms using a voxel structure are quite interesting, both appearing accurate and useful to serve as input for the rest of the GeometricQC tool. The

voxel space method slightly outclasses the voxel subsample method, filtering out some outliers or irrelevant points. The plane matching segmentation is the only one that does not generate a favourable outcome. Among others, the plane matching algorithm contains a larger number of parameters than the others. The performance of this algorithm is sensitive to these parameters and thus error-prone if their values are not properly set. Further experiments are going to run to analyse additional factors and improve its performance.

Table 6: Number of segmented points from the COGITO Basic Artificial Test dataset with 2 std noise

	Distance + normal vector similarity	Voxel subsample	Voxel Space	Plane Matching
Column1	4,564	5,156	6,378	1,148
Column2	4,506	4,682	5,579	2,247
Column3	4,464	5,272	5,898	3,117
Column4	4,452	5,498	6,341	2,622
Column5	4,805	5,585	6,455	1,378
Column6	4,769	5,011	5,718	710
Column7	4,850	5,644	6,009	2,641
Column8	4,816	5,689	6,472	3,796
Wall2	35,896	37,627	40,657	31,524
Wall3	62,182	64,392	69,294	9,473
Wall4	34,722	35,415	38,465	27,719
Wall5	34,625	35,101	38,923	17,816
Wall7	36,952	39,864	41,021	24,323
Wall8	64,477	67,640	69,943	6,532
Wall9	35,992	37,694	38,889	29,791
Wall10	36,700	38,142	39,773	13,742
Slab2	77,227	83,210	94,675	84,613
Slab3	84,670	91,955	84,781	21,089

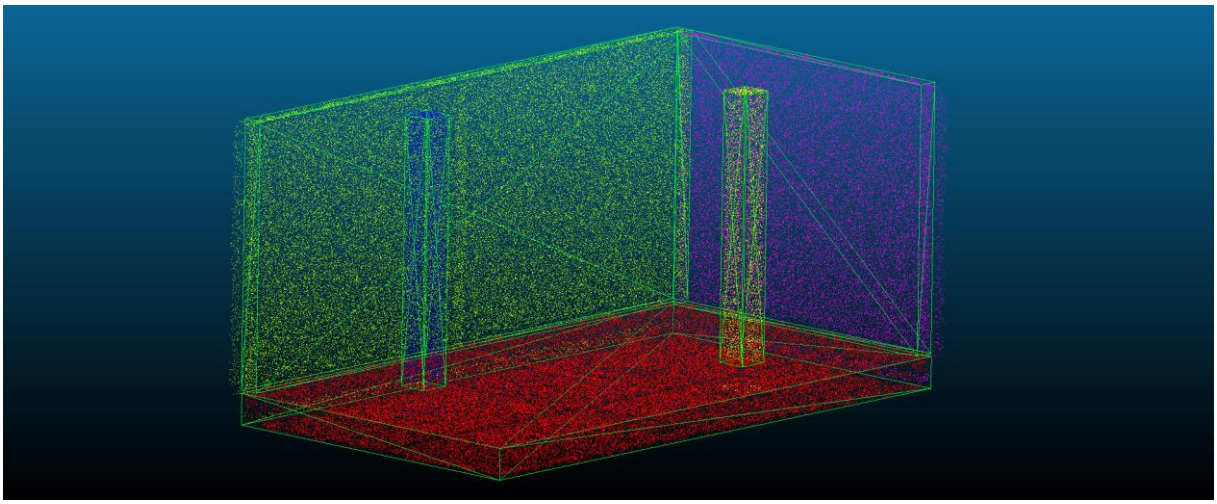


Figure 9: Distance and normal vector similarity matching output example

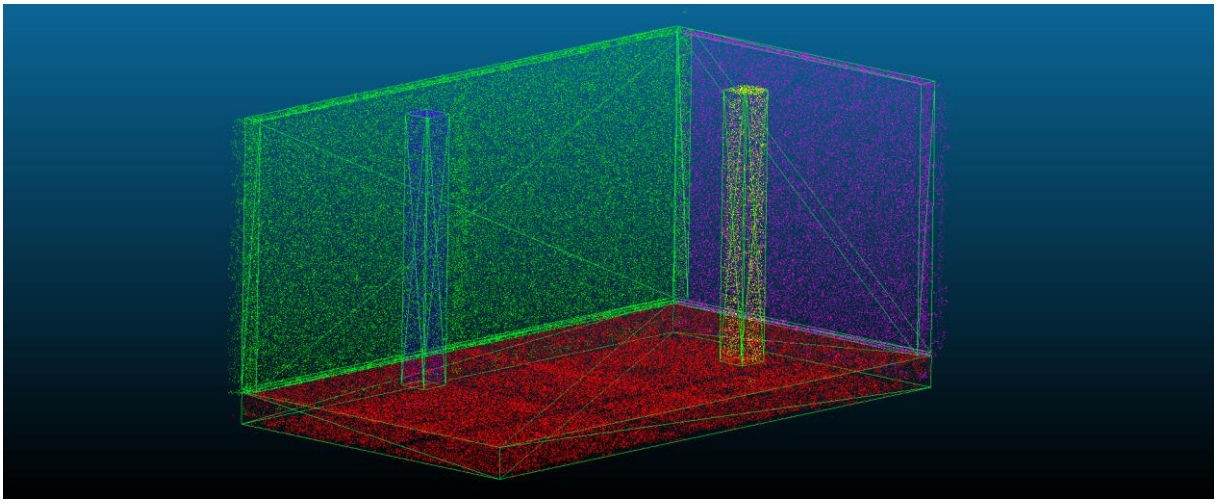


Figure 10: Voxel subsample matching output example

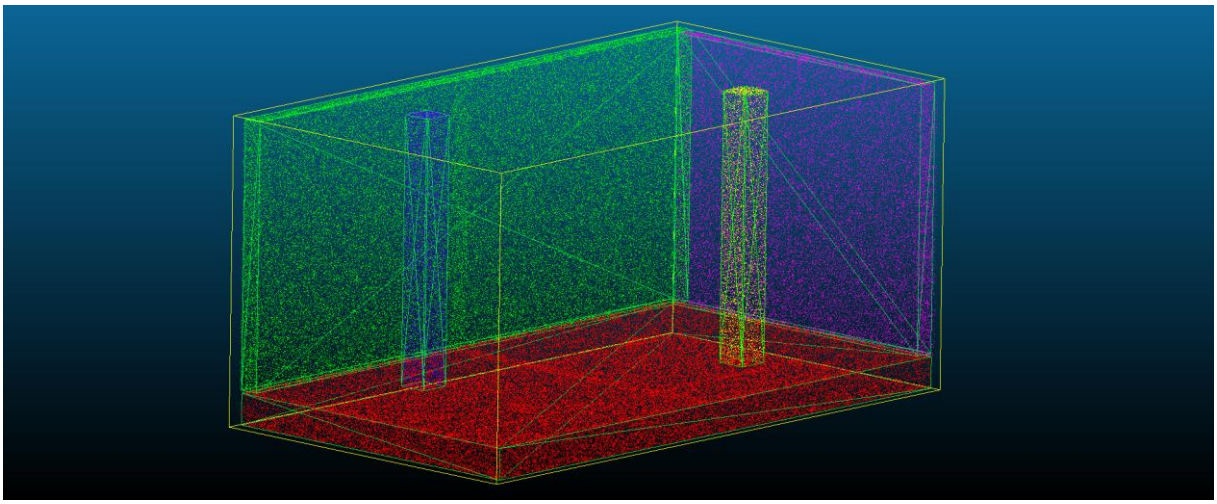


Figure 11: Voxel space matching output example

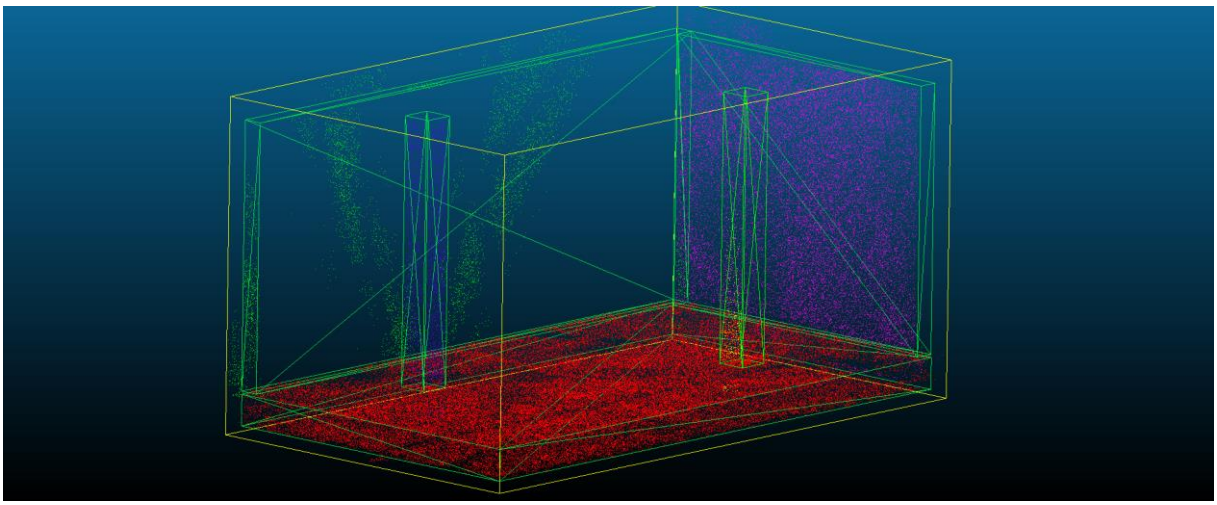


Figure 12: Plane matching output example

Comparing the quality of the segmented points and the processing time, the best matching and segmentation method appears to be the voxel space one. Thus, for the time being this method is chosen for

producing point-mesh matches used for developing the QC-focused sub-components of the GeometricQC tool in T5.3.

4.4.2 University of Waterloo – a real case scenario

The University of Waterloo is a test scenario that has been selected to evaluate the abovementioned point cloud matching methods in a real scale construction project. The provided dataset consists of a set of point clouds, captured at different dates along the construction of a building, and a mesh model of that building (derived from a BIM model). Here, we report results when testing our algorithms with a single input point cloud only. The point cloud contains 786,723 points, including colour, normal vectors, and intensity information. It also contains several points from clutter and occlusions, providing a challenging environment to test the Scan-vs-BIM solution. Figure 13 and Figure 14 show the mesh and the point cloud dataset used during testing process.

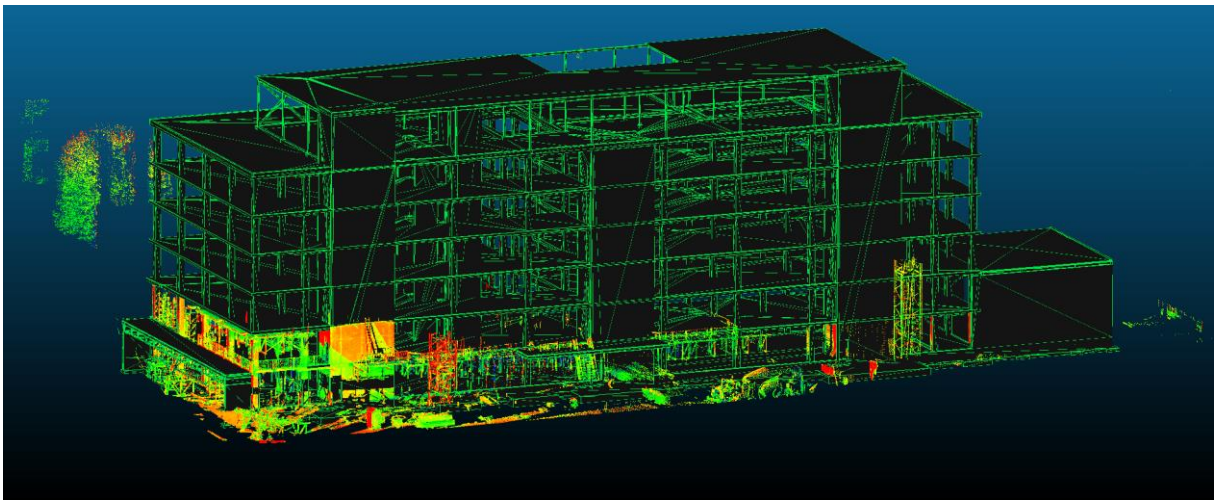


Figure 13: University of Waterloo mesh and point cloud dataset

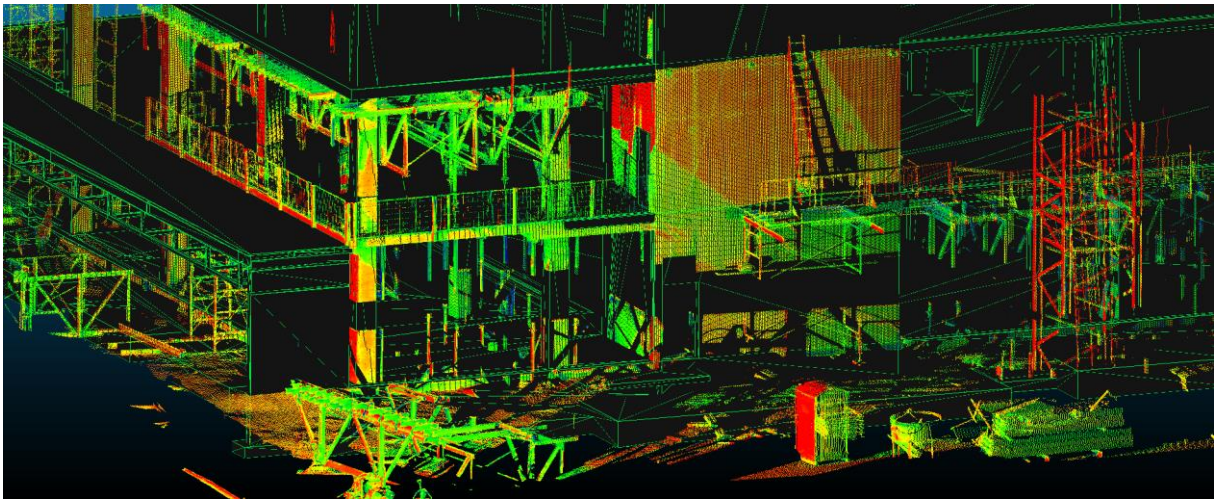


Figure 14: University of Waterloo mesh and point cloud dataset. Detail view

Two different experiments have been conducted with this dataset. The first experiment aimed to compare the processing time each algorithm requires to match and segment 9 triangular meshes, including walls, columns and slabs. Results are shown in Table 9. The results are quite similar to those in Section 4.4.1. The methods using the voxels are faster than the others. Despite the fact that the Brute-Force method is slower than the voxels-based methods, it is a reliable method and a good initial baseline to compare the other ones. The plane matching is the most computational expensive method, spending significant time in the normal

vector and plane segmentation steps, but negligible time in the matching step (4.4 secs of the overall process).

Table 7: Comparison of the processing time (in seconds) for each of the matching algorithms in the University of Waterloo dataset

	Distance + normal vector similarity	Voxel subsample	Voxel Space	Plane Matching
Waterloo	94.0	32.2	23.6	1199.4

The second experiment aimed to compare the quality of the segmented objects and the total number of points in each of them. Results are presented in Table 8, as well as Figure 15 to Figure 18. The same conclusions as in Section 4.4.1 can be drawn. The only remarkable difference is that in this case the voxel space method obtains a segmented object with some small holes in them, which does not occur in the voxel subsample method results. This can be due to the voxel and the triangle size, and the number of points inside the voxels. The point cloud does not contain that many points, thus it is challenging to obtain quality results. In contrast, the voxel space manages to filter out points that are outside the mesh boundaries, making it more robust in this respect.

It is important to mention, that in all the cases, the algorithms are able to filter out the clutter data from the construction objects, providing a clean object segmentation that is useful to the geometric quality control step.

Table 8: Number of segmented points from the University of Waterloo dataset

	Distance + normal vector similarity	Voxel subsample	Voxel Space	Plane Matching
Object128	3,289	3,346	3,511	1,749
Object130	1,595	1,590	1,660	841
Object140	5,314	5,066	5,401	458
Object142	8,878	9,160	9,102	2,698
Object146	3,165	3,047	3,080	652
Object149	5,342	5,317	5,666	732
Object510	24,796	23,350	26,441	0
Object511	46,695	46,196	48,135	3,073
Object1536	7,674	7,109	9,525	4,180

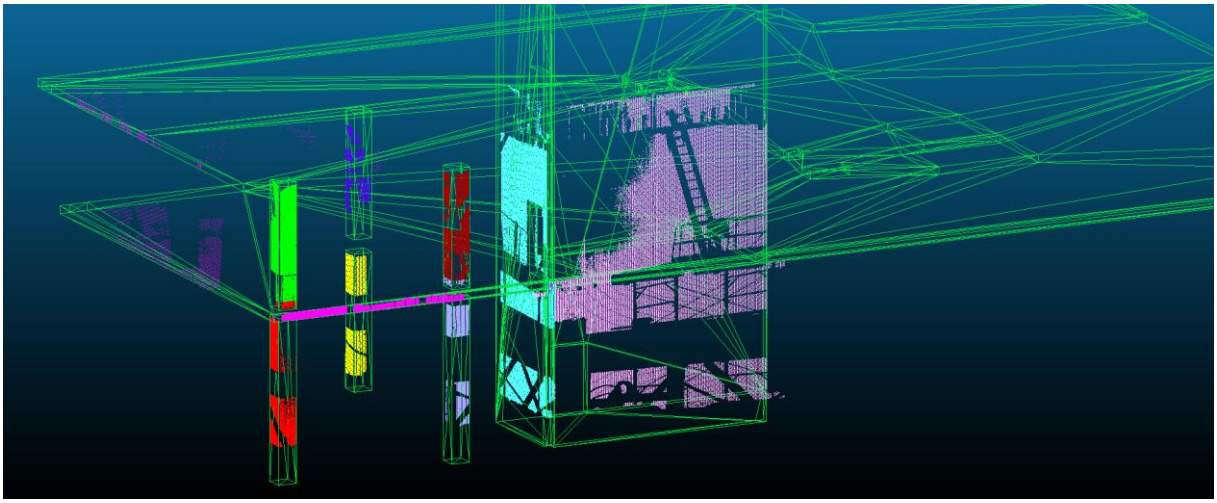


Figure 15: Distance and normal vector similarity matching output example

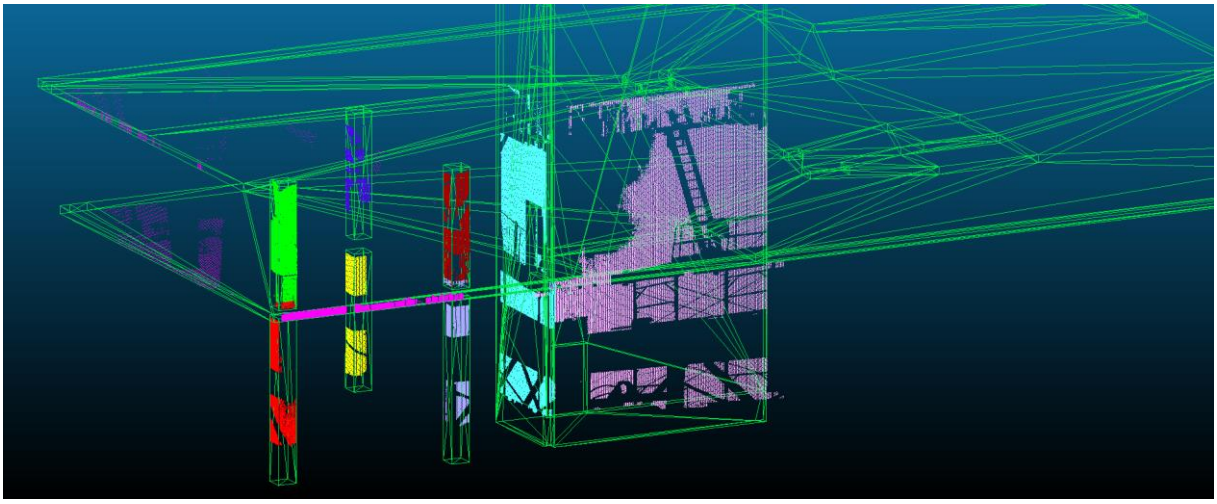


Figure 16: Voxel subsample matching output example

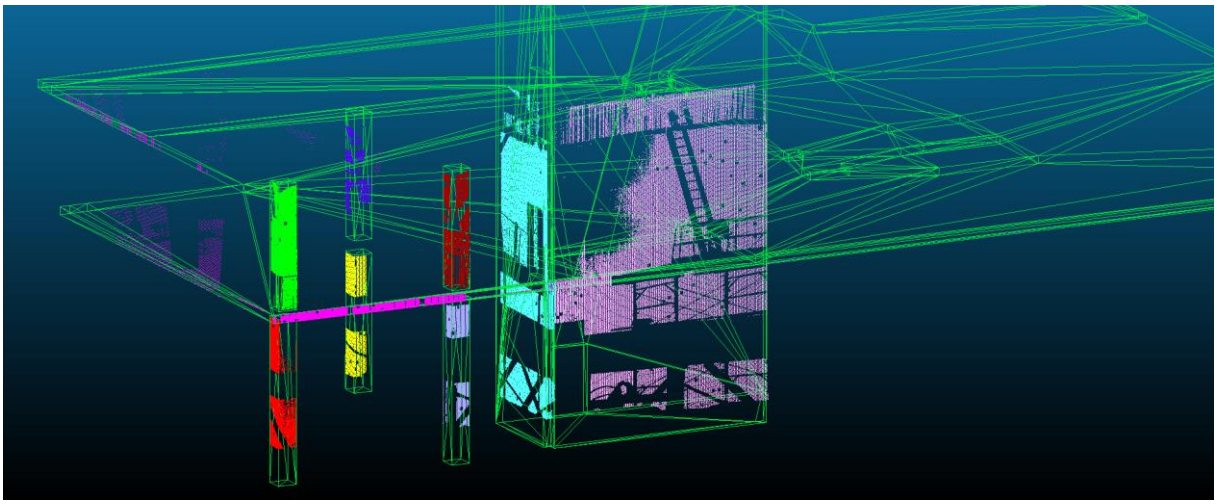


Figure 17: Voxel space matching output example

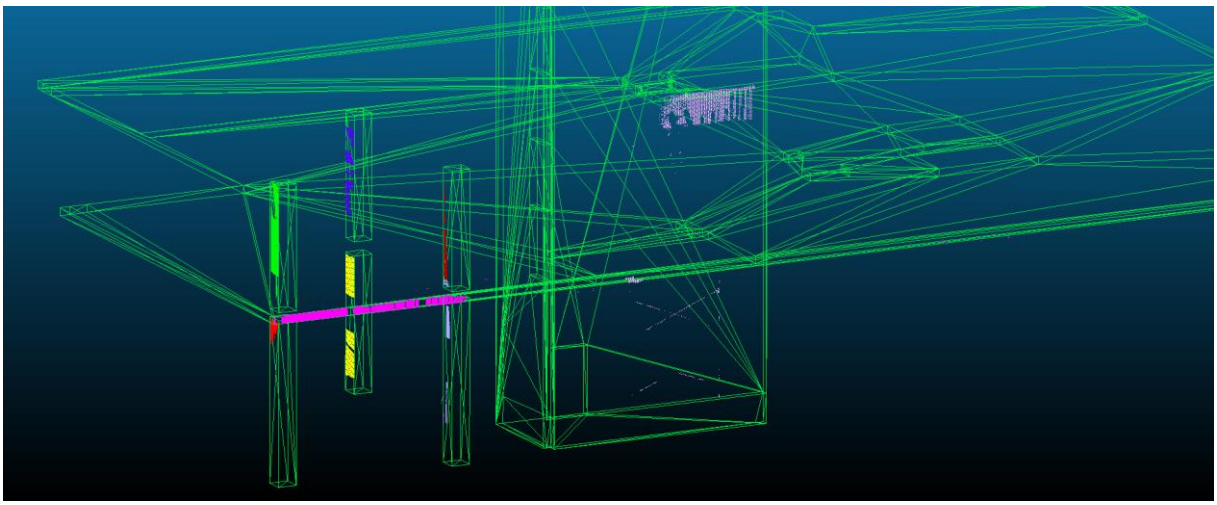


Figure 18: Plane matching output example

4.5 Licensing

The Scan-vs-BIM solution is free software; you can redistribute it and/or modify it under the terms of the **GNU General Public License Version 3** as published by the Free Software Foundation (https://www.gnu.org/licenses/gpl-3.0.en.html).

4.6 Installation Instructions

No complex installation is required. Binaries will be provided in the Cyberbuild Datashare collections. The Point Cloud Matching and Segmentation sub-component is a set of header-only libraries that can be included in any project.

4.7 Development and integration status

Although the Point Cloud Matching and Segmentation sub-component remains under development, the first version of the matching algorithms that had been implemented, especially the voxel space one, already produces good quality and accurate results for the Scan-vs-BIM solution. The plane matching method is the only one that will be investigated further in the second release of the component.

Regarding the second release of the component, the intention is to introduce new matching algorithms, using object segmentation and classification from a pre-processed point cloud, in order to improve the matching results when the scanned data contains more clutter and noise. Additionally, as mentioned above, the plane matching method will be revisited. Finally, the Scan-vs-BIM solution only allows the tool to be run with point clouds in the PLY format, therefore, the second release will include the required E57 open format by including an extension to the Open3D library to load and save the point cloud data in its internal structure.

The Point Cloud Matching and Segmentation sub-component (and similarly Scan-vs-BIM solution), does not interact with any other external component of the COGITO solution; all its interactions are within the GeometricQC tool.

4.8 Requirements Coverage

Although the Point Cloud Matching and Segmentation sub-component is delivered as a back-end solution of the Scan-vs-BIM and the GeometricQC tool, it already covers several of the requirements defined in D2.4 and D2.1.

The functional and non-functional requirements are presented in Table 9. Req-1.1 is partially covered using this sub-component, thanks to the point cloud loader and interpreter. However, despite loading the geometry of the BIM component (by using a triangle mesh loader), the Point Cloud Matching and Segmentation sub-component does not have the capability to load and interpret the IFC data. Req-1.2 will be covered in the second release of this component. Thanks to the C++ property of using different objects in header only classes, Req-2.1, Req-2.2, and Req-2.3 are well covered by this version of the component. As stated above, this set of classes (as part of the full GeometricQC tool library) can be used autonomously in any other project, or as part of the full GeometricQC tool. In addition, the scalability requirement is met due to the object-oriented programming, streamlining the addition of new functionalities and properties to each of the components.

Table 9: Point Cloud Matching and Segmentation sub-component requirements coverage from D2.4

ID	Description	Type	Status
Req-1.1	Loading as planned 4D BIM and point cloud data	Functional	Partially achieved
Req-1.2	Object detection in point cloud	Functional	Not yet supported
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Reusability	Non-Functional	Achieved
Req-2.3	Interoperability	Non-Functional	Achieved

Table 10 presents the stakeholders requirements documented in D2.1 which are relevant to this component. COGI-CS-1 and COGI-CS-4 are covered simply by the programming language selected for development. COGI-QC-8 is partially achieved by matching the as-built data to the as-designed data, allowing to process this matched data in the subsequent GeometricQC tool procedures. With regard to COGI-QC-11, the Point Cloud Matching and Segmentation sub-component allows to load and interpret the as-built data from the acquired point clouds, allowing the interconnection with other sub-components to automate the subsequent QC processes. Finally, COGI-QC-16 is partially achieved by using the Open3D library, which is able to handle the point cloud PLY format. The E57 format will be introduced in the 2nd release of the component.

Table 10: Point Cloud Matching and Segmentation sub-component stakeholders requirements coverage from D2.1

ID	Description	Type	Priority	Status
COGI-CS-1	Runs on desktop or laptop PC	• Operational	Must	Achieved
COGI-CS-4	Runs on Windows	• Operational	Must	Achieved
COGI-CS-5	Runs on Mac	• Operational	Could	Achieved
COGI-QC-8	Supports systematic quality control on earthworks, substructure, concrete works	• Performance	Should	Partially achieved
COGI-QC-11	Automates QC-related activities	• Functional • Operational	Must	Achieved
COGI-QC-16	Handles point clouds standard formats (E57, PLY)	• Functional	Must	Partially achieved.

4.9 Assumptions and Restrictions

The Point Cloud Matching and Segmentation sub-component has a number of assumptions and restrictions, which are presented in the following:

- BIM elements and point clouds are expected to have the vertical along the +Z axis using the right-hand side coordinate frame.
- The coordinate reference frame within the entire GeometricQC tool uses global coordinates, hence, the generated output files contain coordinates information in the same global space.
- The measures are stored using the international metric system, and the distance measurement unit used in the sub-component is the metre [m].
- The input as-built data has to be pre-registered in the same global coordinate system as the BIM file, minimising the registration error to ensure an accurate point cloud matching and segmentation.
- It currently only supports *.ply* files, although it is expected to include support for E57 file format in the 2nd version.
- It currently only handles one input point cloud file to perform the matching. In the 2nd version, it will include multiple (pre-registered) point clouds input support.

5 Conclusions

The Scan-vs-BIM solution is to be employed to generate the input data for the geometric QC components. The solution uses as input the as-designed data in the form of the as-designed BIM model (digital semantically-rich 3D model of the building/asset created by the design team), and the as-built point cloud data obtained through laser scanning surveying during construction. The BIM model data is interpreted from an IFC file, while the point cloud data must be pre-registered in the coordinate system of the project (i.e. the same coordinate system as the BIM model).

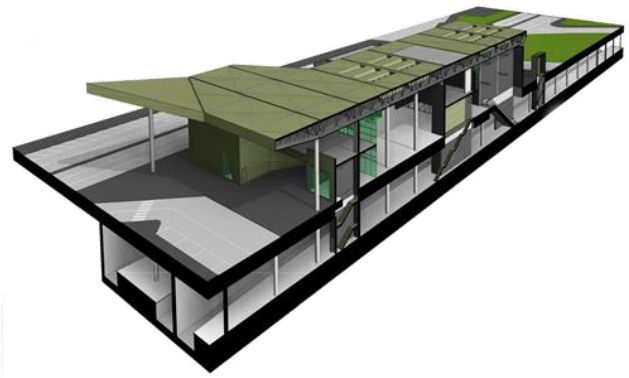
As documented in this deliverable (D5.1), the core functionalities of the Scan-vs-BIM solution are delivered by two sub-components. The main sub-component, namely the *Point Cloud Matching and Segmentation sub-component*, performs the matching of the as-built point cloud data to the as-designed 3D meshes of the components in the BIM model. This results in a segmentation of the input point cloud with each segment containing the 3D cloud points matched to a given 3D component in the BIM model (and an additional segment containing non-matched points). The secondary *BIM Element Manager sub-component* is the middleware tool to read, interpret, and manage information about the components in the BIM model (e.g. mesh geometry, component type, etc.). The sub-components (and the Scan-vs-BIM component overall) have been designed to be used by the GeometricQC tool, which automates geometric QC and exports the results to the Digital Twin Platform. These components have also been designed to be modular, and scalable, allowing the creation of a toolbox, which can be used in any other project. The Scan-vs-BIM solution has been tested with basic artificial test data and with real case data.

References

[1] BS EN 13670:2009, “Execution of concrete structures,” 2010.

[2] BS EN 1090-2, “Execution if steel structures and aluminion structures. Technical requirements for steel structure,” 2018.

[3] M. Fischler and R. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography,” *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1982.



COGITO

CONSTRUCTION PHASE DIGITAL TWIN MODEL

cogito-project.eu

HYPERTECH energy labs

UCL

AARHUS UNIVERSITET

THE UNIVERSITY of EDINBURGH

BoC www.boc-group.com

iti

POLITÉCNICA

QUE TECHNOLOGIES

Novitech New Information Technologies

ASM

ferrovial construction

ΟΛΥΜΠΙΑ ΟΔΟΣ

RHOMBERG SERSA RAIL GROUP

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310