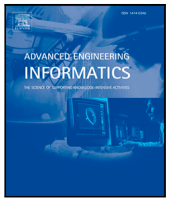
Live semantic data from building digital twins for robot navigation

Advanced Engineering Informatics 56 (2023) 101959

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei



Full length article

Live semantic data from building digital twins for robot navigation: Overview of data transfer methods

Pieter Pauwels [a,*], Rens de Koning [b], Bob Hendrikx [b], Elena Torta [b]

[a] Department of the Built Environment, Eindhoven University of Technology, The Netherlands
[b] Faculty of Mechanical Engineering, Eindhoven University of Technology, The Netherlands

ARTICLE INFO

Dataset link: https://pi.pauwel.be/tue/ADVEI2023_robotnavigation_additionaldata.html

Keywords:
Linked data
Semantics
Robot navigation
3D geometry
Indoor navigation
Buildings

ABSTRACT

Increasing reliance on automation and robotization presents great opportunities to improve the management of construction sites as well as existing buildings. Crucial in the use of robots in a built environment is their capacity to locate themselves and navigate as autonomously as possible. Robots often rely on planar and 3D laser scanners for that purpose, and building information models (BIM) are seldom used, for a number of reasons, namely their unreliability, unavailability, and mismatch with localization algorithms used in robots. However, while BIM models are becoming increasingly reliable and more commonly available in more standard data formats (JSON, XML, RDF), they become more promising and reliable resources for localization and indoor navigation, in particular in the more static types of existing infrastructure (existing buildings). In this article, we specifically investigate to what extent and how such building data can be used for such robot navigation. Data flows are built from BIM model to local repository and further to the robot, making use of graph data models (RDF) and JSON data formats. The local repository can hereby be considered to be a digital twin of the real-world building. Navigation on the basis of a BIM model is tested in a real world environment (university building) using a standard robot navigation technology stack. We conclude that it is possible to rely on BIM data and we outline different data flows from BIM model to digital twin and to robot. Future work can focus on (1) making building data models more reliable and standard (modelling guidelines and robot world model), (2) improving the ways in which building features in the digital building model can be recognized in 3D point clouds observed by the robots, and (3) investigating possibilities to update the BIM model based on robot feedback.

1. Introduction

1.1. A robotized world

The built environment has been digitizing rapidly over the past few years. This includes both technologies for the engineering and construction phase, and the operational real estate phase. In both cases, the use of robots to automate and take over tasks from humans is increasing. Most commonly, robots are considered for taking over (1) repeated tasks that can easily be automated (e.g. order picking in warehouses, cleaning robots, site inspection, welding and assembly), and (2) critical and life-threatening tasks (e.g. bomb squad, emergency support). As a result, the future of technology in Architecture, Engineering and Construction (AEC) and operational maintenance (OM) includes a much more robotized world, which is clearly proposed and outlined by Bock [1].

Further to the use of remote-controlled devices and the above-mentioned 'controlled environments', the future is likely to see autonomous robots deployed more and more in buildings shared with humans, such as hospitals, schools and warehouses. There is a variety of tasks that robots can perform in such environments, for example medicine delivery in hospitals [2], cleaning [3], healthcare support in contaminated zones, or order picking in warehouses [4].

In all of these scenarios, robots need to navigate to different places to complete their tasks. Meals need to be delivered to different rooms and items need to be picked from different shelves. Many of such more advanced cases of robot localization and navigation [5] are constantly changing, uncertain, and hazardous. It is therefore important for robot navigation to be able to rely on as many inputs as possible for localization as well as navigation, including a connection to a live state of its environment (digital twin of buildings and surroundings).

* Corresponding author.
E-mail addresses: p.pauwels@tue.nl (P. Pauwels), r.w.d.koning@student.tue.nl (R. de Koning), r.w.m.hendrikx@tue.nl (B. Hendrikx), e.torta@tue.nl (E. Torta).

https://doi.org/10.1016/j.aei.2023.101959
Received 11 September 2022; Received in revised form 21 February 2023; Accepted 21 March 2023
Available online 5 April 2023

1.2. Maps for robot navigation

Robot navigation refers to the ability of a robot to autonomously move from an initial position to a target position while avoiding obstacles on the way [5]. In order to navigate, robots rely on several input data sources. Data sources used by robots typically include, in addition to the telemetric data obtained by the robot, 2D [6] or 3D [7] occupancy grid maps, or user-specified semantic and metric maps such as OpenStreetMap (OSM) for indoor navigation [8]. These maps are critical for robot localization and navigation, as they form environmental descriptions in the form of 2D or 3D maps. Such maps enable different activities related to navigation such as localization [9], path planning [10] and, when updated real-time, obstacle avoidance [5]. The below two sections provide more detail about such maps, how they are typically created, and how they are used.

1.2.1. SLAM-based maps

There are two mainstreams approaches for creating such maps. The first one assumes that the robot has *no prior knowledge of the environment*: a map is created from onboard sensor data automatically, while simultaneously localizing in this map (i.e. SLAM: Simultaneous Localization and Mapping). Many references to several implementations of this SLAM technique can be found in Panigrahi and Bisoy [11]. This process can be done automatically by the robot (exploration) or by remote operation. Depending on the sensors available on the robot, the map can take the form of a 2D occupancy grid (i.e., for a 2D LiDaR sensor [6]) or a 3D occupancy grid (i.e., for a 3D LiDaR [7] or RGB-D camera [12]). More recent approaches also allow the inclusion of semantic features in such maps like positions of chairs or tables [13,14].

The advantage of using SLAM approaches is that the robot is able to derive the map autonomously and does not require humans to specify relevant environmental configurations. One of the downsides of this approach is that the initial exploration phase to create the map is very time-consuming especially for large buildings. Also, there is the chance that movable elements such as people or furniture become a structural part of the map which would make the map incorrect once they move their position. A considerable amount of cleaning needs to happen before the map can be effectively used, which is an expensive process.

1.2.2. Maps with contextualized semantics

The second approach for robot localization and navigation *relies on a map that is specified beforehand by users*. Such a map has both semantic and metric information. As an example, Naik et al. [8] extended OpenStreetMap (OSM) with robot-specific semantic and geometric information such as doors, corridors and elevators. Point cloud data is much less present in these maps, and localization and navigation relies on a manually created map with well-defined semantic features (edges, corners, etc.). This approach is explained in more detail in Crespo et al. [13], showing how a semantic map can be made available to the robot, including clearly defined concepts like ‘kitchen’, ‘chair’, ‘table’, ‘door’, etc. This enables the robot to take into account semantically meaningful contextual information if needed, in addition to the purely sensory (odometric, visual) data such as point clouds, imagery, and motion sensor data.

In contrast to the automatically generated SLAM-based maps, maps generated manually by such users or ‘operators’ can be more reliable in certain cases, depending on the operator’s sense of precision and completeness. Operators are in charge of creating the correct map, which is much closer to a traditional engineering workflow. They can, for example, correctly label elements such as doors or windows, as well as areas that are available or recommended for robot navigation. Operators can also assign material properties in an easy and more reliable way (manual assignment), which is especially useful for materials that are hard to recognize for robots (e.g. glass, reflecting metals, etc.). As such, they can, for example, identify and label glass surfaces which are notoriously difficult to perceive by common robotic sensors such as Light Detection and Ranging (LiDaR) scanners. This is, in some cases, more efficient than relying on the SLAM approach for localization. However, it requires specialized manual work in creating the semantic map and representing all elements needed for navigation. Furthermore, some environmental elements that are important for computing obstacle-free paths may not be included such as chairs or tables. They need to be added to the map by the robot in real-time. At that point, SLAM-based approaches for simultaneous localization and mapping can become of use again.

1.3. Potential for the use of 3D BIM data

1.3.1. BIM data as live data sets

Instead of relying on manually created semantic maps (Section 1.2.2), this research aims to find ways to create semantic maps for robot navigation starting from Building Information Models (BIM models). BIM models are dominating construction industry [15,16]. In particular large buildings and infrastructure are seldom built without a BIM model. These are detailed 3D semantic models describing the building in detail, typically handed over to the building owner or facility manager both in the form of a native BIM model as well as a vendor-neutral Industry Foundation Classes (IFC) file [17]. Several research initiatives have looked into using these BIM models for Facility Management (FM) [18–20]. In reality, however, such BIM models or IFC models are rarely used in the operational stage of a building, except as a starting point for building a brand new and often 2D-oriented Building Management System (BMS) [21,22].

This situation is gradually changing with the many efforts in the AEC and FM industries to make building data (often BIM-based) more easily available as live data sets rather than static files [23]. Building data is more often transferred and made available on servers (local servers as well as external cloud-based services), in formats that are easier to use than the proprietary commercial formats (e.g. RVT) and the EXPRESS-based IFC format. While the traditional AEC industry tends to operate with closed BIM models in Revit as well as IFC files in the EXPRESS language, several of the latest efforts aim to disclose building data as RDF graphs, JSON data, XML snippets, and similar. Reviews and examples of how these technologies are applied to the building industry, in particular the RDF graphs and JSON formats, are available in several articles [23–32]. As a result of these efforts, building data from BIM modelling environments can be more easily made available as live data streams or a ‘live digital twin of an operational building’ [22,32,33].

1.3.2. BIM data in BMS systems for smart buildings

The Smart Building domain is a clear case of how such BIM-based semantic data are becoming increasingly useful. A large share of the smart building domain is adopting JSON-based and RDF-based data formats in their in-house data management for BMS systems and Building Automation and Control (BAC) systems, as well as in their communication with devices in the building (IoT streams using MQTT protocol for JSON-based data transfer). As such, data-driven smart buildings are created [24,29,30], in which building data is available in a data format that is easier to combine with data from other systems. In these cases, RDF graphs of the building (Linked Building Data[1] graphs, BOT, RealEstateCore, BRICK, DogOnt, etc.) are typically used to create the semantic representation of the building [23]. These RDF graphs are meant to be managed in a modular fashion (= linked building data approach), where each partial graph covers a specific domain aspect of the building (e.g. topology in BOT, appliances in SAREF, home automation in DogOnt, sensor readings in BRICK and SSN, etc.) [27]. These representations cover the topological structure of the building

[1] https://www.w3.org/community/lbd/

(space, entity, storey, zone, etc.) as well as element classifications (wall, door, floor) and element properties (material, width, height, thermal transmittance) [23]. The JSON format is used in smart building applications for the same purpose (topology, element classes, properties), but even more so for transmitting sensor values and streams of data point values (temperatures, sensor readings, etc.) [22]. Because BMS and BAC systems need (near to) real-time operation, they rely on direct communication over network protocols (TCP, HTTP, MQTT), which commonly happens using the JSON file format.

1.3.3. BIM data for robot navigation with semantic maps

With this possibility of managing building data as RDF graphs (linked building data graphs — LBD graphs) in combination with JSON-based data streams, opportunities also arise in the context of robot navigation, in particular the robot navigation case that relies on semantic maps (Section 1.2.2). The available building data (BIM models as LBD graphs and JSON data streams) are namely very similar to the semantic maps and semantic representations used in the second approach for robot navigation that relies currently mostly on manually creating these maps (Section 1.2.2). Instead of computing the robot pose (localization) by comparing input LiDaR readings with the internally computed SLAM grid map, the hypothesis is that direct queries can be made from robot to the live BIM-based digital twin of the building, which is available as RDF graphs and JSON streams in the local servers.

Hence, 3D BIM data has big potential to informing robot navigation, in particular when needing to create such semantic maps. Similar investigations have been done, for example by Kim et al. [34], Kim and Peavy [35], and Karimi et al. [36]. In Kim et al. [34] and Kim and Peavy [35], IFC files of buildings are converted to SDF and URDF formats. These are world model representations that are used in the Gazebo simulator in the case of SDF, and in a Robot Operation System (ROS) in the case of URDF. URDF is thus a more generic version of the SDF file format. Both in the case of URDF and SDF, a robot world model is generated, that can be used by robots for localization and navigation. The IFC building information is hereby as good as possible embedded in the URDF model and available for use. The amount of available information is then limited to the amount of data that managed to be transferred from the IFC file to the URDF file. In the case of Karimi et al. [36], a similar integration between BIM and ROS is targeted, including also Geospatial Information System (GIS) data. Karimi et al. [36] aim for interoperability between BIM, GIS, and ROS, and rely on semantic web technologies to achieve it. Namely, a novel Building Information Robotic System (BIRS) ontology is developed, and data is transferred from BIM models to ROS. This is mainly achieved eventually by creating an XML file that contains the needed information for the topological map (grid map). From within this XML file, a connection can be made to the RDF graph of the building model to retrieve more contextual information.

1.3.4. Overview of data transfer routines

The current study is most similar to the work of Karimi et al. [36], in the sense that it targets the same connection between robot and live BIM data, including the creation of a topological grid map (metric and semantic), and aims to maintain a reference to live building data. However, the works by Kim et al. [34], Kim and Peavy [35], and Karimi et al. [36] also clearly show that several data transfer routines are available; and the methods explained and used in these works are not the only available ones. In fact, the data transfer method used in Kim et al. [34] and Kim and Peavy [35] is limited to a file translation service from IFC to URDF, leaving out further connections at that point. The work by Karimi et al. [36] relies significantly on a custom-built ontology, instead of using state of the art vocabularies for building data, and the data transfer method relies on Dynamo scripting and custom scripting to an XML file, which both generate significant limitations in further use and overall scalability. Hence, the current study extends these works by investigating also alternative data transfer routines instead of relying on just one data transfer method. As several data transfer routines are possible, it is relevant to find the data transfer method that has high enough speed, completeness, potential and ease of use. In that regard, the current study aims to investigate the use of BIM data that is present for existing buildings (e.g. also to support smart building use cases — see) also for the case of robotic navigation. In particular, the current paper will investigate the works on the OWL ontology for IFC [37], JSON versions of IFC[2,3] [31], Linked Building Data (LBD) graphs, IndoorGML[4] [38] and other web-oriented formats [31], such as JSON-LD [39].

1.4. Scope and outline

In this article, the outlined building data models are investigated for their potential to enhance robot localization and navigation, including available data transfer routines. We will hereby scope almost entirely on the data formats and flows that can be followed to exchange data from a BIM model to a navigating robot. This investigation hereby focuses on finding those data transfer routines and those data models that achieve (1) feasibility, (2) scalability, (3) extensibility at robot runtime, and (4) reliability. These criteria are inspired by the current situation in research and industry as explained above, and they will be developed in more detail in a requirements and criteria section (start of Section 3 and Section 3.4).

After a short literature study in Section 2, including a number of reference formats and approaches often used for robot navigation, we outline the most promising methodologies available to stream data from BIM models to robots, and we include a number of reference criteria (Section 3). In Section 4, we evaluate the most promising of these data flow mechanisms, from BIM software to robot, using an example university building at Eindhoven University of Technology. Each of these data flow mechanisms is evaluated against the available criteria in Section 5. Conclusions and future work findings are outlined in Section 6.

2. State of the art for robot world models and BIM

2.1. Robot navigation

2.1.1. Pose estimation

The term ‘navigation’ refers to a combination of localization and motion planning. Both aspects have been extensively researched both in isolation and as combined *navigation stacks*. Localization is concerned with determining the *pose* of the robot in a map, based on both geometric environmental data from the robot’s sensors and *ego-motion* sensor data from inertial measurement units (IMU) or wheel encoder odometry. A distinction can be made between global methods that do not rely on an accurate initial pose estimate and *tracking* methods that do. We focus on the latter, assuming that an estimate is available (see Introduction).

Many successful localization methods employ a probabilistic approach to deal with the inherent uncertainty in the sensor data, map and ego-motion estimated [40]. An overview of common methods is given in Panigrahi and Bisoy [41]. Two popular classes of methods are *scan matching* methods that determine the local *pose* difference that aligns the sensor data to the map, and particle filters, that use a set of weighted particles to represent a probability distribution over the robot pose.

[2] https://github.com/buildingSMART/ifcJSON
[3] https://ifcjs.github.io/info/
[4] https://www.indoorgml.net/

2.1.2. World representations

Both localization methods can be used with various world representations. Cell decomposition methods, such as occupancy grid maps [42], are a popular world modelling choice, providing a grid of free and occupied space. The data formats to represent grid maps depend on the implementation choices of the navigation algorithm at hand. A popular representation is a multi-dimensional array in which each element of the array represents a cell of the grid map and the value assigned to the element it represents. This value typically captures the status of the cell, e.g., 'occupied' or 'free' when the map is binary.

These grid maps also connect easily to motion planners, that can plan a global path through free space. For this purpose, the grid map is transformed into a graph where each cell represents a node. Edges connect nodes representing cells that are adjacent to each other. The computed global path (i.e. the sequence of cells to visit) is then used as input for a local planner that is concerned with following the path while avoiding local obstacles. World models for local planners rely on cost maps which are a form of grid maps in which cell values are updated dynamically based on sensory inputs. In practice, each element of the array representing the grid map can have different discrete values representing, for example, free space, occupied space, space not yet perceived, etc. An overview of motion planning methods can be found in Elbanhawi and Simic [43] and Patle et al. [10].

2.1.3. From SLAM-generated maps to semantic maps

The creation of these grid maps for robot navigation is often achieved by the application of SLAM methods. The robot is often tele-operated to explore an unknown environment. While moving, subsequent sensors' scans (e.g., from onboard LiDaR scanners) are read and overlaid with each other to incrementally create the contour of the environment and its objects. The overlaid contours can then easily be converted to a grid map. Popular implementations of SLAM algorithms are Cartographer [44], GMapping [45] and HectorSlam [46] (see Yagfarov et al. [6] for a comparative study).

Note that maps created with state-of-the-art SLAM methods are of a geometric nature only, because they represent, in a discretized way, the contour of the (building) elements of an environment with little to no semantic details. Creating such maps based on BIM data in principle only requires computing where geometry is present, and where it is not present, to then create the required matrix of grid cells. In a search for semantically contextualized maps, adding semantics to these maps is a very active area of research (see Sections 1.2.1 and 1.2.2).

For both localization and navigation, the possibility of generating reliable semantic detections from point clouds and cameras using convolutional neural networks (CNN) has recently led to much original research on this topic [47]. Semantics can be used both for making detections and map associations more robustly (e.g., Himstedt and Maehle [48]), and for configuring or changing the behaviour of the robot based on semantic knowledge [13]. For example, semantic representations make interactions with human operators for navigation tasks much easier (e.g., "go to room 6"), while taking semantics into account during motion (e.g., steer away from glass and doors for safety).

A distinction can be made between (1) methods that merely label geometric sensor or occupancy data and (2) object-oriented methods that represent instances and their semantic relations. The former can be achieved using computer vision and semantic segmentation methods, yet the outcome is labelled data rather than semantically meaningful and interconnected data. The added value of this approach is therefore limited. The point of the latter is that such semantically rich data sets become available and can be queried at will. BIM models can be used as a source for this geometric-semantic instance knowledge, without relying on sensors to reliably perceive it (computer vision, object detection, semantic segmentation).

Semantic maps still need to align to geometric maps, in our case grid maps, in order to be of use for robot navigation. Robot localization and navigation namely starts from the match between sensory data (e.g. LiDaR data) and this grid map. Only in a second stage, further semantic data can be consulted, starting from the grid map identifier. Features in this grid map thus need to be related to features in the semantic model. This is also the method followed by Karimi et al. [36], for example, which allows querying building data after it is recognized on the grid map.

2.2. BIM-based building data

BIM data is most typically available in IFC format. Although this data model is originally built using the EXPRESS information modelling language [49], it is now built mainly using a Unified Modelling Language (UML) Class Diagram.[5] The resulting UML diagram can be used to generate EXPRESS schema, XSD schema, JSON schema, and OWL ontology. Further to the IFC approach, other data modelling approaches for building data are available. This includes primarily the Linked Building Data (LBD) approach [23,25–28] and Smart Buildings data models [24,29,30]. The below sections document these approaches one by one, while more information is available elsewhere [23]. We hereby focus specifically on the way in which they are *able to represent semantic features (walls, doors, rooms, windows, profiles, columns) together with a 2D or 3D geometric representation that can be used to create 2D or 3D grid maps for localization and navigation purposes in an interactive manner*.

2.2.1. Industry Foundation Classes (IFC)

IFC is the most well-known standard in the AEC industry and allows to define a building model fully in a number of formats, namely RDF, SPF, XML, and JSON. In all cases, the same UML diagram sits at the core of the model. This leads to complete building models that represent all 3D geometry, 2D geometry, object types, and object properties. The SPF version of IFC is the most prevalent and commonly supported version of IFC. This is a data model based on STEP that can be accessed through a number of parsers. The best option to parse an IFC-SPF file is to use the Standard Data Access Interface (SDAI) library that comes with the SPF and EXPRESS language.[6] SDAI connects mainly with C and C++ libraries and is not that commonly used in the AEC industry software. An alternative often used library to access IFC-SPF data is the IfcOpenShell library that is available in C++ and has a Python binding.[7] Other custom IFC parsers are available as well.

Next to the SPF version of IFC, commonly available formats are JSON, XML, and RDF. The RDF format for IFC has been standardized into the ifcOWL ontology, that is regularly available.[8] It has been shown, however, that the choices made in creating this OWL ontology for IFC have lead to verbose RDF graphs that have plenty of content that is not commonly used or easily usable (e.g. geometry and ordered lists in general [50,51]). Nevertheless, IFC-RDF data can readily be queried using the SPARQL[9] query language [52] and it can be parsed easily in one of the diverse RDF coding libraries available (e.g. rdflib, dotnetrdf, Jena, RedLand RDF, rdf.js, etc.).

The XML version of IFC has been standardized long time ago, and this has lead to an XSD schema[10] that can be used to validate ifcXML files and structure them in an agreed form. In practice, many ifcXML files, however, diverge from this standard XML format, and make them unpredictable and then difficult to parse or query [31], no matter how many XML libraries are available. The JSON version of IFC has no standardized format. A number of different JSON encodings are available, sometimes aiming to cover the entire schema (e.g. ifcJSON[11])

[5] https://github.com/buildingSMART/IFC4.3.x-development
[6] https://www.steptools.com/stds/step/sdai.html
[7] https://ifcopenshell.org/
[8] https://standards.buildingsmart.org/IFC/DEV/IFC4/ADD2_TC1/OWL
[9] https://www.w3.org/TR/sparql11-query/
[10] https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/
[11] https://github.com/buildingSMART/ifcJSON

while others are covering specific parts (e.g. IFC.js[12]). Some JSON files follow a full tree structure (e.g. Listing 1, snippet from buildingSMART ifcJSON project[13]), while others focus much more on flat lists of objects.

```
{
  "type": "IfcDistributionPort",
  "globalId": "1f45e60b-c79b-4422-977b-82adb1b73462",
  "ownerHistory": {
    "type": "IfcOwnerHistory",
    "ref": "e4980768-d888-49ec-8297-0968dbd994ca"
  },
  "name": "Inlet",
  "objectPlacement": {
    "type": "IfcLocalPlacement",
    "placementRelTo": {
      "type": "IfcLocalPlacement",
      "relativePlacement": {
        "type": "IfcAxis2Placement3D",
        "location": {
          "type": "IfcCartesianPoint",
          "coordinates": [
            0.0,
            0.0,
            0.0
          ]
        }
      }
    },
    "relativePlacement": {
      "type": "IfcAxis2Placement3D",
      "location": {
        "type": "IfcCartesianPoint",
        "coordinates": [
          12.0,
          12.0,
          4.0
        ]
      }
    }
  },
  "flowDirection": "SINK",
  "predefinedType": "DUCT",
  "systemType": "AIRCONDITIONING",
  "nests": [
    {
      "type": "IfcRelNests",
      "ref": "d295c148-65ff-4aa4-82ed-726e36ee8946"
    }
  ],
  "connectedFrom": [
    {
      "type": "IfcRelConnectsPorts",
      "ref": "229fc988-e700-418c-93f0-52150398730e"
    }
  ]
},
```

Listing 1: An inlet distribution port in ifcJSON

2.2.2. Linked Building Data (LBD)

Linked Building Data (LBD) can be defined as sets of domain-specific RDF graphs that together form one large graph representing a piece of built infrastructure, e.g. a building. While an LBD graph can contain non-RDF data, it is usually understood to have primarily RDF data. The vocabularies governing these LBD graphs are built and standardized under the Linked Building Data Community Group of the World Wide Web Consortium (W3C) initiative.[14] This initiative started to a large extent from the ambition to make data like the IFC natively available in RDF and OWL, while still leading to high-quality RDF graphs. The main ambitions here were to create a set of modular, interlinked, simple enough, and extensible ontologies. This has lead to an ecosystem of ontologies [27], including the BOT ontology,[15] OPM ontology,[16] BEO[17] and MEP[18] ontologies, and so forth, which can together be used to represent building data. The example in Listing 2 shows how a building and an air terminal can be expressed in an LBD-compliant RDF graph.

```
inst:building_326
  a bot:Building ;
  bot:hasGuid "6d313a7d-0bf7-4f08-947f-80599436ab30"^^xsd:
    string ;
  props:hasCompressedGuid "1jCJfz2$TF29H$W5cKDgim"^^xsd:
    string .

inst:airTerminal_331
  a bot:Element ;
  a mep:AirTerminal__DIFFUSER ;
  bot:hasGuid "703c0f93-955f-4bdc-a8c0-41e62f30f3bb"^^xsd:
    string ;
  props:hasCompressedGuid "1mF0_JbLzBtAZ0GUOlCFEx"^^xsd:
    string ;
  props:tag ""^^xsd:string ;
  props:modelReference "1234"^^xsd:string ;
  props:modelLabel "Ceiling Diffuser"^^xsd:string ;
  props:manufacturer "Acme"^^xsd:string ;
  props:productionYear "2011"^^xsd:string ;
```

Listing 2: A building and air terminal in TTL.

While converters are available to transform IFC-SPF files into LBD graphs,[19,20,21] the more interesting option consists of exporting LBD data directly using a dedicated exporter script that can be embedded as an add-in to software.[22] Through those codes,[23] it is in fact possible to maintain a live database (triple store) with the LBD representation of the building model while it is being modelled in BIM software like Revit. To enable such functionality, one simply has to execute this plugin at regular intervals (e.g. every minute) and send updates to an underlying live database in the common data environment of the engineering firm [53]. While this is no standard practice, it is possible and this can lead to a live digital twin that is accessible through available database interfaces (e.g. SPARQL endpoint, or an API with access control) - see for example [33]. Finally, LBD data sets can be created *from scratch* as well, using custom or generally available software, e.g. Python + rdflib + OntoText GraphDB, something that is much more difficult with a single ifcOWL ontology because of the large complexity and large number of constraints that needs to be met to be compliant with the IFC standard. This is one of the several reasons why the use of more modular LBD graphs is recommended over aiming to build ifcOWL graphs [23,27,28,53,54].

2.2.3. Smart buildings data models

In addition to IFC and LBD data, recent smart buildings research has looked into a number of representation schemas as well. This includes RealEstateCore (REC), BRICK, Haystack, SAREF, to name the most prevalent ones. Most of these schemas or vocabularies focus on the representation of objects, building topology, and properties of objects, and lack a concrete approach for the representation of geometry, because such detailed geometry is of less importance in smart buildings.

[12] https://ifcjs.github.io/info/
[13] https://github.com/buildingSMART/ifcJSON/blob/master/Samples/IFC_4.0/BuildingSMARTSpec/air-terminal-element.json
[14] https://www.w3.org/community/lbd/
[15] https://w3c-lbd-cg.github.io/bot/
[16] https://w3c-lbd-cg.github.io/opm/
[17] https://pi.pauwel.be/voc/buildingelement
[18] https://pi.pauwel.be/voc/distributionelement
[19] https://github.com/jyrkioraskari/IFCtoLBD
[20] https://github.com/pipauwel/IFCtoLBD
[21] https://kgg.openmetrics.eu/
[22] https://github.com/MadsHolten/revit-bot-exporter
[23] https://github.com/w3c-lbd-cg/tools

As the representation of geometry is key in our work, however, we leave these ontologies out of scope.

Data according to these other ontologies (BRICK, REC, etc.) can nevertheless still be reached as part of the LBD data (see also Fierro and Pauwels [55]). LBD is in principle a network of connected data, that makes use of multiple ontologies. The LBD data could in addition to a BOT, BPO, MEP, and BEO classification, also include a REC, BRICK, SAREF or Haystack classification. This was tested with SAREF and SSN in Schneider et al. [27], and is perfectly feasible as long as there are no conflicting semantics or conflicting statements. Other examples are available as well, such as Mavrokapnidis et al. [33] and Chamari et al. [22]. In other words, the engineer and engineering software need to follow a clear data representation strategy that avoids conflicting statements. As of this point, we assume that an LBD representation of the building can easily be enriched with data according to REC, BRICK, Haystack, and SAREF ontologies, and smart buildings data can hence be easily connected as well.

3. Method for evaluating data flows from BIM to robot

Based on the above literature overview, the overall context of smart buildings, robots in slightly dynamic environments, and the promising availability of web-oriented representations of building data [31], this section looks into the use of that data. In principle, this article does not consider any use case to be out of scope. Potential use cases for robot navigation could be maintenance and inspection robots, last mile delivery robots, and robots navigation hospitals for delivery and support (see introduction). Although it is possible to target robots on construction sites (e.g. Kim et al. [56]), we specifically assume existing buildings for which a BIM model is available after the as-built handover phase. Specifically, our experiments are performed with the same dataset that was also used for the smart building research in Chamari et al. [22]. In Sections 5 and 6, a discussion will be included on the applicability of the results in practical end-user environments.

Data transfer from a BIM environment to a robot can occur in a number of methods, and they can be categorized according to a number of characteristics, as shown below.

1. Scope of standardization — broad or narrow:
   Data transfer always needs to follow certain agreements and guidelines. The wider the support for such agreement is, the wider the scope of standardization, the more scalable and feasible the data transfer method.
2. Information transfer — many or few transformation operations:
   Initial data transfer from BIM model to robot often involves data transformation operations. The more numerous and more significant these transformations are, the less feasible and scalable the method becomes.
3. Dependencies — software-neutral or software-specific:
   Data transfer is tied into existing software and therefore software-dependent, or it is software-neutral and therefore further away from the designer interface. This has impact on feasibility of the transfer method.
4. Extensibility at run-time — limited or extensible:
   Data transfer leads to certain data being available at runtime. In certain cases, this remains limited to data that was transferred (no backward connection or query access — limited); in other cases, a large set of further semantic data can potentially be obtained (extensible).
5. Reliability — outdated or up-to-date:
   As soon as data is transferred, it needs to be kept up to date, both the geometry in the grid maps and the associated semantics. This can be achieved in a number of ways, for example by keeping in sync with an updated BIM model, by having access to the live status of the building, or similar.

These characteristics define to what extent certain objectives can be achieved. Several objectives were defined in the introduction of this article, namely the achievement of (1) feasibility, (2) scalability, (3) extensibility at robot runtime, and (4) reliability. These criteria can be achieved by choosing the data transfer method with best matching characteristics, in our case primarily *broad standardization scope, few data transformation operations, software-neutral dependencies, extensible at run-time, and up-to-date*.

In the following, we list three mainstream approaches that could be identified as most promising based on the literature study in Section 2.2. For each of these transfer methods, a number of approaches is available with more fine-grained decisions and options. They will be briefly presented here, after which the most promising approaches will be tested in more detail in Section 4.

3.1. Transfer method 1: Traditional IFC-based file transfer

A first transfer method (TM1) that can easily be extracted from the list of available data handling routines is the use of the Industry Foundation Classes (IFC). This approach has a number of alternatives.

3.1.1. SPF and MVDs (TM1.1)

The standard recommended approach in this case, is to define a Model View Definition[24] (MVD) that scopes a part of the larger IFC schema to the applicable use case. This can be done using the mvdXML format, after which dedicated software development needs to take place in order to allow BIM software to export data according to that MVD. In practice, this has lead to the implementation of a number of standard MVDs in native BIM software, namely the Reference View (RV), Coordination View (CV), and Design Transfer View (DTV). It is not clear how alternative custom MVDs need to be implemented other than writing a new plug-in into software or manually tweaking some of the parameters in the pre-implemented MVDs [54]. No reliable exporters are available in native software for the XML, JSON, or RDF versions of IFC. An XML export option is available in some cases, yet follows very different XSD schemas and export flavours, leading to no reliable export option. Therefore, the result is always an IFC file in the STEP Physical File format (SPF).

As a result, this approach needs to rely on dedicated procedural code and parsers for an import by a robot. This dedicated procedural code could rely on the IfcOpenShell[25] library (C++ or Python), which is one of the most commonly used libraries for that purpose. Other dedicated software can be used, such as Blender, to load in any case the geometry and object identifiers, and transfer the geometric representation to the robot. In this case, it is advised to rely on a data transfer routine that transforms the IFC file into a robot world model, such as URDF or SDF. This data translation procedure is followed by Kim et al. [34] and Kim and Peavy [35], leading to a usable robot world model for navigation on construction sites. When relying on such a file-based operation, it may prove to be useful to also adopt the BIM Collaboration Format (BCF)[26] to communicate issues with the BIM model from robot back to BIM model.

This approach has a wide scope of standardization, as IFC is an international standard. It includes relatively few data transformations: standard IFC files are exchanged between BIM model and robot and typically require a direct manual data transfer [34,35]. This approach is software-neutral, as the IFC format can be generated and used by any BIM tool, and does not depend directly on specific software. Note that the *quality* of the IFC model does depend on the software used, as well as modelling practices and export settings. Finally, the end result

[24] https://technical.buildingsmart.org/standards/ifc/mvd/
[25] https://ifcopenshell.org/
[26] https://www.buildingsmart.org/standards/bsi-standards/bim-collaboration-format-bcf/

does not have high extensibility at robot-runtime, as the resulting robot world model is fully included into ROS without backlinks to the original model. This also makes that the robot world model becomes outdated easily.

3.1.2. IFC-RDF (TM1.2)

For the RDF version of IFC, users are typically expected to use a converter that converts the SPF version of the IFC file into an RDF graph. This can then be loaded into a triplestore after which it can be queried and/or parsed. In principle, this pipeline can be automated. It would however be easier if this functionality could be made available directly in BIM modelling tools, similar to the Open and Java Database Connectivity (ODBC and JDBC) connections that are made available by default for storing BIM data into a relational database.

The resulting RDF graph is typically large, as it includes all 3D geometric data and all ordered list descriptions for all geometry [51], which leads to lengthy descriptions that are difficult to use. Using SPARQL queries, the RDF graph can be queried for useful information, potentially using simplification rules [52,57], which can then be passed on for robot navigation.

This approach has a broad scope of standardization, as the ifcOWL ontology and corresponding RDF graphs are largely standardized. The data transfer method typically requires more data transformations than the previous method, several of which are more difficult. In this approach, the link to the original model is lost; namely, the user transforms the data to IFC-SPF and then to IFC-RDF, and does not go back to the original BIM model. Hence, the data becomes outdated quickly. However, with the use of RDF graphs, links to external data sets can easily be explored and exploited. Hence, a medium level of extensibility is achieved at robot run-time with this method. This approach is software-neutral.

3.1.3. JSON (TM1.3)

The JSON version of IFC can be used as well, although this is not standardized. Relying on JSON would make sense, because JSON is regarded as a promising data format in general, it is typically used by any state of the art web service or platform, and it is highly compatible with concurrent software libraries and development practices. Therefore, we expect that JSON can also be an important data format to represent and handle environmental description data by the robot, and the level of interoperability towards the robot is thus expected to be higher. For this data flow, it is possible again to rely on a converter that converts the SPF version of the IFC file into a JSON file.[27]

Several converters are available, some of which are embedded in existing tools. For example, it is possible to obtain a JSON representation of the building geometry by loading the IFC data into Blender and exporting it again into JSON format. However, by passing through Blender, lots of the semantics available in the IFC file are lost and need to be retrieved elsewhere. Similarly, one could rely on the IFC.js library to parse an IFC file and obtain the model in JavaScript objects (JSON). In this case, however, it is less clear how connections need to be made to the actual 3D geometry, which is less easily accessible in the IFC.js library.

This JSON transformation pipeline is promising, because of the high level of re-usability for robot navigation. The JSON format easily matches the input and output formats of many Python-based libraries used for robot navigation and localization (e.g. OpenCV, scikit, numpy, pandas, tensorflow). This makes this approach very extensible at run-time, albeit mainly with computed data (less linked data or databases). This approach has a narrow scope of standardization: only few small groups have made agreements on the JSON format for IFC, and those can only be used within those groups. The data transfer method requires a considerable number of data transformations, using any of a number of converters, making this routine less scalable and less reliable. A continuous workflow should nevertheless be reachable in JSON, and this would be a very valuable workflow. However, at the moment, the connection with an original BIM model is typically lost, and there is a considerable risk that semantic and geometric data becomes outdated. This transfer method is furthermore software-specific, as the JSON file format of IFC depends entirely on the tools that generate the data.

3.2. Transfer method 2: Live linked data server

A second transfer method relies on a live linked data server. With the term linked data, we here refer directly to 'RDF graphs'. We here consider two transfer methods, namely using an IFC server (IFC-RDF - TM2.1) and using an LBD server (TM2.2).

3.2.1. IFC server (TM2.1)

A linked data server that hosts IFC data refers to the use of a triple store (graph database) to host IFC data in RDF format. If such a transfer method is followed, then the earlier mentioned converters need to be used, from IFC to RDF. As indicated, these converters are currently typically used in a file-based approach, yet is also possible to automate the transfer from IFC to RDF in the backend of a server (only IFC needs to be uploaded). The resulting RDF graph is assumed to be stored in a triple store, as opposed to TM 1.2 that relies on RDF files.

As this approach remains close to IFC itself, there is a broadly accepted level of standardization in this approach. Furthermore, this transfer method requires few data transformations, most of which can be automated and do not require manual interventions. This method is also software-neutral and relatively extensible. Extensions can in principle easily be made, as the RDF triple store with IFC data can be extended with additional custom data (e.g. smart building data in BRICK). However, the reliance on the IFC data model as a main data format reduces the level of modularity and ease of extensibility. Finally, the reliability of the data can be quite high, on the condition that the data on the IFC server is kept up to date. As it is relatively hard to keep IFC-RDF data up to date, due to the several file transformations, this point of reliability does not score very high — data will be difficult to be kept up to date.

3.2.2. LBD server (TM2.2)

Alternative to TM2.1, which is close to TM1.1 and TM1.2, this Transfer Method TM2.2 limits entirely on the use of LBD data (RDF graphs). BIM model data is hereby directly converted into LBD graphs, and those are inserted into RDF-based triple stores (e.g. graphDB). Data can then be accessed using the API, or the open SPARQL endpoint of the triple store.

The level of standardization is less broad compared to previous IFC-based approaches, as IFC is an accepted ISO standard and industry standard, while LBD graphs are only as standard as their schema (ontology) dictates. There exists no formal ISO, CEN or W3C standard that endorses LBD graphs as an accepted standard. Nevertheless, the ontologies define how the data is to be used and there exists an informal agreement among communities about these ontologies. The number of data transformations is limited, and similar to the number of transformations needed in TM2.1, yet, several routines are available as there are several ways available to obtain an LBD graph. Hence, this number of data transformation steps could be improved. The procedure is software-neutral, as LBD graphs can be obtained starting from any software, also from scratch. This transfer method is also easily extensible: it is easy to expand the graph with additional data that may be obtained by the robot, for example. In fact, extensibility is a key feature of (modular) LBD graphs. Finally, it is quite easy to keep the modular LBD graphs up-to-date, as they are server-based, and can be updated via several routines.

[27] https://github.com/buildingSMART/ifcJSON/tree/master/file_converters

Table 1
Key characteristics of diverse data transfer methods.

	Standardization	Transfer steps	Dependencies	Extensibility	Reliability
Traditional IFC-based file transfer					
TM1.1 - SPF and MVDs	Broad	Few	Software-neutral	Limited	Outdated
TM1.2 - IFC-RDF File transfer	Broad	Many	Software-neutral	Medium	Outdated
TM1.3 - IFC-JSON File transfer	Narrow	Many	Software-dependent	High	Outdated
Live Linked Data Server					
TM2.1 - IFC Server	Broad	Few	Software-neutral	Medium	Outdated
TM2.2 - LBD Server	Medium	Medium	Software-neutral	High	Up-to-date
JSON-based web services					
TM3.1 - Retrieve web-based JSON data	Narrow	Few	Software-neutral	High	Up-to-date
TM3.2 - Connection to live web-based API	Narrow	Few	Software-dependent	High	Up-to-date

3.3. Transfer method 3: JSON-based web services

The third group of transfer methods, from BIM model to robot, is equally web-based as the linked data server approaches in TM2. In this case, the transfer method relies on JSON-based web services instead of RDF content. These are similar to TM1.3 and TM2.1 (IFC server), with the difference here that the below methods are explicitly relying on web services with JSON communication from which navigation maps are created and used in operation.

3.3.1. Retrieve web-based JSON data (TM3.1)

Transfer Method 3.1 includes a web server that returns JSON data of the BIM model. In that sense, it is very similar to TM1.3 and TM2.1. A custom server is built that responds to HTTP requests using dedicated snippets of JSON that can be consumed by the robot in the creation and updating of grid maps and their related semantics. For this method, the level of standardization is narrow, although this depends on the type of JSON data that is used and what its level of standardization is. In a custom web server, however, the JSON representation of the BIM data is typically not standard (see for example diverse versions of IFC-JSON, as well as the custom structure of IFC.JS). The number of data transformation steps is few, and they are mostly included in the web server and thus hidden for an end user. This transfer method is software-neutral and highly extensible: it is easy to add more data. Finally, the data is made available on a server, and can thus be assumed to be kept up to date.

3.3.2. Connection to live web-based API (TM3.2)

This last approach relies on JSON-based data from a web-based API. However, for TM3.2 it is assumed that this data comes straight from the BIM environment. In other words, in this transfer method from BIM model to robot, the idea is that the robot communicates directly to the web-based JSON-based API of the BIM tool(s). This could for example be a Forge-based API.[28] Alternatively, the robot could also connect directly with a Speckle stream[29] or similar. The data is hereby assumed to be not IFC-based, as it is available directly from the BIM tool.

As a direct connection is made with software, this approach is not software-neutral, but rather software-dependent. The number of data transformations is very small, because this relies only on a direct communication with existing BIM tools via their APIs — no considerable user interaction needed. The level of standardization is very narrow, namely, the data is as standardized as the API of the software that is used for communication. The data is easily extensible, as the robot can combine the data easily with data from other web services. And finally, data can be assumed to be kept up to date, as a live connection is maintained with the latest status of the BIM model. One shortcoming here is that these updates need to come in via the BIM software and thus are less easy to make.

[28] https://forge.autodesk.com/
[29] https://speckle.systems/

3.4. Requirement analysis and criteria

As a conclusion or summary to the above section, the main characteristics of the diverse transfer methods are summarized in Table 1. In order to choose for the most appropriate of the above outlined transfer methods, a number of criteria had been specified earlier in the article. Namely, in consideration of an increasingly data-driven and connected world, our targets were: *broad standardization scope, few data transformation operations, software-neutral dependencies, extensible at runtime, and up-to-date data*. In the case of robot localization and navigation in existing buildings, further expected requirements are:

1. The data represented by the BIM model needs to be *spatially accurate*.
2. The robot needs to have *semantic knowledge of spaces and connecting interfaces* (e.g., rooms, areas and doors).
3. The robot needs to be aware of *material properties* and the relation to its sensing capabilities (e.g. glass)
4. The robot needs to be able to *query this data efficiently (real time)* during operation, through an abstracted programming or query interface.

Considering these data requirements, most promising or significant data transfer methods are evaluated to be:

- TM2.2: LBD server
- TM3.1: Retrieve web-based JSON data

TM2.2 (LBD server) was picked as the better option over TM2.1, as this mainly implies a better usable set of data (LBD instead of IFC-RDF). TM2.2 is therefore expected to be better extensible and also easier to keep up to date. The second outstanding option is TM3.1, which is a connection to live web-based JSON representations of building data. This is currently also the most costly one, as it requires setting up a complete web-based backend with web services that can be queried for the right data. This is currently out of scope for implementation. As an alternative, it was decided for this article to test the less expensive TM1.3, which also tests the quality of the JSON data, in the case of IFC-JSON. Doing this test also allows us to validate whether the direct transfer of JSON data via file-based operations (TM1.3) is sufficiently feasible as well.

4. In-depth evaluation of data flows

Based on the above review, an experiment has been conducted for the two main data transfer methods, from BIM model to Robot, that were found above (TM1.3 and TM2.2). In both cases, a number of implementation choices were made during implementation (choice of tools and converters). Nevertheless, the implementation of these two data transfer methods suffices to get an indication of how promising each of these methods is.

In this section, an overview of those two implementation trajectories will be explained for one reference building, namely floor 8 and 9 of the

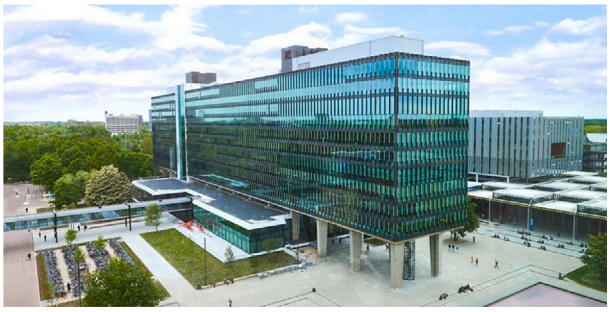


Fig. 1. The Atlas university building.

Atlas building in the campus of TU Eindhoven (Fig. 1). In this context, we use the following definitions for a number of key terms, namely spaces, interfaces and elements. These definitions are inspired by (but not identical to) their definitions as part of the BOT ontology.

- Space[30]: an empty area that is available for use, and that is bounded by either physical or virtual boundaries
- Interface[31]: a generic concept to qualify the relationship of two or more things in the world, where at least one is a building element or zone. We limit our interpretation here by considering the interface the relationship between a space and a bounding (virtual or physical) element.
- Element[32]: a construction entity with a characteristic technical function, form or position.

4.1. Route 1 - IFC-JSON file transfer

This route has a considerable number of file conversion and transfer operations, from BIM model all the way to the robot. First, the BIM model is transformed into a JSON dataset. Two alternative routes are possible, namely a conversion routine via IFC-JSON transformers, and a conversion routine via Blender SPF-import and JSON-export. We only implemented the IFC-JSON transformer. After the JSON content was created, it was transformed into JSON-LD, which can be used to build a 2D vector map for robot navigation (PNG). Finally, robot navigation takes place. The steps to convert the data from IFC to a JSON-LD property graph used for robot localization as described in Hendrikx et al. [58] are reported in Fig. 2.

4.1.1. Transformation to JSON dataset using ifcJSON converters

The original model for the Atlas building is available in Autodesk Revit. This BIM model has been exported into an IFC file representation. At the time of writing, two alternative converters were available to transform this IFC data into a JSON format, namely the transformer that is part of the IFC.JAVA project,[33] and a Python-based transformer from the IFC-JSON team.[34] The IFC.JAVA project is an object model of the IFC classes that has default Jackson (de-)serializers embedded in the code for parsing and generating IFC as XML and JSON. This converter (IFC.JAVA) follows the transformation mechanisms of this Jackson serialiser. This leads to an IFC-JSON snippet as shown in Listing 3.

```
"type":"IfcSpace",
"globalId":"15c75cfb-4a4e-404f-812a-77c1d3c20367",
"ownerHistory":"870e941a-e5b0-48d7-acff-58c01e10d40c",
"name":"81",
"isDefinedBy":[...],
"objectPlacement":{...},
"representation":{...},
"longName":"Room",
"compositionType":"ELEMENT",
"predefinedType":"SPACE",
"elevationWithFlooring":0.0,
"boundedBy":[{
    "type":"IfcRelSpaceBoundary",
    "globalId":"c1b1eb78-3e1d-4363-a7e9-9d74cfd416a5",
    "ownerHistory":"870e941a-e5b0-48d7-acff-58c01e10d40c",
    "name":"1stLevel",
    "relatingSpace":"15c75cfb-4a4e-404f-812a-77c1d3c20367",
    "relatedBuildingElement":{
        "type":"IfcWall",
        "globalId":"ff562d90-dab8-4c2e-9947-0f26d97ec225",
        "ownerHistory":"870e941a-e5b0-48d7-acff-58c01e10d40c",
        "name":"Basic Wall:Internal wall 1000mm:546521",
        "hasAssociations":[...],
        "objectType":"Basic Wall:Internal wall 1000mm:397192",
        "isTypedBy":[...],
        "isDefinedBy":[...],
        "objectPlacement":{
            "type":"IfcLocalPlacement",
            "globalId":"48ebf639-8da0-46b4-b819-868057b619fe",
            "placementRelTo":"6bd92362-5467-4606-b970-4cbdc29d163b",
            "relativePlacement":{
            "type":"IfcAxis2Placement3D",
            "location":{
                "type":"IfcCartesianPoint",
                "coordinates":[200.460100211374, -62.1876504421825, 0.0]
            },
            "axis":{
                "type":"IfcDirection",
                "directionRatios":[0.0, 0.0, 1.0]
            },
            "refDirection":{
                "type":"IfcDirection",
                "directionRatios":[-1.0, 0.0, 0.0]
            }
        }
    },
```

Listing 3: The IFC.JSON snippet generated by the IFC.JAVA library

4.1.2. Transformation into JSON-LD

JSON-LD is used by the localization algorithm in Hendrikx et al. [58]. Building features are queried from the IFC-JSON graph structure and explicitly linked to a footprint representation (e.g., lines and polygons) in global coordinates. Within the resulting JSON-LD document, the building features are linked to a representation that is marked as sensor-specific, as shown in Listing 4. This snippet shows how one of the columns in the building with tag 356071 has an additional representation of type `ObjectFeatureRepresentation`. This representation hosts a Polygon with in this case four (4) Points to mark the four corners of the rectangular column footprint. These corners can be perceived by the robot sensors.

[30] https://w3c-lbd-cg.github.io/bot#Space
[31] https://w3c-lbd-cg.github.io/bot#Interface
[32] https://w3c-lbd-cg.github.io/bot#Element
[33] https://github.com/pipauwel/IFC.JAVA
[34] https://github.com/buildingSMART/ifcJSON/tree/master/file_converters

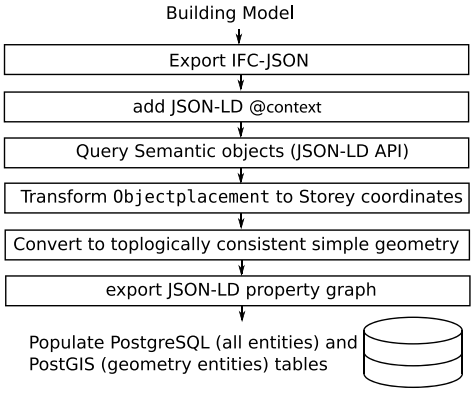


Fig. 2. Conversion from the BIM model to the representation used by the robot, stored in a database, as used in this work. This database contains both the property graph entities and the geometric entities (using PostGIS).

```
{
  "@id": "c71c8bf4-c687-4e71-88e9-85f39f04c6cd",
  "@type": "IfcColumn",
  "tag": "356071",
  "represented_by": [
    {
      "@id": "07322913f44d4e2aa4ef13f78814c2f2",
      "@type": "ObjectFeatureRepresentation",
      "perceivable_by": {
        "@id": "PlanarLidar2D"
      },
      "represented_by": [
        {
          "@id": "f8b364f1277943d0adff0e49f28d6924",
          "@type": "Polygon",
          "consists_off": [
            {"@id": "021da27a2b9d4dfdb871dfdda9c4ec5e",
              "@type": "Point"},
            ....
          ]
        }
      ]
    }
  ]
}
```

Listing 4: A JSON snippet, representing a column as perceived by a planar laser scanner

4.1.3. Floor plan generation

From the JSON-LD property graph, it is possible to generate a geometric vector map with annotated semantic features (see also Hendrikx et al. [58]). A small section of the resulting map of a floor of the building considered in this work is represented in Fig. 3. This geometric map as well as its semantic features are directly generated from the BIM model and therefore has the same geometric and semantic accuracy as this BIM model. Any geometric flaws present in this BIM model are maintained in the geometric map, which can be recognized by navigating the robot through the real building. Validating semantic mistakes in the BIM model (e.g. misclassifications) is more difficult, and has not been considered any further in this work. Geometric accuracy was validated qualitatively and visually by navigating through the actual building (see Section 5).

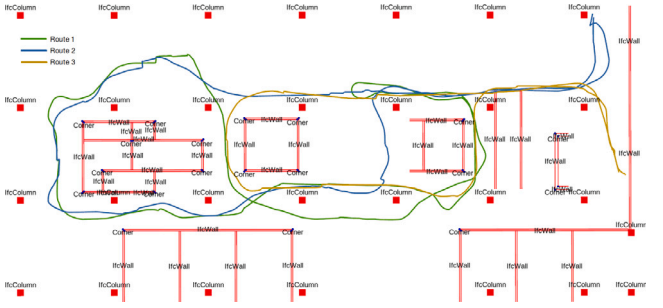


Fig. 3. The map considered in [58], as generated from the BIM model, with static features relevant for the LiDAR sensor. The features are annotated with the types of the objects they represent. The three paths driven by the robot during validation are also reported.

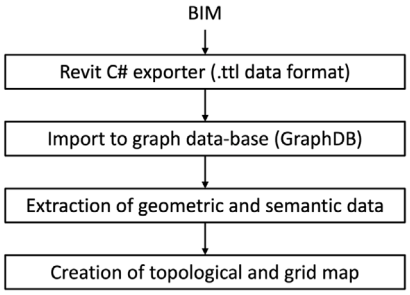


Fig. 4. Conversion from the BIM model to topological and metric maps following the LBD Server route.

4.1.4. Localization

Localization is achieved by querying spatial features and their semantic relations from the database using the well-known PostgreSQL database with the PostGIS spatial extension. We store the JSON-LD property graph representation in the database as well, making it possible to query for relations and entities using the SQL query language. The database is queried for spatial features that are close to the robot. The sensor type is part of the query, resulting in features that are part of an `ObjectFeatureRepresentation` perceivable by the given sensor. The query returns the feature id, feature type, object id and object type for each feature, together with the spatial object that contains the actual coordinate data structure. For example, a query for perceivable features with a 2D LiDAR scanner near the current position, may return the object `{type:''IfcColumn'', id:''96033e''}`, together with its representation `{type:''Polygon'', id:''553236''}`. This explicit symbolic link between the geometry, its interpretation and the object is maintained in the association-based localization approach described in detail by Hendrikx et al. [58].

4.2. Route 2 - LBD server

This route largely consists of three stages (Fig. 4). The first stage is the stage in which a Revit plugin is developed using C#. The stage in which the output of the Revit plugin is converted into a bitmap image in PNG format is the second stage. The last stage is the actual robot navigation.

So, for this route, an existing Revit-to-LBD exporter plugin was expanded with additional functionality that enhances the output data with required additional data, which were mainly geometry (e.g. space boundaries for spaces) and specific object types (e.g. doors in curtain

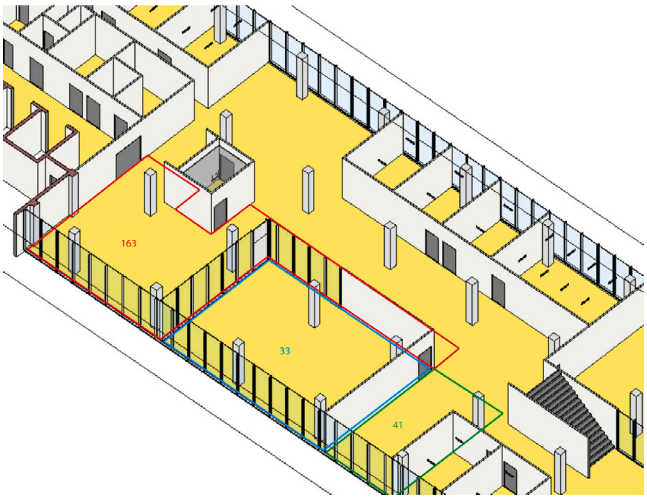


Fig. 5. The BIM model in Revit with the space boundaries for rooms 163, 33, and 41 shown.

walls). The resulting data is loaded into a triple store, namely Ontotext GraphDB.[35] Using few defined scripts (described in de Koning et al. [59]), required geometric and feature data can be retrieved from this triple store (CSV output), and these results are used to automatically generate a 2D floor plan that includes walls and doors mainly together with a topological map that defines how spaces are interconnected. From the 2D floor plan, a gridmap is generated which indicates which cells are occupied by a structural element (e.g., wall or window) and which cells are not occupied by a structural element.

4.2.1. Modified revit-to-LBD exporter

The Revit plug-in is developed using C#. The building elements of the BIM model are collected and written to an RDF graph, namely a Turtle (.ttl) file.[36] The C# script consists of two parts: the first part collects the building elements and the second part writes the Turtle file. In this code, building elements are retrieved using standard C# code, while the geometric data is added to the building elements by writing the `bot:interfaces` and collecting the geometric information of those interfaces. Those interfaces are the space boundaries between rooms and their bounding elements like walls, doors and columns (Fig. 5).

The interfaces of the columns and walls are collected by searching for the bounding elements of a space and their specific category names (column, wall). The 2D geometry of the space boundary between element and space is available in the Revit API and can be used to encode each interface(s) between a wall or column element and a space. Each side of a column or wall generates one `bot:interface`, leading to four distinct interfaces for each rectangular column with the space. The geometry of the walls and columns can then be created by combining the geometry encoded for each of these interfaces (rectangle).

The interfaces of the doors are collected differently, namely by searching for inserts in the walls of type ‘door’. As these doors are embedded in walls, their geometric data needs to be obtained differently. For doors, instance geometry needs to be obtained, which includes a 2D curve that encodes its start and end points. Listing 5 shows the code that is used to write the geometry curves for doors. Note that these curves are simple ‘lines’ in this particular case of doors; yet, both in the Revit API and in the IFC models, the term ‘curve’ is used to refer to this geometry. The ‘line’ element is a subtype of ‘curve’.

```
1  GeometryElement instanceGeometryElement = instance.
     GetInstanceGeometry();
2  foreach (GeometryObject o in instanceGeometryElement)
3  {
4      // Try to find curves
5      Curve curve = o as Curve;
6      if (curve != null)
7      {
8          if (curve.GetEndPoint(0).X == curve.GetEndPoint(1).X
             && curve.GetEndPoint(0).Y == curve.GetEndPoint(1).Y
             )
9              continue;
10
11         else
12         {
13             tString += NL + "inst:Interface_" +
                 interfaceCounter + NLT + "a bot:Interface ;" + NLT
                  + "bot:interfaceOf " + "inst:room_" + room.Id.
                 ToString() + ", " + "inst:" + "door" + "_" + eli.Id.
                 ToString() + " ;";
14             tString += NLT + "fog:asSfa_v2-WKT \"LINESTRING (" +
                  (curve.GetEndPoint(0).X * 12* 25.4).ToString().
                 Replace(',', '.') + " " + (curve.GetEndPoint(0).Y
                 * 12* 25.4).ToString().Replace(',', '.') + ", " +
                 (curve.GetEndPoint(1).X * 12* 25.4).ToString().
                 Replace(',', '.') + " " + (curve.GetEndPoint(1).Y
                 * 12* 25.4).ToString().Replace(',', '.') + ")\" ."
                  + NL;
15         }
16         interfaceCounter++;
17     }
18 }
```

Listing 5: C# code for retrieving door curves and outputting them to an RDF graph

4.2.2. Output LBD graph

After running the custom Revit-to-LBD exporter, data is available in a comprehensive RDF graph. This RDF graph relies on the prefixes listed in Listing 6.

```
1 @prefix bot: <https://w3id.org/bot#> .
2 @prefix props: <https://w3id.org/props#> .
3 @prefix beo: <https://pi.pauwel.be/voc/buildingelement#> .
4 @prefix fog: <https://w3id.org/fog#> .
5 @prefix rdfs: <https://www.w3.org/2000/01/rdf-schema#> .
6 @prefix rvt: <https://example.org/rvt#> .
7 @prefix inst: <https://linkedbuildingdata.net/ifc/resources2
    0200915_130453/> .
```

Listing 6: Namespaces and prefixes in the output TTL graph

The different building elements included in the RDF graph are the rooms, the walls, the columns, the doors and the windows. All those building elements are exported to the Turtle file (.ttl) with the properties they have, ranging from the dimensions to the level of the element. As can be seen in Listing 7, this representation also includes the 2D geometric representation (line string) with exact coordinates of the building element. Also the interfaces between two components (e.g. room and wall) are stored, following the BOT ontology and similarly including 2D geometry, allowing to find the areas where an interface is constituted by doors that can be passed through.

[35] https://graphdb.ontotext.com/
[36] https://www.w3.org/TR/turtle/

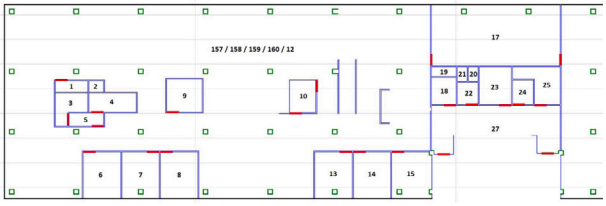


Fig. 6. Partial view of the generated 2D map of the 8th floor of the Atlas building. Green (columns), red (doors), blue (walls).

```
inst:door_499701
  a bot:Element ;
  a beo:Door ;
  props:number "499701" ;
  props:TypeId "399499" ;
  props:Category "Doors" ;
  props:PhaseDemolished "None" ;
  props:larguraporta "0.15" ;
  props:SillHeight "0.00" ;
  props:PhaseCreated "New Construction" ;
  props:Category "Doors" ;
  props:Type "Simple door" ;
  props:relativeheight "1.98" ;
  props:DesignOption "-1" ;
  props:Family "Simple door" ;
  props:FamilyandType "Simple door: Simple door" ;
  props:Level "9th Floor" ;
  props:Image "<None>" ;
  props:Volume "0.03 m3" ;
  props:HostId "498863" ;
  props:HeadHeight "2.00" ;
  props:Area "2 m2" ;
  props:Level "9th Floor";
  fog:asSfa_v2-WKT "LINESTRING (202550.100211374 82607.952094
    0842, 202550.100211374 83607.9520940842)(202650.10021137
    4 82607.9520940842, 202650.100211374 83607.9520940842)" .

inst:Interface_425
  a bot:Interface ;
  bot:interfaceOf inst:room_545656, inst:wall_363209 ;
  fog:asSfa_v2-WKT "LINESTRING (211159.204694596 1341.7006532
    0252, 211159.204694596 18291.7006532025)" .

inst:Interface_426
  a bot:Interface ;
  bot:interfaceOf inst:room_545656, inst:wall_511201 ;
  fog:asSfa_v2-WKT "LINESTRING (211159.204694596 18291.700653
    2025, 206359.204694596 18291.7006532025)" .
```

Listing 7: Building elements in output RDF graph

So, the geometric data of the elements are added as WKT linestrings (see Section 2.2). Those linestrings are the coordinates of the two points and those points are the start- and endpoint of the curve. So, a line between the two points indicates the location of the interface which means that different lines together form the circumference of the building elements. For the walls, one interface is written per room and the location of those walls are indicated with a line on the side with which it adjoins the room. An exception to this are the walls that have a wall perpendicular to it, those walls are written in two segments and so, two linestrings. This follows the regular 2nd level space boundary approach that is available also within IFC (more information in [60]). The doors are written in two interfaces because the curves of the doors are exported as one for the outer side of the door and one for the inner side of the door. In this way, the geometric data of the different building elements are exported as an RDF graph in Turtle format.

Also the relationships between different building elements are written in the RDF graph (Listing 8). Those relationships are the relationships between the walls and the related doors and windows (`bot:hasSubElement`), and the relationships between the rooms and their bounding elements like the walls and columns (`bot:adjacentElement`). This follows the default LBD formats, ontologies and agreements.

```
inst:wall_447042 bot:hasSubElement inst:door_780485 .
inst:wall_524398 bot:hasSubElement inst:door_780563 .
inst:wall_445683 bot:hasSubElement inst:door_780630 .
inst:wall_445683 bot:hasSubElement inst:door_780658 .
inst:wall_445683 bot:hasSubElement inst:door_780770 .
inst:room_543753 bot:adjacentElement inst:wall_491335 .
inst:room_543753 bot:adjacentElement inst:wall_512526 .
inst:room_543753 bot:adjacentElement inst:wall_491414 .
inst:room_543753 bot:adjacentElement inst:wall_491453 .
inst:room_543755 bot:adjacentElement inst:wall_491335 .
inst:room_543755 bot:adjacentElement inst:wall_491367 .
```

Listing 8: Relations included in the RDF graph between building elements (snippet)

4.2.3. Information retrieval

In the second part of the code (creation of the PNG gridmap), several software applications are used: GraphDB, Matplotlib and Python. A SPARQL query allows to retrieve the required data to create a PNG map (Listing 9). In this phase, the geometric data of the different elements are collected and related to the right room of the building.

```
PREFIX props: <https://w3id.org/props#>
PREFIX bot: <https://w3id.org/bot#>
PREFIX beo: <https://pi.pauwel.be/voc/buildingelement#>
PREFIX fog: <https://w3id.org/fog#>

SELECT DISTINCT ?room ?WKTWall ?WKTDoor ?WKTColumn #
  Determine which variable are shown
WHERE{
    {
        # Make sure that every room is a room
        ?room props:Category "Rooms".
        ?room props:Level "8th Floor".
        # Find an interfaces which is both a bot:interfaceof of
           a room and a wall
        ?interface bot:interfaceOf ?room, ?object.
        ?object a beo:Wall.
        ?interface fog:asSfa_v2-WKT ?WKTWall.
    }
    # Find an interfaces which is both a bot:interfaceof of a
      room and a door
    UNION{
        ?room props:Category "Rooms".
        ?room props:Level "8th Floor".
        ?interface2 bot:interfaceOf ?room, ?object2.
        ?object2 a beo:Door.
        ?interface2 fog:asSfa_v2-WKT ?WKTDoor.
    }
    UNION{
        ?room props:Category "Rooms".
        ?room props:Level "8th Floor".
        ?interface2 bot:interfaceOf ?room, ?object3.
        ?object3 a beo:Column.
        ?interface2 fog:asSfa_v2-WKT ?WKTColumn.
    }
}
```

Listing 9: SPARQL query that can be used to retrieve the required geometric data to create a 2D floor plan

This query creates a table with four columns from which the first column includes the URIs of the rooms of the BIM model. Those rooms are related to the correct linestrings of the walls, doors and columns which are included in the second to the fourth column of the table. This data is then downloaded to a Comma-Separated Values (CSV) file which is used as further input for the conversion of the geometric data.

4.2.4. Floor plan generation

The CSV downloaded from GraphDB is used as input for the Python code with Matplotlib. The Python code uses the linestrings of the building elements to plot the lines between the coordinates of those linestrings, using Matplotlib in PNG format. This conversion process can be used as further input for the creation of maps for indoor robot navigation (see also de Koning et al. [59]). All those plotted lines together form a map of, in this case, the 8th floor of Atlas (see Fig. 6). The Python code plots the different building elements in different colours to highlight the different semantics of the building elements that could be used by the path planner or the localization algorithm to improve performance. This also shows the number of linestrings an element consists of, so four for the columns and two for the doors. The walls have, as explained, just one linestring per room, but by exporting all rooms, the walls have an inner and an outer line. Furthermore, each space can be assigned its own 2D and 3D geometry, which is kept out of the process in this particular test case, but has been done elsewhere with the OBJ format, for example [61,62].

5. Validation

In this section, the two investigated and implemented data flow methods are evaluated against the available criteria in Section 3.4. The evaluation is done by using the generated maps for localization as described by [58] (Route 1 - IFC-JSON file transfer - TM1.3) and for localization and navigation connecting the generated gridmap to state-of-the-art navigation toolboxes provided by the Robotic Operating System (Route 2 - LBD Server - TM2.2). For both methods, the following steps are followed during the validation:

1. Localization is done based on the 2D map generated from the data obtained from the BIM model.
2. The robot navigates from one room of the building to another in a tele-operated (Route 1) or autonomous manner (Route 2)
3. The features extracted from the 2D LiDaR scanner during navigation are compared with the features of the generated maps.

For both Route 1 and Route 2, we achieved semantic localization and autonomous navigation, respectively. Both routes are tested in the Atlas building on the TUe campus. Several simulations and real-world tests were performed, and the final real-world tests were video-recorded, including a visualization of the data available for the robot to perform its localization and navigation. The videos showing the robot localizing (Route 1) and autonomously navigating (Route 2) are publicly available.[37] In principle, both tests were successful in the sense that:

1. data can be transferred successfully from BIM model to robot world model in both cases;
2. robot localization and navigation is feasible in both cases;
3. the data represented by the BIM model and robot world model is *spatially accurate* to the extent that localization is possible;
4. the robot has *semantic knowledge of spaces and connecting interfaces* (e.g., rooms, areas and doors), allowing navigation;
5. the robot has access to *material properties* in relation to its sensing capabilities (e.g. glass);
6. the robot can *query required data efficiently* during operation.

While these outcomes can be expected, these tests were also aimed at investigating how the two data transfer approaches work in practice. While it is recognized that the LBDServer approach (Route 2 - TM2.2) is more extensible, more standard, and potentially more data-rich because of the live data connection, it is also recognized that the IFC-JSON file-based approach (Route 1 - TM1.3) is more light-weight and easier to set-up as the web-based infrastructure and its connections are not needed. Transfer Method 1.3 (IFC-JSON based) is a lot more manual, however, and much less standard, leading to a higher risk for errors and lower scalability. If this approach can be implemented on a web server (see TM 3.1), this would create a lot of added value and potential.

For the different criteria outlined in Section 3.4, a more elaborate validation is given in the below Sections.

[37] https://youtube.com/playlist?list=PLR6weRZsGeth5ML8j0JwxRRMx8nOE04D0

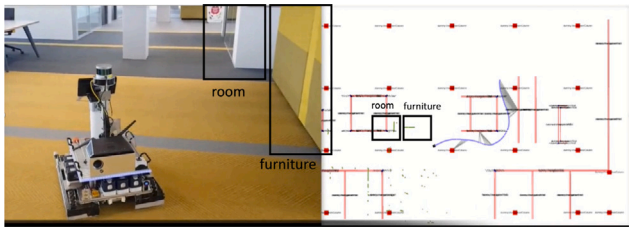


Fig. 7. Left side, view of the building. Right side 2D map with overlapping LiDaR readings (green). The room location on the 2D map is spatially inaccurate as can be seen by the LiDaR readings detecting the room corner being shifted w.r.t. the expected room location on the map. Furniture is detected by the LiDaR scanner, but is not represented in the map.

5.1. The data represented by the BIM model needs to be spatially accurate

In order to ensure correct localization, the spatial accuracy of the maps derived from the BIM data should be high. In other words, given that there are no unmodelled elements in the space where the robot navigates, sensors' readings should match the expected readings based on the map layout. In experiments conducted with both methods, we have seen that that is not always the case. Fig. 7 shows the map used for semantic localization (Route 1). By superimposing the LiDaR readings to the generated 2D map, it can be seen that the corner of the room is detected closer to the robot than what is reported on the map.

Other elements such as stairs might also be reported incorrectly on the map because they were modelled incorrectly in the BIM model. This is an important problem or challenge, because many as-designed BIM models differ from the real world. The challenge here is to make this difference as small as possible, and in any case below an acceptable tolerance level. This tolerance level for differences between stored building geometry and LiDaR scan depends on the case in which the robot navigation is being used. Security- and safety-critical cases need a much smaller tolerance level compared to low-cost and approximate navigation cases. In our current case, we are on the accuracy level of the second: approximate navigation, since our BIM model clearly needs further updates to be usable for autonomous navigation. As this article investigated primarily data transfer methods first, future research can now investigate in more detail how to make updates to the live digital twin or BIM model based on on-site measurements to further improve model accuracy.

Fig. 8 shows a partial view of the map used for autonomous navigation (Route 2). According to the robot's location, the modelled stairs should be perceived by the robot's LiDaR scanner. However, that is not the case which implies that the stairs were modelled closer to the location of the robot than they are in reality. Indeed, as can be seen in Fig. 8, the staircase are more to the right in reality, and behind the column.

5.2. The robot needs to have semantic knowledge of spaces and connecting interfaces (e.g., rooms, areas and doors)

Overall, in both cases, there is good availability of connectivity information of rooms and spaces and elements. As the second transfer method (TM 2.2 - LBD Server) preserves more data and semantics, this

Fig. 8. Left side, view of the building. Right side 2D map with overlapping LiDaR readings (red). The stairs' location on the 2D map is spatially inaccurate as can be seen by the LiDaR readings detecting free space instead of the stairs.

Fig. 9. More detailed BIM models can include more dynamic data, in which case encoding and recognition of interfaces with spaces potentially becomes significantly more complex.

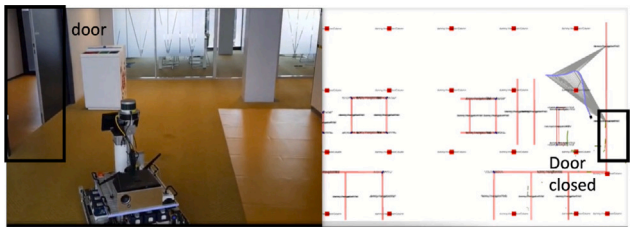


Fig. 10. Left side, view of the building. Right side 2D map with overlapping LiDaR readings (green). The door is represented as closed in the map (red lines) but it is open in reality and detected as open by the lidar (dotted green line).

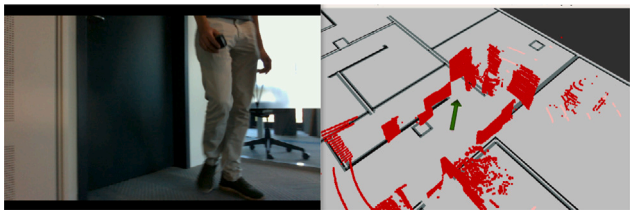


Fig. 11. Left side, view of the building. Right side 2D map with overlapping LiDaR readings (red). The door is represented as open in the map but it is closed in reality.

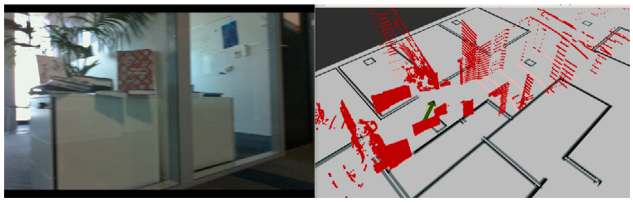


Fig. 12. Left side, view of the building. Right side 2D map with overlapping LiDaR readings (red). The robot faces a glass wall and the LiDaR readings pass through indicating free space.

data transfer method performs better than TM 1.3 (IFC-JSON file). In the former case, a clear topology graph is easily available, including interfaces between elements and spaces, so the space topology and element interfaces can be easily retrieved. This is much more complex in TM1.3, where much of this data is flattened or more indirectly encoded in the JSON version of IFC.

It is worthwhile to note here that some structural elements modelled in the BIM model and extracted by either of the data transfer methods change position over time (e.g., doors can be open or closed). While this is of limited impact in our case as we used a limited sample file (only door positions change), this can become a much more significant challenge if more furniture is made available in the model as is for example the case in the model displayed in Fig. 9.

For the case of the doors, the difference of status (e.g., open vs closed) is important when using maps derived from the data of a BIM model for navigation and localization. Namely, a closed or locked door clearly occludes the path towards a navigation goal which needs to be taken into account during path planning. For the work addressed in this paper, we assumed all doors closed for the map generated using Route 1 and all doors open for the map generated using Route 2. However, this assumption was not always met as can be seen in Fig. 10 for Route 1 and in Fig. 11 for Route 2. In the case of Fig. 10, in fact, a solid wall was expected, while in reality a door was presented. So the BIM model did not match with reality. In the case of Fig. 11, an open door was expected, while it was fully closed in reality.

5.3. The robot needs to be aware of material properties and the relation to its sensing capabilities (e.g. glass)

Mobile robots are mostly equipped with 2D or 3D LiDaR scanners which are sensitive to glass surfaces. A glass surface might be detected as free space because the LiDaR beams pass through the glass as shown in Fig. 12 (Route 2). This might result in the wrong navigation decisions by the robot. When the building digital twin provides information on material properties, the robot can adjust its path or correctly interpret the LiDaR readings.

In TM1.3 (IFC-JSON file), material properties had been lost during the conversion process. It is possible to still include this information in the eventual JSON data by modifying the data transfer procedure. Because of the direct character of the transfer (not extensible), this easily becomes an error-prone and unscalable method. In contrast, in TM2.2 (LBD Server), further material characteristics are readily available, as well as smart building data (BRICK ontology), and potentially even damage data (DOT ontology). They can simply be queried on demand and then used. In this sense, the second approach that relies on TM2.2 performs better because of its higher level of extensibility.

5.4. The robot needs to be able to query this data efficiently (real time) during operation, through an abstracted programming or query interface

For both data routes, maps are generated offline from the data extracted from the BIM model. For Route 1, all spatial and semantic data were stored in a database deployed on the robot which was queried in real-time during movements. For Route 2, the data to create the gridmap were queried in real-time before starting navigation. In that sense, both methods contained manual operations, leading to needed improvements to make the procedures live and real-time. The amount of manual operations is however considerably smaller in the case of TM2.2 (Route 2), as most of the operations do occur through queries against an online database server.

In any case, as a conclusion, a very important challenge arose here. Namely, the robot performs its localization and navigation always locally in ROS (on the 'edge') for obvious safety and computational

efficiency purposes (e.g. obstacle avoidance). Data about the building is available on the cloud, in both data transfer methods. In all possible scenarios, a clear and good strategy is required to balance how often and how much data is requested and queried from the cloud. For example, it may be one approach to query all data in IFC-JSON format before starting the navigation and then to perform all further navigation locally. This reduces networking time and cloud computing needs. Alternatively, one could choose to always keep the link with data on the cloud, which is closer to the LBD server approach, and regularly perform data checks through queries over the local network. This increases networking load and computational load for the server, yet likely leads to more secure navigation for the robot. This particular challenge of *balancing cloud and edge computing efforts* needs to be investigated in more detail in the future.

6. Conclusion and future work

6.1. Conclusions

This paper presents several data transfer methods (TM1 - TM3) to extract data from BIM models and make them available to create semantic and grid maps for robot navigation. The different transfer methods are listed and their characteristics are evaluated in one complete table. A distinction is hereby made between (1) traditional IFC-based file transfer, (2) live linked data server, and (3) JSON-based web services. Evaluated characteristics are their scope of standardization, information transfer steps, dependencies, extensibility at run-time, and reliability. Of the methods presented, the two most significant ones, namely IFC-JSON file transfer (TM1.3) and LBD server (TM2.2), were implemented and validated with robot localization and navigation tests. The IFC-JSON file transfer method (TM1.3) was hereby tested as an alternative to transfer method TM3.1 that relies on JSON-based web services. As this was too costly to implement in this stage of research, the alternative and most closely resembling routine via IFC-JSON file transfer was implemented as well. As this came out promising, future work can look into porting this method to a server-based implementation, which will require further standardization of the IFC-JSON file format.

Validation experiments (Route 1 and Route 2) above all highlight the feasibility of the approach in general, i.e., of creating maps from original BIM models and using them for real-time robot navigation. Both the IFC-JSON and the LBD server routine came out successful. While the IFC-JSON approach was considerably more manual and did not generate live data access, the LBD server approach appears to be more scalable and extensible, although this approach needs more work to set up correctly. The two experiments were evaluated using outlined information transfer criteria:

1. broad standardisaton scope
2. few data transformation operations
3. software-neutral dependencies
4. extensible at robot run-time
5. up-to-date data (reliability)

In addition to these goals or objectives, further functional criteria were evaluated in the context of the specific use case that was tested, namely indoor robot navigation in an existing building. This included the following additional criteria:

1. The data represented by the BIM model needs to be *spatially accurate*.
2. The robot needs to have *semantic knowledge of spaces and connecting interfaces* (e.g., rooms, areas and doors).
3. The robot needs to be aware of *material properties* and the relation to its sensing capabilities (e.g. glass)
4. The robot needs to be able to *query this data efficiently (real time)* during operation, through an abstracted programming or query interface.

6.2. Improvements and challenges

Several improvements and challenges have been identified based on the experiments made. First and most importantly, the workflows and the overall success of navigating through a building based on BIM model data depends on the quality of that BIM model data. Standardization of the data structure, including a structured BIM modelling approach, is needed in order to make the transfer of BIM data to robot streamlined and less expensive in terms of programming effort.

The BIM model also needs to align sufficiently well with reality, so that the geometry is sufficiently reliable and recognizable for robot navigation. The elements of the BIM model need to be spatially and geometrically accurate to achieve required localization and navigation performance. Attention should be given to achieve a high degree of correspondence between the BIM model and the actual construction of the building to allow accurate matching of real-time sensor readings and the maps generated from the BIM model. It would be helpful if a quality assurance and checking procedure would be available that checks whether the difference between model and geometry remains within an acceptable tolerance level. In this case, a specific tolerance level needs to be defined for allowing robot localization and navigation, to be able to identify implementation cost and risk for the robot navigation.

6.3. Future work

In terms of future work, some specific features can easily generate added value for diverse applications with the autonomous robot. First, the available information of the elements (e.g. material, thickness) can be used for the correct interpretation of robot sensor readings. Being aware of the presence of glass can be a very valuable piece of information for a robot, allowing it to take an alternative route.

Second, dynamic features, such as doors, furniture and windows, should be represented in the generated map in a dynamic manner (i.e. movable) to account for a possible change in state (e.g. open and closed). For this particular case, valuable future work lies in investigating whether the building data can be updated dynamically to take into account the dynamics of the built environment the robot operates in. We regard establishing this bi-directional data connection between robots and data extracted from BIM as a direct next step in our research line, leading to a more live digital twin of the building. This can not only help robot navigation, this could also be used to mend the BIM model and correct it. When the correspondence between the original BIM model and the data extracted and used to create the map (LBD cloud) is preserved, then updates to the map could also be reflected back to the BIM model.

Finally, our research has not yet investigated the use of 3D features for localization and navigation; hence the maps generated from BIM data are two-dimensional. Future research will look at the usage of 3D sensors (e.g., point clouds) to improve localization and navigation performance which will require the generation of 3D maps from the data exported from BIM. This is possible, but is expected to increase computational complexity significantly.

Declaration of competing interest

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

Data availability

All data related to this article is publicly available at https://pi.pauwel.be/tue/ADVEI2023_robotnavigation_additionaldata.html

References

[1] T. Bock, The future of construction automation: Technological disruption and the upcoming ubiquity of robotics, Autom. Constr. 59 (2015) 113–121, https://doi.org/10.1016/j.autcon.2015.07.022.

[2] M.P. Fanti, A.M. Mangini, M. Roccotelli, B. Silvestri, Hospital drugs distribution with autonomous robot vehicles, in: 2020 IEEE 16th International Conference on Automation Science and Engineering, CASE, IEEE, 2020, pp. 1025–1030, https://doi.org/10.1109/CASE48305.2020.9217043.

[3] V. Prabakaran, M.R. Elara, T. Pathmakumar, S. Nansai, Floor cleaning robot with reconfigurable mechanism, Autom. Constr. 91 (2018) 155–165, https://doi.org/10.1016/j.autcon.2018.03.015.

[4] N. Boysen, D. Briskorn, S. Emde, Parts-to-picker based order processing in a rack-moving mobile robots environment, European J. Oper. Res. 262 (2) (2017) 550–562, https://doi.org/10.1016/j.ejor.2017.03.053.

[5] R. Siegwart, I.R. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots, second ed., MIT Press, 2011.

[6] R. Yagfarov, M. Ivanou, I. Afanasyev, Map comparison of lidar-based 2D SLAM algorithms using precise ground truth, in: 2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV, IEEE, 2018, pp. 1979–1983, https://doi.org/10.1109/ICARCV.2018.8581131.

[7] J.J. Im, A. Leonessa, A. Kurdila, A real-time data compression and occupancy grid map generation for ground-based 3D lidar data using wavelets, in: Dynamic Systems and Control Conference, Vol. 44182, 2010, pp. 557–562, https://doi.org/10.1115/DSCC2010-4269.

[8] L. Naik, S. Blumenthal, N. Huebel, H. Bruyninckx, E. Prassler, Semantic mapping extension for OpenStreetMap applied to indoor robot navigation, in: 2019 International Conference on Robotics and Automation, ICRA, IEEE, 2019, pp. 3839–3845, https://doi.org/10.1109/ICRA.2019.8793641.

[9] A.R. Khairuddin, M.S. Talib, H. Haron, Review on simultaneous localization and mapping (SLAM), in: 2015 IEEE International Conference on Control System, Computing and Engineering, ICCSCE, IEEE, 2015, pp. 85–90, https://doi.org/10.1109/ICCSCE.2015.7482163.

[10] B. Patle, A. Pandey, D. Parhi, A. Jagadeesh, et al., A review: On path planning strategies for navigation of mobile robot, Def. Technol. 15 (4) (2019) 582–606, https://doi.org/10.1016/j.dt.2019.04.011.

[11] P.K. Panigrahi, S.K. Bisoy, Localization strategies for autonomous mobile robots: A review, J. King Saud Univ. - Comput. Inf. Sci. 34 (8, Part B) (2022) 6019–6039, https://doi.org/10.1016/j.jksuci.2021.02.015.

[12] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, W. Burgard, An evaluation of the RGB-d SLAM system, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 1691–1696, https://doi.org/10.1109/ICRA.2012.6225199.

[13] J. Crespo, J.C. Castillo, O.M. Mozos, R. Barber, Semantic information for robot navigation: A survey, Appl. Sci. 10 (2) (2020) 497, https://doi.org/10.3390/app10020497.

[14] I. Kostavelis, A. Gasteratos, Semantic mapping for mobile robotics tasks: A survey, Robot. Auton. Syst. 66 (2015) 86–103, https://doi.org/10.1016/j.robot.2014.12.006.

[15] C. Eastman, C.M. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors, second ed., John Wiley & Sons, 2011.

[16] A. Borrmann, M. König, C. Koch, J. Beetz, Building Information Modeling, Springer International Publishing, 2018, https://doi.org/10.1007/978-3-319-92862-3.

[17] ISO 16739, Industry Foundation Classes (IFC) for Data Sharing in the Construction and Facility Management Industries, International Standardisation Organisation, Geneva, Switzerland, 2013.

[18] B. Becerik-Gerber, F. Jazizadeh, N. Li, G. Calis, Application areas and data requirements for BIM-enabled facilities management, J. Constr. Eng. Manag. 138 (3) (2012) 431–442, https://doi.org/10.1061/(ASCE)CO.1943-7862.0000433.

[19] Y. Arayici, T. Onyenobi, C. Egbu, Building information modelling (BIM) for facilities management (FM): The mediacity case study approach, Int. J. 3-D Inf. Model. (IJ3DIM) 1 (1) (2012) 55–73, https://doi.org/10.4018/ij3dim.2012010104.

[20] M. Kassem, G. Kelly, N. Dawood, M. Serginson, S. Lockley, BIM in facilities management applications: a case study of a large university complex, Built Environ. Proj. Asset Manag. 5 (3) (2015) 261–277, https://doi.org/10.1108/BEPAM-02-2014-0011.

[21] G.B. Ozturk, Interoperability in building information modeling for AECO/FM industry, Autom. Constr. 113 (2020) 103122, https://doi.org/10.1016/j.autcon.2020.103122.

[22] L. Chamari, E. Petrova, P. Pauwels, A web-based approach to BMS, BIM and IoT integration: a case study, in: Proceedings of the CLIMA 2022 Conference, Rotterdam, Netherlands, 2022, https://doi.org/10.34641/clima.2022.228.

[23] P. Pauwels, K. McGlinn, Buildings and Semantics: Data Models and Web Technologies for the Built Environment, first ed., CRC Press, 2022, https://doi.org/10.1201/9781003204381.

[24] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M.B. Kjærgaard, M. Srivastava, K. Whitehouse, Brick: Towards a unified metadata schema for buildings, in: Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments, BuildSys '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 41–50, https://doi.org/10.1145/2993422.2993577.

[25] P. Pauwels, S. Zhang, Y.-C. Lee, Semantic web technologies in AEC industry: A literature overview, Autom. Constr. 73 (2017) 145–165, https://doi.org/10.1016/j.autcon.2016.10.003.

[26] G.F. Schneider, W. Terkaj, P. Pauwels, Reusing domain ontologies in linked building data: the case of building automation and control, in: 8th International Workshop on Formal Ontologies Meet Industry, Vol. 2050, 2017, URL: https://ceur-ws.org/Vol-2050/FOMI_paper_4.pdf.

[27] G. Schneider, M. Rasmussen, P. Bonsma, J. Oraskari, P. Pauwels, Linked building data for modular building information modelling of a smart home, in: J. Karlshøj, R. Scherer (Eds.), Proceedings of the 12th European Conference on Product and Process Modelling, CRC Press, 2018, pp. 407–414, https://doi.org/10.1201/9780429506215-51.

[28] M.H. Rasmussen, M. Lefrançois, G.F. Schneider, P. Pauwels, BOT: the building topology ontology of the W3C linked building data group, Semant. Web 12 (1) (2019) 143—161, https://doi.org/10.3233/SW-200385.

[29] M. Pritoni, D. Paine, G. Fierro, C. Mosiman, M. Poplawski, A. Saha, J. Bender, J. Granderson, Metadata schemas and ontologies for building energy applications: A critical review and use case analysis, Energies 14 (7) (2021) https://doi.org/10.3390/en14072024.

[30] H. Wicaksono, B. Yuce, K. McGlinn, O. Calli, Smart cities and buildings, in: P. Pauwels, K. McGlinn (Eds.), Buildings and Semantics: Data Models and Web Technologies for the Built Environment, CRC Press, London, UK, 2022, https://doi.org/10.1201/9781003204381.

[31] P. Pauwels, D. Shelden, J. Brouwer, D. Sparks, S. Nirvik, T. McGinley, Open data standards and BIM on the cloud, in: P. Pauwels, K. McGlinn (Eds.), Buildings and Semantics: Data Models and Web Technologies for the Built Environment, CRC Press, London, UK, 2022, https://doi.org/10.1201/9781003204381.

[32] C. Boje, S. Kubicki, A. Guerriero, Y. Rezgui, A. Zarli, Digital twins for the built environment, in: P. Pauwels, K. McGlinn (Eds.), Buildings and Semantics: Data Models and Web Technologies for the Built Environment, CRC Press, London, UK, 2022, https://doi.org/10.1201/9781003204381.

[33] D. Mavrokapnidis, K. Katsigarakis, P. Pauwels, E. Petrova, I. Korolija, D. Rovas, A linked-data paradigm for the integration of static and dynamic building data in digital twins, in: Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 369–372, https://doi.org/10.1145/3486611.3491125.

[34] S. Kim, M. Peavy, P.-C. Huang, K. Kim, Development of BIM-integrated construction robot task planning and simulation system, Autom. Constr. 127 (2021) 103720, https://doi.org/10.1016/j.autcon.2021.103720, URL: https://www.sciencedirect.com/science/article/pii/S0926580521001710.

[35] K. Kim, M. Peavy, BIM-based semantic building world modeling for robot task planning and execution in built environments, Autom. Constr. 138 (2022) 104247, https://doi.org/10.1016/j.autcon.2022.104247.

[36] S. Karimi, I. Iordanova, D. St-Onge, Ontology-based approach to data exchanges for robot navigation on construction sites, J. Inf. Technol. Constr. 26 (2021) 546–565, https://doi.org/10.36680/j.itcon.2021.029.

[37] P. Pauwels, W. Terkaj, EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology, Autom. Constr. 63 (2016) 100–133, https://doi.org/10.1016/j.autcon.2015.12.003.

[38] H.-K. Kang, K.-J. Li, A standard indoor spatial data model—OGC IndoorGML and implementation approaches, ISPRS Int. J. Geo-Inf. 6 (4) (2017) https://doi.org/10.3390/ijgi6040116.

[39] J. Werbrouck, M. Senthilvel, M. Rasmussen, Federated data storage for the AEC industry, in: P. Pauwels, K. McGlinn (Eds.), Buildings and Semantics: Data Models and Web Technologies for the Built Environment, CRC Press, London, UK, 2022, https://doi.org/10.1201/9781003204381.

[40] S. Thrun, W. Burgard, D. Fox, Probabilistic Robotics, Vol. 1, MIT press Cambridge, 2005.

[41] P.K. Panigrahi, S.K. Bisoy, Localization strategies for autonomous mobile robots: A review, J. King Saud Univ. - Comput. Inf. Sci. (2021) https://doi.org/10.1016/j.jksuci.2021.02.015.

[42] S. Thrun, Learning occupancy grid maps with forward sensor models, Auton. Robots 15 (2) (2003) 111–127, https://doi.org/10.1023/A:1025584807625.

[43] M. Elbanhawi, M. Simic, Sampling-based robot motion planning: A review, IEEE Access 2 (2014) 56–77, https://doi.org/10.1109/ACCESS.2014.2302442.

[44] W. Hess, D. Kohler, H. Rapp, D. Andor, Real-time loop closure in 2D LIDAR SLAM, in: 2016 IEEE International Conference on Robotics and Automation, ICRA, 2016, pp. 1271–1278, https://doi.org/10.1109/ICRA.2016.7487258.

[45] G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with Rao–Blackwellized particle filters, IEEE Trans. Robot. 23 (1) (2007) 34–46, https://doi.org/10.1109/TRO.2006.889486.

[46] S. Kohlbrecher, J. Meyer, O. von Stryk, U. Klingauf, A flexible and scalable SLAM system with full 3D motion estimation, in: Proc. IEEE International Symposium on Safety, Security and Rescue Robotics, SSRR, IEEE, 2011, https://doi.org/10.1109/SSRR.2011.6106777.

[47] X. Qi, W. Wang, M. Yuan, Y. Wang, M. Li, L. Xue, Y. Sun, Building semantic grid maps for domestic robot navigation, Int. J. Adv. Robot. Syst. 17 (1) (2020) https://doi.org/10.1177/1729881419900066.

[48] M. Himstedt, E. Maehle, Semantic Monte–Carlo localization in changing environments using RGB-D cameras, in: 2017 European Conference on Mobile Robots, ECMR 2017, 2017, https://doi.org/10.1109/ECMR.2017.8098711.

[49] International Organization for Standardization, ISO 10303-11: Industrial automation systems and integration - product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual, 2004.

[50] P. Pauwels, W. Terkaj, T. Krijnen, J. Beetz, Coping with lists in the ifcOWL ontology, in: 22nd Workshop of the European Group of Intelligent Computing in Engineering, Eindhoven, Netherlands, 2015, pp. 113–122.

[51] P. Pauwels, T. Krijnen, W. Terkaj, J. Beetz, Enhancing the ifcOWL ontology with an alternative representation for geometric data, Autom. Constr. 80 (2017) 77–94, https://doi.org/10.1016/j.autcon.2017.03.001.

[52] C. Zhang, J. Beetz, B. de Vries, BimSPARQL: Domain-specific functional SPARQL extensions for querying RDF building data, Semant. Web 9 (6) (2018) 829–855, https://doi.org/10.3233/SW-180297.

[53] M. Rasmussen, M. Lefrançois, P. Pauwels, C. Hviid, J. Karlshøj, Managing interrelated project information in AEC Knowledge Graphs, Autom. Constr. 108 (2019) https://doi.org/10.1016/j.autcon.2019.102956.

[54] P. Pauwels, Information exchange over the web for the AEC industry, in: Eastman Symposium, 2021, URL: https://dbl.gatech.edu/eastman-symposium.

[55] G. Fierro, P. Pauwels, Survey of Metadata Schemas for Data-Driven Smart Buildings (Annex 81), Technical Report, International Energy Agency, 2022, https://annex81.iea-ebc.org/publications.

[56] D. Kim, A. Goyal, A. Newell, S. Lee, J. Deng, V.R. Kamat, Semantic relation detection between construction entities to support safe human-robot collaboration in construction, in: Computing in Civil Engineering 2019, 2019, pp. 265–272, https://doi.org/10.1061/9780784482438.034.

[57] P. Pauwels, A. Roxin, SimpleBIM: from full ifcOWL graphs to simplified building graphs, in: S. Christodoulou, R. Scherer (Eds.), Ework and Ebusiness in Architecture, Engineering and Construction, CRC Press, 2016, pp. 11–18.

[58] R.W.M. Hendrikx, P. Pauwels, E. Torta, H.P.J. Bruyninckx, M.J.G. van de Molengraft, Connecting semantic building information models and robotics: An application to 2D LiDAR-based localization, in: 2021 IEEE International Conference on Robotics and Automation, ICRA, 2021, https://doi.org/10.1109/ICRA48506.2021.9561129.

[59] R. de Koning, E. Torta, P. Pauwels, R. Hendrikx, M. van de Molengraft, Queries on semantic building digital twins for robot navigation, in: 9th Linked Data in Architecture and Construction Workshop, in: CEUR Workshop Proceedings, 3081, CEUR-WS.org, 2021, pp. 32–42, URL: https://ceur-ws.org/Vol-3081/03paper.pdf.

[60] G.N. Lilis, G.I. Giannakis, D.V. Rovas, Automatic generation of second-level space boundary topology from IFC geometry inputs, Autom. Constr. 76 (2017) 108–124, https://doi.org/10.1016/j.autcon.2016.08.044.

[61] M. Bonduel, A. Wagner, P. Pauwels, M. Vergauwen, R. Klein, Including widespread geometry formats in semantic graphs using RDF literals, in: Proceedings of the 2019 European Conference for Computing in Construction, European Council on Computing in Construction, 2019, pp. 341–350, https://doi.org/10.35490/ec3.2019.166.

[62] J. Werbrouck, P. Pauwels, M. Bonduel, J. Beetz, W. Bekers, Scan-to-graph: semantic enrichment of existing building geometry, Autom. Constr. 119 (2020) 15, https://doi.org/10.1016/j.autcon.2020.103286.