2022 EC³
2022 European Conference on Computing in Construction
Ixia, Rhodes, Greece
July 24-26, 2022

# IFC-BASED GENERATION OF SEMANTIC OBSTACLE MAPS FOR AUTONOMOUS ROBOTIC SYSTEMS

Muhammad Anas Gopee[1], Samuel A. Prieto[1], and Borja García de Soto [1]
[1]S.M.A.R.T. Construction Research Group, Division of Engineering, New York University Abu Dhabi (NYUAD), Experimental Research Building, Saadiyat Island, P.O. Box 129188, Abu Dhabi, United Arab Emirates

## Abstract

Autonomous Robotic Systems (ARSs) in the construction industry usually have to perform preliminary mapping of construction environments before deployment. For large and complex sites, this can be unpractical and time-consuming, making the avoidance of preliminary mapping a motivation for this study. With Building Information Modeling (BIM), a lot of information is already available about sites. This study proposes a method to make that information available to ARSs to streamline autonomous tasks and remove the need for mapping. This is achieved by automatically generating semantic and color-coded obstacle maps from IFC files. The results are obstacle maps that can be used for autonomous navigation that remove the need for preliminary mapping.

## Introduction

The Industry Foundation Classes (IFC) schema is a platform-neutral Building Information Modeling (BIM) format developed to facilitate interoperability. As such, an IFC file can be parsed to provide rich semantic information with relative ease compared to other proprietary BIM formats (e.g., RVT, NWD, DWG). Autonomous Robotic Systems (ARSs) in the Architecture, Engineering, and Construction (AEC) industry often rely on sensors to do a preliminary mapping of their environment. This is not very efficient for large and complex construction sites. The rich semantic information present in IFC files can be leveraged to remove the need for such mapping, or to complement it with additional semantic information. IFC files usually contain everything from spatial elements to geometry. This makes them a good candidate for extracting semantic information for autonomous applications. Of course, IFC files do not contain all required information for navigation (such as machines and temporary structures). LiDARS and other sensors, typically used by ARSs during navigation, can be used to update the map accordingly as new obstacles are detected. IFC files are structured in a hierarchical manner and entities that have corresponding attributes can be identified based on different tags or labels, nested within each other. This structure makes them relatively easy to understand at a high level and parse. Figure 1 shows in a tree structure an extract of the IFC file for a simple house model. The portion of the file shown corresponds to spaces present in the first storey of the building. IFC files generally contain an item with the IFCPROJECT tag that is decomposed into IFCBUILDINGSTOREY items, each representing a story of the building. Each IFCBUILDINGSTOREY is decomposed into IFCSPACE items that account for the spaces within a storey. The IFCSPACES have attributes that can relate to elements that are inside the space or bounding the space. The general trend in existing literature dealing

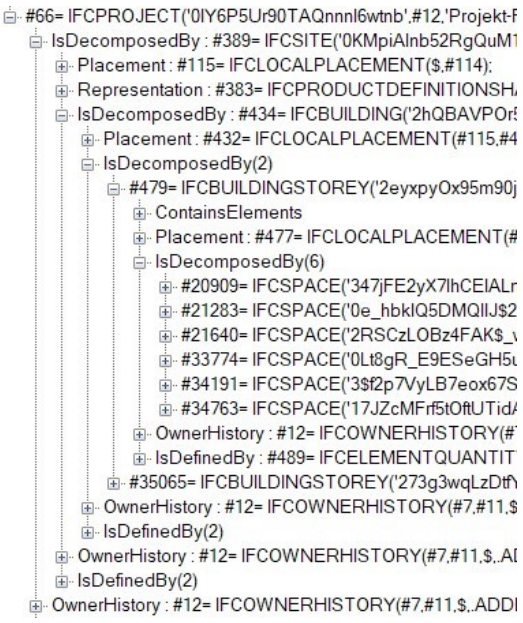


*Figure 1: Extract showing the structure of an IFC file for a simple house model*

with BIM-based construction automation is that the information extracted from the BIM is application-specific and proprietary. The focus of this study is to make the extracted information as versatile and universal as possible. Multiple robust path-planning algorithms already exist (e.g., A* (Hart et al. 1968), Dijkstra (Dijkstra 1959) and D* (Stentz 1994)) and are available on robotics software like Gazebo (*Gazebo Simulator* Accessed: 12/06/2021), ROS (*Robot Operating System* Accessed: 12/06/2021) and MATLAB Robotics Toolkit (*MATLAB Robotics Toolbox* Accessed: 12/06/2021) or game engine platforms like Unity3D (*Unity* Accessed: 12/06/2021). The aim is to create an interface between BIM and those path-planning software. IFC files contain valuable semantic information about the whole building model. This can be useful with applications that make use of the relevant as-planned details of a model. For example, applications such as progress monitoring can rely on this data to generate progress reports. As such, it is useful to store the data present in IFC files in a structured way so that it can be easily accessed and understood by an ARS. Considering applications that occur before the construction phase is over (i.e., the navigation areas are incomplete), it is also important to update navigation maps considering the schedule of construction tasks.

## Previous work

This section details the previous work done on the topic of information extraction for robot navigation. It is not about navigation itself but rather about the information needed for autonomous navigation. The IFC parsing section looks at work done about extracting information from IFC files. The obstacle map generation section looks at how information from IFC or other BIM formats has been used to generate useful navigation maps.

### IFC Parsing

Currently, the parsing of IFC files is done either by the use of existing libraries (Krijnen Accessed: 12/06/2021)(Gerold Accessed: 12/06/2021) or custom parsers (Wang & Tang 2021)(Dimyadi et al. 2008)(Kim et al. 2013) that leverage the ordered structure of IFC files. The general method is the use of tokenizers that depend on the delimiters present in IFC files to break down the multiple attributes. A powerful open-source library, IfcOpenShell (Krijnen Accessed: 12/06/2021), is often used for IFC parsing purposes. Examples of previous works related to IFC parsing are summarized below.

Wang et al. (Wang & Tang 2021) proposed a method for parsing IFC files and storing the obtained data in a MySQL database for persistence. They leveraged the object-oriented design pattern of IFC to analyze the files using Java. They developed their own module to parse the IFC files that included a tokenizer and a syntactic parser. The end-product is a robust relational database storing information from the IFC. While this could be useful for many applications, some more processing might be required for use in an ARS. For example, it is worthwhile to standardize the geometry information as an ARS would not be able to easily understand the many different geometric representations present in IFC. Parsing and storing the whole IFC file may also not be the best option for use in ARS navigation. With more data present in the database, the processing time to obtain the required information will increase. If real-time applications are considered, this may not be optimal as computing power is limited. The fact that they develop their own module for parsing is also a bit limiting as well, given that powerful libraries like IfcOpenShell exist.

Kim et al. (Kim et al. 2013) extract semantic information from IFC files in order to generate construction schedules. They achieved that by extracting element geometry and placement. The model was split into work zones, and the work zone in which an element is situated was used to determine the scheduling of the related construction tasks. Rich semantic information was extracted from each element in this study; however, manual input was still required. For example, the position and size of the work spaces were defined by the user. The model could be automatically divided into spaces that would make sense for an ARS (e.g., in a way that depends on bounding elements and obstacles.)

In a study detailing how IFC data can be used in a fire simulation system, Dimyadi et al. (Dimyadi et al. 2008) proposed a method of parsing IFC files to obtain relevant information. The approach, however, focused only on the interpretation of space geometry, connections between spaces and basic material properties of elements such as walls and doors. While an interface between IFC and a fire simulation system was successfully created, the limited information might only be suitable for those kinds of systems, and not for navigation systems that require more detailed semantic information.

Overall, it can be said that previous work related to parsing and storing IFC data is not optimized for ARS navigation. For such an application, the information must be filtered and processed to be relevant for navigation and also allow better real-time performance.

### Obstacle Map Generation

BIM-based obstacle map generation is not a new topic. Previous studies have investigated this idea in multiple forms, albeit with some limitations. Despite most of the research being focused on the generation of navigation models (Liu et al. 2021), very few approaches deal with ARS specific navigation or have into account that the information can flow both ways (from the IFC model to the ARS and viceversa). For example, Song et al. (Song et al. 2020) included a section for BIM-based map construction in their paper about autonomous surveillance UAVs. Their method involves the generation of a point cloud from the BIM model. This works fine for basic obstacle avoidance, but there is a loss of valuable semantic information. Especially since their method involves converting the file into OBJ format, which only stores geometric information of the model.

Lin et al. (Lin et al. 2013) proposed a 3D approach for path planning with IFC files. This method is particularly interesting as it allows easy visualization of how an ARS would move from one story to another. It also has minimal loss of semantic information. However, the approach uses a proprietary program for path-planning, and the generated 3D maps are not as universal as a regular 2D. 2D maps in the form of bitmaps or vector images can easily be loaded onto path-planning software, whereas 3D models may not be as straightforward. The approach also does not consider the state of the building in stages before construction, meaning that some AEC applications that occur before the completion of a building cannot rely on it. Xu et al. (Xu et al. 2017) performed a similar kind of path-planning using IFC. However, their method is more appropriate for understanding the shortest connecting paths inside a building for human movement and does not consider generating a detailed map for the navigation of an ARS.

Karimi et al. (Karimi et al. 2021) and de Koning et al. (de Koning et al. 2021) present two of the few approaches specifically focused for ARS navigation, providing an output suitable and usable for the ARS. Karimi et al. propose an ontology-based approach that extracts the information from the IFC model and builds a graph-based model con-

taining the information of the spaces and how they are linked between each other. Nonetheless, their approach is based on the assumption that the IFC contains all the information needed, which based on practical aspect of the industry, is usually not the case. The need for a specific set of information present in the IFC makes the approach less robust and usable in real working conditions. The work by de Koning et al. also proposes a graph-based generation of a navigation map from the IFC data. It is based on the generation of a "digital-twin" of the building used to share information between the ARS and the IFC model, establishing a communication that works both ways. The way to extract the information is based on a plugin specifically developed for REVIT, that is highly dependant of the modelling convention of the specific IFC model, which could make that approach less robust and usable under real construction site conditions. Hamieh et al. (Hamieh et al. 2020) proposed an indoor path planning method called BiMov for the generation of navigation maps with different levels of detail based on IFC. The first and second levels of detail are maps made for studying building accessibility and pedestrian movement. The third level is the one that could be suitable for an ARS to navigate the building. They also consider the calendar model (project schedule) to verify the accessibility status of certain spaces based on the point in time. This is particularly useful for autonomous systems during the construction phase. BiMov makes use of the IfcOpenShell library to process the IFC files. However, information obtained after parsing from the IFC files is not stored in a way that can be easily accessed by an ARS.

In summary, previous studies that deal with some sort of map generation from IFC files generally focus on path-planning from point A to B. There is no real focus on generating a map that would be versatile and usable on other platforms. In order to provide an interface between IFC files and ARS, it is worthwhile to generate navigation maps that can be used on existing platforms, for example, a 2D map in the form of a bitmap or vector graphic.

## Methodology

The proposed methodology extracts relevant information from IFC files to generate semantic obstacle maps that can be used by ARSs for navigation. Semantic information is color-coded and added to the 2D obstacle maps. Other semantic information useful for navigation, such as door operation types, are also stored so that they can be easily accessible when/if needed. By the end of the process, the text-based information present in the IFC file (which is hard to parse in real-time for an ARS) is simplified and structured in two input arguments: a JSON file containing relevant information and an obstacle map bitmap image to be used in the navigation. The generated maps will only be as detailed as the IFC itself. As such, to obtain a usable obstacle map, the IFC model used must be complete and not be missing any important model information. Being complete in the context of ARS navigation would mean having obstacles (e.g., walls, changes in elevation, etc.) defined, as well as the connections between spaces such as doors and stairways. The methodology is broken down into three parts: 1) IFC Parsing, 2) Storing Parsed Data and 3) Obstacle Map Generation. Figure 2 shows an overview of the whole methodology. The code written for this study is open-source and can be found on Github (*SMART-NYUAD/ifcobstaclemap* Accessed: 12/06/2021).

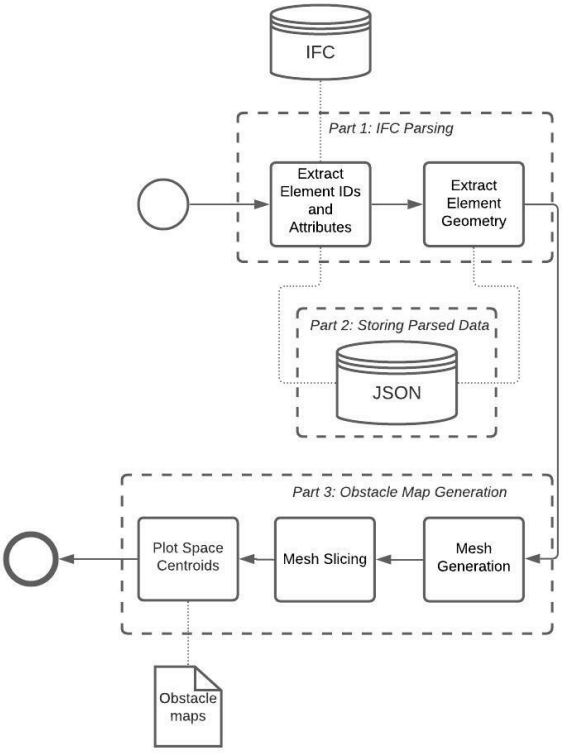


*Figure 2: Overview of the proposed methodology*

### Part 1: IFC Parsing

For this study, instead of focusing on creating a custom IFC parser to obtain semantic and geometric information, IfcOpenShell is used. IfcOpenShell is an open-source library for C++ or Python that uses OpenCASCADE (*OpenCASCADE* Accessed: 12/06/2021) internally to convert the implicit geometry in IFC files into explicit geometry that any CAD software or modeling package can understand. There is no need to create a custom parser when IfcOpenShell is already robust to different types of IFC schema (such as IFC2x3 and IFC4 which were tested in this research) and is constantly updated to support new IFC features.The library also allows for the simple extraction of IFC data. Getting attributes of any IFC data can be done by simply using dot notations in Python. For example, *IfcDoor.OverallWidth* returns the width of a certain door. For the parsing of IFC files in this paper, IfcOpenShell is used on Python3.

*Geometry Extraction*

Geometry is one of the more delicate attributes when it comes to IFC. Elements can have different types of representations such as compound solid geometry, swept geometry and boundary representation (Tobiá 2015). If one was to manually parse IFC geometry, an in-depth understanding of the different representations would be needed. To go around this, the shape processing features of IfcOpenShell can be used. IfcOpenShell shape processing allows the conversion of most geometric representations into vertices, faces and edges information. This is leveraged for the purpose of this paper.

In order to obtain accurate obstacle maps, there needs to be a robust method to obtain the geometry of any possible obstacle. The geometry iterator of IfcOpenShell is used to iterate through every IfcElement present in the file to create meshes for each one of them. This allows further mesh processing to be done on the elements to obtain relevant information. This will be further detailed in the obstacle map generation section. Overall, this method can be used to provide access to standardized geometric information of individual elements, which can be useful to an ARS or project monitoring team. IfcOpenShell also allows the geometry to be extracted based on the absolute coordinates in the IFC file, with the USE_WORLD_COORDS parameter set to True. Using this method, the obtained vertices, edges and faces for the geometry also include placement information. This removes the need of having to deal with IFC placement, which is usually hard to obtain as individual element placement attributes are given relative to other container elements rather than the absolute placement.

*Navigation Information*

Besides geometry, there are a lot of other types of information stored in IFC files that can be useful for navigation. Considering that a color-coded obstacle map will already be made available to the ARS, the information shown in Table 1 may still be useful for navigation.

*Table 1: Example of attributes useful for navigation*

| IFC Element | Attributes |
|---|---|
| IfcDoor | OperationType |
| | OverallHeight |
| | OverallWidth |
| IfcStairFlight | NumberOfRiser |
| | NumberOfTreads |
| | RiserHeight |
| | TreadLength |

Assuming that doors would be the primary method of navigating between two connected spaces, it is important to have at least some basic semantic information about the different door types. If the ARSs were to interact with a door, knowledge about the door operation type (e.g., single vs. doubled-hinged, right vs. left swing, knob vs. lever handle) is crucial. For stairs, the main method to navigate between two vertically connected spaces, the ARS can use information such as the number of steps, step height and width to determine how to navigate based on its locomotion method (i.e., wheels, tracks, legs, etc.).

**Part 2: Storing Parsed Data**

Storing data that can be useful for navigation in a separate file from the IFC has several advantages. While the IFC format was created to be platform neutral and to facilitate information exchange, the interoperability it provides is between different BIM software and not so much to other platforms. IFC is not a file format that would be easy to process for an ARS. There is a lot of information that may not be relevant for such platforms. Irrelevant information from the IFC may significantly slow down processes such as information query. This would not be the most efficient for applications that need to access this type of data in real-time. As such, it is worthwhile to store navigation-relevant information in a file that is universal and versatile, and the data chosen to be stored must only contain relevant information. For those reasons, the JavaScript Object Notation (JSON) format was considered for this research. JSON is an open standard file format and data interchange format that is language independent, making it ideal for data interchange (*Introducing JSON* Accessed: 12/06/2021). This is suitable for our purpose as it provides flexibility on the language used by the ARS.

The data parsed from the IFC, structured in nested Python dictionaries, can easily be dumped into a JSON file. The JSON file then allows for information to be queried. The type of information to be stored in the JSON file is selected within the scope of ARS navigation. For elements such as stairs or any obstacle, information that would be useful for navigation purposes can be obtained from the geometry. Stair attributes, such as the number of risers or riser height, can be calculated from mesh information. However, there is no way to easily determine the OperationType attribute of doors through geometry, and this information is crucial for an ARS that needs to interact with doors. As such, the OperationType is given priority to be stored in the JSON file. The IfcGlobalId attribute, which is unique for each element in an IFC file, is chosen as the key in the Python dictionary. Values that are stored for all elements are the geometry information (in terms of vertices, faces and edges) and element type (e.g., IfcWall, IfcDoor, IfcWall). For IfcDoor, the OperationType, OverallWidth and OverallHeight are also stored as these attributes are essential for navigation. The door operation type is stored with the same tag that it has in the IFC file. For example, for a door that is single-hinged and swings left, the operation type is stored as SINGLE_SWING_LEFT. Detailed explanations of each operation type tags can be found on the buildingSMART website (*IfcDoorTypeOper-*

*ationEnum* Accessed: 12/06/2021).
Figure 3 shows an example of how the information is stored for the door with ID: 18PQUIFELEARLM4c0brquf. The figure is a representation of the JSON file instead of a snapshot of the JSON file itself in order to make the information more legible. The ID 18PQUIFELEARLM4c0brquf has values that give the geometry (in the form of vertices(verts), faces and edges), the element type, which is IfcDoor and the door operation type, which is revolving in this case. The width and height of 2 *m* and 2.1 *m* are also stored. With this structured and easy to parse JSON file, the need for the ARS to deal with complex structures and useless (in terms of navigation) information stored in an IFC file is completely removed.

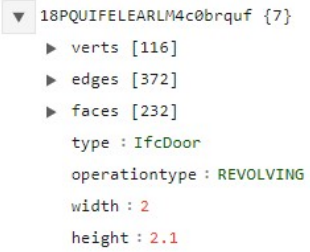


*Figure 3: Example of IfcDoor element stored in JSON*

**Part 3: Obstacle Map Generation**

*Mesh Generation*

In order to generate an obstacle map, the geometric information for each element must be standardized. As mentioned in the Geometry Extraction section, the geometry processing features of IfcOpenShell are used to obtain vertices, faces, and edges information for each building element. From this information, a triangulated mesh can be produced. Meshes are chosen as the standard representation as mesh processing is relatively common and has multiple available libraries, such as Trimesh (Dawson-Haggerty et al. Accessed: 12/06/2021) for Python.
Trimesh is chosen for the purpose of this study as its mesh concatenation feature is useful to combine the meshes of the different building elements to generate a unified mesh of the whole building. This unified mesh can then be used to generate the obstacle maps. In order to add semantic information to the meshes, a color attribute was used to differentiate between the main types of building elements. The differentiation was done within the scope of autonomous navigation. The three main types of building elements present would be 1) obstacles, such as walls and furniture, 2) openings/horizontal space connections, such as doors, and 3) vertical connections, such as stairs. Figure 4 shows examples of generated meshes for different elements. Once the meshes of each element are created, they can be concatenated to create a unified mesh for the model. This unified mesh also retains the color information that is added based on the element type, and can be used to have an overview of the navigability of the model. Figure 5 shows an example of a unified mesh created from a simple house model. While IfcOpenShell already has some tools such as IfcConvert to generate vector graphics of floor plans, they do not add any semantic information to the map, which is why the extra step of generating a colored-coded mesh is performed.

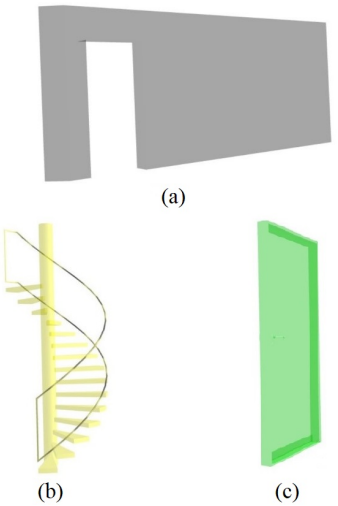


*Figure 4: Example of generated meshes for (a) wall, (b) staircase, and (c) door*

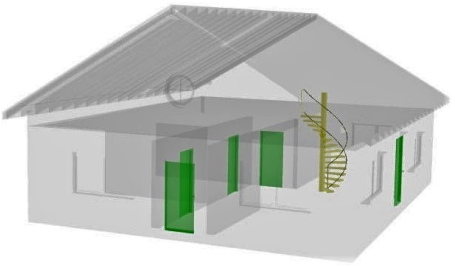

*Figure 5: Example of unified mesh*

*Mesh Slicing*

Obstacle maps for each story are then generated simply by taking horizontal cross-sections from the unified, color-coded mesh. The height at which the slicing is done can be adjusted based on the height of the ARS for which the obstacle maps are being generated to account for relevant obstacles. In order to generate those slices, Trimesh is not ideal as the slicing function does not consider the color information added to the mesh. This would result in the loss of the valuable semantic color-coding. As such, another Python mesh processing library, PyVista (Sullivan & Kaszynski 2019), is used. PyVista is more adapted to dealing with color information in meshes and gives more flexibility when generating the slices. Multiple slices were taken between the floor level of the story and the height of the ARS, and then stacked to give a whole picture of the obstacles that are relevant for the ARS. After stacking, the resulting obstacle map is exported as a bitmap image. This is repeated for all the stories of the building.

*Space Centroids*

ARSs are often fed waypoints (i.e., a point of reference used for location and navigation) during autonomous tasks, and those waypoints are often chosen manually. For certain applications, such as 3D laser scanning or taking images that can be used to assess the progress of construction work, the centroid of spaces can be considered useful waypoints to scan the space (e.g., a rectangular room). The proposed approach can automatically generate those points from the IFC file. Therefore, a useful addition to the obstacle maps would be the centroid of each defined space in the building. The coordinates can be determined by simply calculating the centroid of the meshes of each IfcSpace element. Red markers are then added to the obstacle map at those coordinates, providing a central position for the ARS to stop and perform a general scan, or take images of a given space.

## Experiment

As a proof of concept, this section provides an overview of how the proposed methodology works by using a sample IFC file (Figure 2) to generate the JSON file and obstacle maps.

### IFC file

The chosen open-source IFC file was obtained from ifcwiki.org (*IFC Wiki* Accessed: 12/06/2021) and was authored in ArchiCAD20 (*ArchiCAD 20* Accessed: 12/06/2021). The file uses the IFC4 schema. The building represented by the IFC is a 5-story (including a basement) institutional-type building with a footprint of 850 $m^2$. The floors are divided into different spaces/rooms (around 20 spaces on average per floor) and contain furniture (tables, chairs) and doors of different operation types, such as single-hinged swing left or right doors, double doors and revolving doors. Figure 6 shows a screenshot of the IFC model. A standard floor plan of the ground floor is shown in Figure 7. This type of floor plan can be easily obtained from any BIM software when importing an IFC file. One of the limitations of this type of floor plan, when considered for ARS navigation, is that different elements cannot be easily differentiated. Obstacles are also not represented based on the height of the ARS; instead, the whole elements are shown.

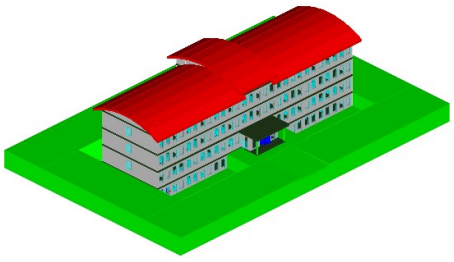

*Figure 6: Graphical representation of the IFC being studied*

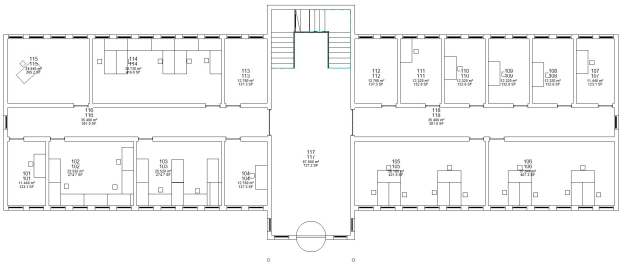

*Figure 7: Floor plan of Ground Level*

### Parsing and Storing Data

The data in the IFC file was parsed as detailed in the Methodology section. The parsed, navigation-specific, data was then stored in a JSON file so that it can be later accessed. The JSON file was inspected and was found to be as expected.

### Mesh Generation and Obstacle Maps

The unified mesh generated from the IFC file (shown in Figure 8) is as expected and matches the IFC model. As the model contained five stories, five different obstacle maps were generated. Figure 9 shows the obstacle map generated for the basement of the building. Obstacles are shown in black, doors in green and stairs in yellow.

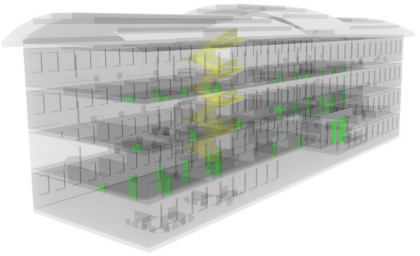

*Figure 8: Unified mesh of the IFC*

### Results Discussion

The generated obstacle maps match the information obtained from the IFC file. Using the results from the basement of the building, information from the IFC shows the presence of 16 doors and one stair. Sixteen doors (colored in green) and one stair (in yellow) can be counted from the obstacle map generated. The semantic information provided by the obstacle map is therefore validated. Overall, the obstacle map can give an added layer of semantic information over a standard floor plan. The waypoints (red squares) in Figure 9 accurately represent the centroid of each defined space in the building. However, it should be mentioned that during construction, the disposition of the actual layout of the construction site might not be exactly as shown in the obstacle map, for example, some spaces might not be completed yet, and temporary obstacles (e.g., scaffolding) might be present. In any case, the generated waypoints could provide a reliable position for the location and navigation of the ARS during its autonomous tasks.

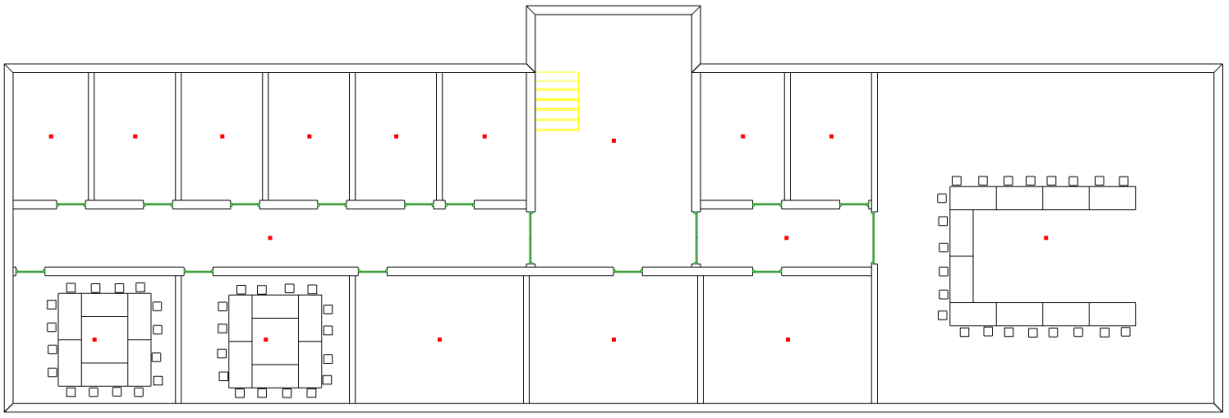

*Figure 9: Obstacle map of the basement from the IFC for the sample building. Different openings between the spaces (doors) in green. Vertical openings between floors (stairs) in yellow. Centroid of each space (waypoints) in red.*

## Conclusion, Limitations and Future Work

The need for automation in the AEC industry is increasing. Tedious and repetitive tasks, such as progress monitoring, would really benefit from automation. ARSs used in automated applications often only rely on onboard sensors for navigation. With the advent of BIM, it is now possible to make building information available to autonomous systems before deployment on-site. To make this process more efficient, it is worthwhile to automate the extraction of information from BIM models as manual extraction is often time-consuming and requires in-depth knowledge of the BIM tool used. This paper proposes a method for bridging the gap between BIM and ARS. By relying on the IFC file, relevant information was extracted by an algorithm to produce data that can be more easily understood by an ARS. Navigation was the main scope of the paper. As such, obstacle maps, loaded with semantic information in the form of color-coding, were generated from the IFC. Other relevant details such as door operation type were extracted and stored in a JSON file for easy access.

The proposed implementation can be improved. Firstly, the level of detail of the color-coded obstacle maps could be increased by considering even more semantic information present in the IFC. Currently, only three types of elements are considered. Secondly, the implementation could be adapted for applications where the building is not fully constructed, such as progress monitoring. This could be done by adding the scheduled tasks to the IFC and generating the obstacle map based on whether a certain element was already built or not. This would allow for the obstacle maps to be dynamically updated to reflect the current state of the building more accurately. This schedule information could also be stored in the JSON file to be accessed by the ARS to determine the as-planned construction progress of a certain element. The waypoint generation should also account for non-rectangular room shapes, where more than one point is needed to fully scan the room due to wall occlusions. Future work by the authors will address such improvements.

## References


*ArchiCAD 20* (Accessed: 12/06/2021), On-line: https://graphisoft.com/solutions/products/archicad.

Dawson-Haggerty et al. (Accessed: 12/06/2021), 'trimesh', On-line: https://trimsh.org/.

de Koning, R., Torta, E., Hendrikx, B., Pauwels, P. & van de Molengraft, M. (2021), 'Queries on semantic building digital twins for robot navigation'.

Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numerische Mathematik* **1**(1), 269271.

Dimyadi, J., Spearpoint, M. & Amor, R. (2008), Sharing building information using the IFC data model for FDS fire simulation, *in* 'Fire Safety Science', pp. 1329–1340.

*Gazebo Simulator* (Accessed: 12/06/2021), On-line: http://gazebosim.org/.

Gerold, F. (Accessed: 12/06/2021), 'IFC++', On-line: https://ifcquery.com/.

Hamieh, A., Makhlouf, A. B., Louhichi, B. & Deneux, D. (2020), 'A BIM-based method to plan indoor paths', *Automation in Construction* **113**.

Hart, P., Nilsson, N. & Raphael, B. (1968), 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths', *IEEE Transactions on Systems Science and Cybernetics* **4**(2), 100107.

*IfcDoorTypeOperationEnum* (Accessed: 12/06/2021), On-line: https://standards.buildingsmart.org/IFC/DEV/IFC4_2/FINAL/HTML/schema/ifcsharedbldgelements/lexical/ifcdoortypeoperationenum.htm.

*IFC Wiki* (Accessed: 12/06/2021), On-line: https://www.ifcwiki.org/index.php?title=IFC_Wiki.

*Introducing JSON* (Accessed: 12/06/2021), On-line: https://www.json.org/json-en.html.

Karimi, S., Iordanova, I. & St-Onge, D. (2021), 'An ontology-based approach to data exchanges for robot navigation on construction sites', *arXiv:2104.10239* .

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

Krijnen, T. (Accessed: 12/06/2021), 'IfcOpenShell', On-line: http://www.ifcopenshell.org/.

Lin, Y. H., Liu, Y. S., Gao, G., Han, X. G., Lai, C. Y. & Gu, M. (2013), 'The IFC-based path planning for 3D indoor spaces', *Advanced Engineering Informatics* **27**, 189–205.

Liu, L., Li, B., Zlatanova, S. & van Oosterom, P. (2021), 'Indoor navigation supported by the industry foundation classes (ifc): A survey', *Automation in Construction* **121**, 103436.

*MATLAB Robotics Toolbox* (Accessed: 12/06/2021), On-line: https://www.mathworks.com/products/robotics.html.

*OpenCASCADE* (Accessed: 12/06/2021), On-line: https://www.opencascade.com/.

*Robot Operating System* (Accessed: 12/06/2021), On-line: https://www.ros.org/.

*SMART-NYUAD/ifcobstaclemap* (Accessed: 12/06/2021), On-line: https://github.com/SMART-NYUAD/ifcobstaclemap.

Song, C., Wang, K. & Cheng, J. C. P. (2020), BIM-Aided Scanning Path Planning for Autonomous Surveillance UAVs with LiDAR, *in* 'Proceedings of the 37th International Symposium on Automation and Robotics in Construction (ISARC)', Vol. 4, International Association for Automation and Robotics in Construction (IAARC), pp. 1195–1202.

Stentz, A. (1994), 'Optimal and efficient path planning for partially-known environments', *Proceedings of the 1994 IEEE International Conference on Robotics and Automation* .

Sullivan, C. B. & Kaszynski, A. (2019), 'PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK)', *Journal of Open Source Software* **4**(37), 1450.
**URL:** *https://doi.org/10.21105/joss.01450*

Tobiá, P. (2015), 'An Investigation into the Possibilities of BIM and GIS Cooperation and Utilization of GIS in the BIM Process', *Geoinformatics FCE CTU* **14**, 65.

*Unity* (Accessed: 12/06/2021), On-line: https://docs.unity3d.com/Manual/Navigation.html.

Wang, R. & Tang, Y. (2021), Research on Parsing and Storage of BIM Information Based on IFC Standard, *in* 'IOP Conference Series: Earth and Environmental Science', Vol. 643, IOP Publishing Ltd.

Xu, M., Wei, S., Zlatanova, S. & Zhang, R. (2017), BIM-BASED INDOOR PATH PLANNING CONSIDERING OBSTACLES, *in* 'ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences', Vol. 4, Copernicus GmbH, pp. 417–423.