Knowledge-Based Systems 232 (2021) 107440

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

ViMantic, a distributed robotic architecture for semantic mapping in indoor environments

D. Fernandez-Chaves [a,b,*], J.R. Ruiz-Sarmiento [a], N. Petkov [b], J. Gonzalez-Jimenez [a]

[a] *Machine Perception and Intelligent Robotics group (MAPIR), Dept. of System Engineering and Automation, Biomedical Research Institute of Malaga (IBIMA), University of Malaga, Spain*
[b] *Johann Bernoulli Institute of Mathematics and Computing Science, University of Groningen, The Netherlands*

ARTICLE INFO

Article history:
Received 8 October 2020
Received in revised form 9 July 2021
Accepted 22 August 2021
Available online 25 August 2021

Keywords:
Semantic maps
Robotic architecture
Mobile robots
Unity 3D
ROS
Object detection
Detectron2
Robot@Home

ABSTRACT

Semantic maps augment traditional representations of robot workspaces, typically based on their geometry and/or topology, with meta-information about the properties, relations and functionalities of their composing elements. A piece of such information could be: fridges are appliances typically found in kitchens and employed to keep food in good condition. Thereby, semantic maps allow for the execution of high-level robotic tasks in an efficient way, *e.g.* "*Hey robot, Store the leftover salad*". This paper presents *ViMantic*, a novel semantic mapping architecture for the building and maintenance of such maps, which brings together a number of features as demanded by modern mobile robotic systems, including: (i) a formal model, based on ontologies, which defines the semantics of the problem at hand and establishes mechanisms for its manipulation; (ii) techniques for processing sensory information and automatically populating maps with, for example, objects detected by cutting-edge CNNs; (iii) distributed execution capabilities through a client–server design, making the knowledge in the maps accessible and extendable to other robots/agents; (iv) a user interface that allows for the visualization and interaction with relevant parts of the maps through a virtual environment; (v) public availability, hence being ready to use in robotic platforms. The suitability of *ViMantic* has been assessed using Robot@Home, a vast repository of data collected by a robot in different houses. The experiments carried out consider different scenarios with one or multiple robots, from where we have extracted satisfactory results regarding automatic population, execution times, and required size in memory of the resultant semantic maps.

1. Introduction

Mobile robots are progressively landing in human environments like hotels, hospitals, offices, homes, etc., carrying out elementary tasks as caregivers, security guards or house cleaners, among others [1,2]. A key aspect for their successful operation in those applications is the ability to model and manage the information relevant to the tasks to be accomplished. Let us suppose a scenario where a robot is commanded to heat the meal. For addressing such a task the robot must know which world elements could be used to warm up the food (*e.g.* stoves, microwaves, ovens, etc.), where they can be found, and how to interact with them. This high-level information of the world is also called *semantic knowledge*, since it provides meta-information about the elements in the environment regarding their relations, characteristics, and functionality, that is, their semantics [3,4]. An example of this knowledge could be that stoves are appliances, typically placed in kitchens, that can warm up food. When semantic knowledge is used to enhance the traditionally available information about the robot workspace, *e.g.* geometric and/or topological maps, objects detected in the environment, etc., this results in a *semantic map* [5,6].

In this way, in the scope of mobile robotics, semantic maps are models that include information about spatial elements (rooms, objects, etc.) augmented with the semantics required for an efficient robot operation (see Fig. 1). These maps empower the cognitive capabilities of robots, enabling them to carry out high-level queries (*e.g.* to retrieve the objects that are suitable to perform a given a tasks and where they can be found), or to infer new knowledge (*e.g.* the type of a room according to the objects detected inside) [7–9].

In order to be operative in modern mobile robotic systems, a semantic map has to exhibit certain features as well as mechanisms for its management, which are provided by *semantic mapping architectures*. They include:

* Corresponding author at: Machine Perception and Intelligent Robotics group (MAPIR), Dept. of System Engineering and Automation, Biomedical Research Institute of Malaga (IBIMA), University of Malaga, Spain.

E-mail addresses: davfercha@uma.es (D. Fernandez-Chaves), jotaraul@uma.es (J.R. Ruiz-Sarmiento), n.petkov@rug.nl (N. Petkov), javiergonzalez@uma.es (J. Gonzalez-Jimenez).

https://doi.org/10.1016/j.knosys.2021.107440

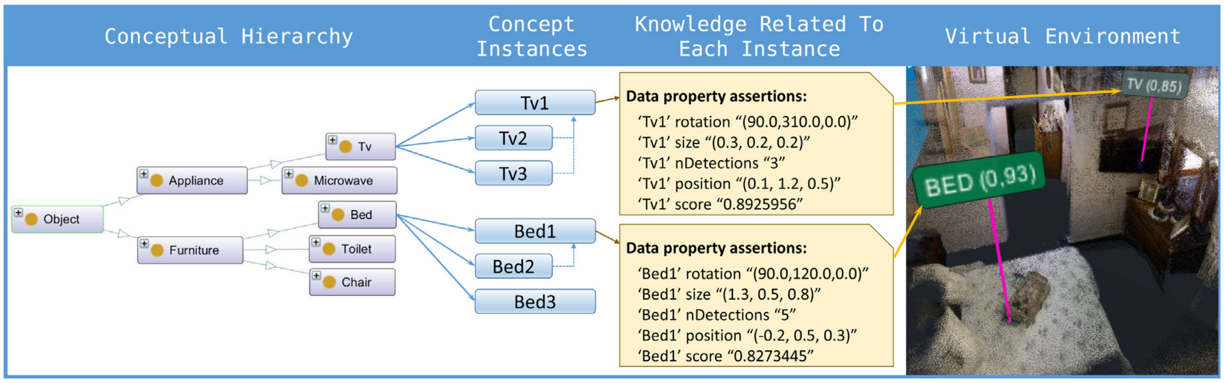

Conceptual Hierarchy
Concept Instances
Knowledge Related To Each Instance
Virtual Environment
Object
Appliance
Furniture
Tv
Microwave
Bed
Toilet
Chair
Tv1
Tv2
Tv3
Bed1
Bed2
Bed3
Data property assertions:
'Tv1' rotation "(90.0,310.0,0.0)"
'Tv1' size "(0.3, 0.2, 0.2)"
'Tv1' nDetections "3"
'Tv1' position "(0.1, 1.2, 0.5)"
'Tv1' score "0.8925956"
Data property assertions:
'Bed1' rotation "(90.0,120.0,0.0)"
'Bed1' size "(1.3, 0.5, 0.8)"
'Bed1' nDetections "5"
'Bed1' position "(-0.2, 0.5, 0.3)"
'Bed1' score "0.8273445"
TV (0,85)
BED (0,93)

Fig. 1. Example of a simple semantic map built by *ViMantic*. It includes a conceptual hierarchy modeling the properties of the elements appearing in the domain at hand as well as their relations, instances of those elements and their observed features, and a screenshot of the GUI designed to show part of that information to the user.

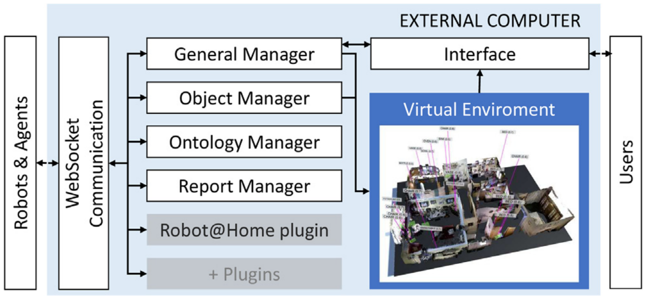


Fig. 2. Overview of the proposed server side of the architecture showing its main components and their interconnections. The white boxes stand for components, the blue one represents the 3D virtual environment and the gray boxes are optional plugins that add extra functionalities.

- **A well defined model representation**. A key component is a formalized and clear model for accommodating the semantic information, also including mechanisms for its manipulation, *i.e.* insertion, modification or query [9–11].
- **Model population**. The chosen model has to be *automatically populated* with information coming from the environment at hand. For that, they are needed methods that transform robot sensory data into high level information, for example images into recognized objects [5,12,13].
- **Distributed execution**. As these architectures are to be ran in mobile robots, which usually exhibit heavily constrained resources, a valuable feature is to provide mechanisms to operate in a *distributed fashion*. This permits the externalization of the building/management of the map, enabling a robot to retrieve just the information needed to complete its tasks from a centralized device, as well as the utilization of the map by other agents or intelligent devices sharing the same workspace [14–16].
- **Suitable interfaces**. Additionally, for robots collaborating with people, the architecture must provide an *interface to interact with humans* in different ways, *e.g.* to show and retrieve high-level information from the map [17,18].
- **Quality control**. Another desirable feature is the measurement of the quality of the resultant maps, *i.e.* how successful the architecture is when building these representations. This also enables a comparison of different state-of-the-art methodologies for semantic mapping [19,20].
- **Public availability**. The architecture should be *"ready to use"*, meaning that its implementation has to be public and easily integrable in most robotic platforms.

Although there are numerous works in the literature providing some of these features and mechanisms [15,21,22], to the best of our knowledge, there is no a solution providing all of them.

In this paper, we contribute a comprehensive semantic mapping architecture including both, state-of-the-art techniques and dedicated components, ready to be integrated in mobile robotic platforms. The proposed solution, coined *ViMantic*, satisfies the common issues previously posed, i.e. model definition, automatic population, distributed execution, human–robot interface, quality control, and public availability. For that, *ViMantic* has been designed as a client–server architecture that can run in different devices, then ready to be adapted to the emergent paradigm of edge computing [23]. Briefly, the server, which could operate on the own robot or on an external device (*e.g.* a tablet, a smartphone, a personal computer, or a laptop, among others), is in charge of building and managing the semantic map, while providing a virtual representation of it to support a friendly user interaction. The client, in its turn, runs on the robot itself (multiple clients can run at the same time in different robots) and aims at sensing the environment, detecting elements of interest in it (*e.g.* objects, rooms, etc.) and sharing such information with the server. An implementation of the contributed architecture, along with directions for installing and using it, has been made publicly available as a GitHub repository.[1]

On the one hand, the server resorts to ontologies to represent the model that accommodates the semantic information relevant to the problem at hand [24–26]. Ontologies are hierarchical representations that formally define the elements in the problem as concepts with properties, which are structured according to a subsumption ordering, *e.g.* microwaves are a subclass of appliances that have a box-shape (see Fig. 1). These structures contain all the knowledge of a collected semantic map. To measure the quality of a semantic map, *ViMantic* provides the functionality for computing a number of metrics when comparing two ontologies, where one of them could be an ontology codifying ground truth information. The server is implemented on Unity 3D [27], being the previous model a component of it (see Fig. 2). Unity 3D is a video game engine that allows us to build a 3D virtual environment incorporating the elements of the semantic map, which can be visualized by means of friendly graphical user interfaces that also enable interaction with them.

On the other hand, the client side relies on the Robot Operating System (ROS) framework [28], which provides off-the-shelf solutions for navigation, metric map building, gathering of sensory data, etc. To address the automatic population requirement,

[1] https://github.com/DavidFernandezChaves/ViMantic-Unity3DNode.

the client incorporates an object detection component for identifying the elements in the robot workspace, with a *Convolutional Neural Network* (CNN) at its heart [7,29], although it could be replaced or complemented by any other tool extracting relevant information from sensory data. Detected objects are packed with their location in the robot frame and shared with the server, which is in charge of their processing and integration into the semantic map.

In order to validate our proposal, we carried out different experiments with the Robot@Home dataset [30]. This repository was collected by a robot during a number of raids in different houses, including data from sensors typically found in robotic platforms like laser scanners and RGB-D cameras. In addition, the dataset provides reconstructions of such environments in the form of point clouds, which helps us to improve the visualization of different outputs from the proposed architecture (*e.g.* localization of the detected objects, assigned categories, etc.). This contributes to further validate the *ViMantic* suitability for the building and managing of semantic maps. We also describe two use cases showing different ways to exploit the information provided by these maps.

The following section puts our work into context with the related literature. Then, Section 3 generally describes the semantic map model adopted in this work. Section 4 explains the architecture components on the server side, while Section 5 describes those on the client side. We introduce the experiments carried out in Section 6, along with a discussion on the obtained results and two use cases exploiting semantic maps. Finally, Section 7 provides the main conclusions and achievements of the work presented in the paper.

2. Related work

Over the last decades, a menagerie of proposals for the building and utilization of semantic maps have appeared in the robotics field (see Table 1). Galindo et al. [10] presented one of the earliest and most influential works in this respect, in which they proposed a multi-hierarchical representation that relates the concepts included in an ontology with spatial elements obtained from sensors. Such a representation was adopted and exploited in posteriors works, like in Galindo et al. [6] for task planning, or in Galindo and Saffiotti [31] for autonomous goal generation. Later, this approach was extended by Ruiz-Sarmiento et al. [9], who presented the *multiversal semantic map* concept. In that novel model, each universe is a combination of possible links between the aforementioned hierarchies, which takes into account the uncertainty coming from processes involved in the map building (*e.g.* object detection or room categorization). The work by Zender et al. [32], contemporary of the one by Galindo et al. [10], proposed a similar approach with a single hierarchy. Such a representation codifies maps based on sensors' data and conceptual abstractions such as “Corridor”, “Kitchen” or “Coffee Machine”. The codification is done into an ontology by means of the Web Ontology Language (OWL). In such work, they resorted to a SIFT-based object recognition system to automatically populate the ontology. In this regard, other works proposed alternative methods such as classifiers using Convolutional Neural Networks (CNN) [16] or Probabilistic Graphical Models (PGMs) [39] to automatically populate the ontology, that is, without requiring human intervention during the process.

A significant number of papers in the literature have reckoned on ontologies as formal models to encode semantic knowledge exploitable by robots. For instance, Tenorth et al. [33] proposed a system called *KnowRob-map* that employs Bayesian Logic Networks (BLNs) to predict object types according to their description in an ontology (*e.g.* a flat surface with four legs, located in a kitchen, is probably a table). Pangercic et al. [34] explored the building of semantic maps of kitchens using an ontology to classify different types of furniture according to their physical characteristics. Interestingly, this work takes into account the handles observed in such pieces of furniture for their categorization, for example, a tall planar surface with two long handles is likely to be a refrigerator. Günther et al. [12] also categorized furniture according to its description in an ontology. In contrast to other works, the authors focused on the flat regions of the furniture and their relations. For example, a horizontal plane could be part of a chair or of table, but if it is related with a vertical one, the chair hypothesis gains strength. Another example is the research by Reinaldo et al. [35], where they proposed an intelligent navigation system based on recognized objects and their semantics. This system permits mobile robots to assume different behaviors according to the recognized objects and their properties. Other proposals, such as those by Pronobis and Jensfelt [11] or Qi et al. [36], combine ontologies with topological maps, enabling them to also classify areas in the environment according to their type (*e.g.* office, kitchen, corridor, etc.).

Less attention, however, has been given to the collaborative/distributed building and management of semantic maps. In this regard, Prestes et al. [14] proposed a centralized ontology where different robots or other intelligent agents could simultaneously add or query semantic information. As a consequence, the knowledge of an environment was available to every robot operating within it, avoiding the maintenance of duplicate data. Subsequently, Riazuelo et al. [15] expanded this concept, reporting an architecture called *RoboEarth Semantic Mapping* that uses a cloud ontology to encode semantic knowledge, while a Simultaneous Localization and Mapping (SLAM) algorithm is employed to build geometric maps. In this architecture, other agents can access the data in the cloud in order to retrieve relevant information to complete their tasks. At this point it is worth mentioning the comprehensive review conducted by Kostavelis and Gasteratos [22], where the authors surveyed available semantic mapping approaches for dealing with mobile robotic tasks.

As far as Human–Robot Interaction (HRI) is concerned, proposals such as that of Cosgun and Christensen [37], or the previously mentioned one by Zender et al. [32], consider human assistance during the semantic mapping. According to them, this allows us to avoid the utilization of object categorization algorithms, given the many challenges they entail. Another example including humans in the loop is the work by Bastianelli et al. [21], subsequently extended in Gemignani et al. [38], where authors describe an interactive semantic mapping approach that considers a person guiding the robot by voice commands. However, they themselves maintain that their proposal can be improved by means of the integration of state-of-the-art categorization techniques.

Recently, virtual environments have been uncovered as promising tools for HRI. In this sense, there are appearing preliminary works exploring their possibilities in robotics, as the one by Navarro et al. [18]. This work proposes virtual environments to perform an immersive teleoperation of robots. For that, the authors reconstruct the robot's 3D environment in a virtual space using point clouds obtained from RGB-D cameras. Users can interact and control the robot from the virtual environment, either through a screen or using a virtual reality device to immerse themselves in the virtual environment. Roldán et al. [17] is another example in this line, where the operation interface with the robot takes place in a virtual environment, also adding the possibility that different robots operate simultaneously in the same workspace.

As discussed, there is a large body of literature proposing semantic mapping models, architectures for their management,

Table 1
List of most relevant semantic mapping models/architectures proposed in the literature and the features they provide.

Proposal	Galindo et al. [10]	Ruiz-Sarmiento et al. [9]	Galindo et al. [6]	Galindo and Saffiotti [31]	Zender et al. [32]	Nüchter et al. [5]	Fernandez-Chaves et al. [16]	Tenorth et al. [33]	Pangercic et al. [34]	Günther et al. [12]	Pronobis and Jensfelt [11]	Reinado et al. [35]	Qi et al. [36]	Riazuelo et al. [15]	Prestes et al. [14]	Cosgun and Christensen [37]	Bastianelli et al. [21]	Gemignani et al. [38]	*ViMantic* (ours)
Formal model	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓				✓
Automatic population	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓
Distributed execution							✓							✓	✓				✓
HRI					✓											✓	✓	✓	✓
Public implementation																			✓

and ways to exploit them. However, semantic mapping techniques must exhibit a number of features in order to be flexibly integrated in modern mobile robotic systems (see Table 1). Our proposal, coined *ViMantic*, satisfies such needs by relying on a client–server architecture that includes a formal model for accommodating semantic knowledge, techniques for the automatic population of such model, distributed execution features, and a virtual environment-based HRI. *ViMantic* is publicly available, hence any interested researcher can benefit from it.

3. Grounding the semantic map concept

As commented, a semantic map is a representation of the robot workspace containing information at different levels of abstraction, ranging from the *low-level* (sensory data, *e.g.* images, point clouds, etc.), to the *high-level* (concepts such as `Table`, `Chair`, etc., as well as their properties and relations). Multiple proposals exist to accommodate and manage such information, although most of them make similar design decisions regarding the critical components of semantic maps [22]. In this way, we adopt here a consensus model consisting of: (i) a formal representation of the concepts in the domain at hand, and (ii) the linking of those concepts with spatial elements in the robot environment. Additionally, we augment these components with: (iii) a virtual model of the environment stating the (raw and/or processed) sensory information gathered from such world elements.

Fig. 1 yields a toy example of a semantic map and its components, where: the conceptual hierarchy corresponds to (i), the formal representation of the concepts; instances of concepts are linked to spatial elements, also including acquired knowledge like their position or size, hence modeling (ii); the virtual environment implementing (iii) incorporates the gathered and processed information from the spatial elements. The following sections describe how these semantic maps are built and managed in *ViMantic*, which is divided into two main components: the server, described in Section 4, and the client presented in Section 5. Remark that the proposed architecture has been carefully designed to cope with the aforementioned issues demanded to modern semantic mapping techniques: model definition, automatic model population, distributed execution, human–robot interface, and public availability.

4. Server side: representing, managing and exchanging information

The components on the server side are in charge of: (i) defining the formal model behind the semantic map, (ii) providing services for modifying/querying it according to requests from clients, (iii) offering a visual representation supporting the interaction with the user and (iv) comparing the semantic map with another previously built or ground truth information, which can be used to measure its quality. To develop those components we have resorted to Unity 3D, a popular video game engine. Among other features, Unity 3D offers tools to handle multiple multi-platform connections, work with three-dimensional models, or design visually appealing interfaces [17,18]. We have chosen this game development engine because, in addition to the mentioned features, it has a smooth learning curve and the community that supports it is overwhelmingly large, not only for game development, but also in other application areas, such as intelligent agents [27]. Moreover, this framework provides a repository of ready-to-use plugins implementing different algorithms, *e.g.* encapsulating Artificial Intelligence (AI) behaviors for agents, or managing collisions between virtual objects, which can be helpful in robotic applications.

The server is made up of a number of components, as shown in Fig. 2, which implement different functionalities and are able to freely exchange information among them. Briefly, its main components are: (i) the *General Manager*, which is in charge of managing the flow of data by creating connections with robots/agents, and also handles the system configuration (see Section 4.1), (ii) the *Object Manager* that processes the object detections coming from the robot workspace and creates or updates objects in a *virtual environment* (also called *virtual objects*) (Section 4.2), (iii) the *Ontology Manager*, which handles the model codifying the semantic information and provides access to it, for example, to accommodate the information coming from the previous component (see Section 4.3), (iv) the *Graphical User Interface* (Section 4.4) that supports human–robot interaction through buttons, messages, input fields, etc. and (v) the *Report Manager* (Section 4.5) which generates reports comparing the semantic map obtained at the end of a run with another semantic map given. The *WebSocket* standard [40] is used to communicate the components on the server side with those in the clients (robots and agents). Next sections give more details about these modules and the information they exchange.

The aforementioned *virtual environment* deserves further discussion, as it is at the core of Unity 3D (see the blue box in Fig. 2). This environment contains all the needed information to virtually represent the world being modeled. In our case, this includes representations of the knowledge acquired and produced by the robots (*e.g.* gathered images, laser scans, built geometric maps, detected objects, etc.), as well as semantic information (*e.g.* object labels and confidence scores).

Note that our proposal for autonomous semantic mapping works under the assumption that the world is static. Although there are many objects such as toilets, beds, sinks, etc. which are not usually moved, others such as chairs or flower pots can change their location. In this architecture, these changes can be managed through user supervision and the graphical interface.

The chosen modular design allows for the addition of new components or *plugins* in a straightforward way. Thereby, developers can make use of this mechanism to access the information in the semantic map and implement more complex and efficient behaviors for robots/agents. We provide more information about this feature in Section 4.6.

4.1. General manager

This component deals with general tasks in the virtual environment such as loading or saving configuration settings (*e.g.* path to the ontology file, semantic map identifier, different parameters for dealing with object detections, etc.), or handling the display/occlusion of the different menus in the interface (*e.g.* main view, settings window, or new connection window). Additionally, the *General Manager* also establishes connections between physical robots (or agents) and the server as demanded by the user.

To accomplish that, it is just needed to introduce the robot IP address. Then, the *General Manager* instantiates an avatar for that robot in the virtual environment (also called *virtual robot*). The virtual robot is set up according to its associated configuration settings (see Table 2), and a *WebSocket* connection is established linking it with its respective physical robot. From that point onwards, all the messages received from that IP address will be associated with this avatar, unambiguously identifying it and enabling the existence of multiple, simultaneous avatars working within the same semantic map.

Table 2
Server configuration parameters used in the experiments.

# Parameter	Value
Confidence score threshold for an object to be inserted	0.8
Maximum distance between the robot and the object to be inserted	2 m.
Maximum distance between two object detections sharing category to be merged	0.1 m.
Minimum number of object detections needed to insert it in the virtual environment	2

4.2. Object manager

The *Object manager* is the component in charge of processing the object detections originated from clients. These detections come in the form of messages, which can contain one or multiple detections, and that encapsulate: the objects' categories as predicted by the recognition system (see Section 5.1), their associated confidence scores, and their bounding boxes (defined by their spatial extensions and poses with respect to the robot).

Once a new message carrying object detections is received, the *Object Manager* transforms the local coordinates of the objects' poses, which are relative to robot poses, to their global coordinates in the reference system of the virtual environment. This is done by composing the pose O_R of each object with the pose R_V of its associated robot avatar in global coordinates, resulting in the global object pose O_V, that is $O_V = R_V \oplus O_R$. Once properly positioned, they are inserted in the virtual environment as 3D bounding boxes with their respective size (see green boxes in Fig. 8). These boxes are also called *virtual objects*.

In order to group detections belonging to the same physical object, we have exploited the physics provided by Unity 3D to detect when any of these virtual objects is close to another (as set by a given distance threshold) or even in contact. In case of collision, if they additionally share the same object category, we merge their information giving rise to a *virtual parent object* that encapsulates: the object category and the average confidence score, the number of virtual objects that have been merged (equivalent to the number of detections of the same physical object), and the union of the children bounding boxes.

Once the message containing such object detections is processed, the resulting information about new/updated virtual objects is sent to the *Ontology manager* (see Section 4.3). In this way, each virtual object is represented in the ontology as an instance of its associated category or concept. Both, virtual objects and instances, incorporate an unique identifier in order to unequivocally identify them during this information exchange.

4.3. Ontology manager

As previously mentioned, a semantic mapping architecture has to provide a formal model that accommodates the semantic information. For addressing this, we have resorted to *ontologies* [41]. An ontology is a formal representation of the knowledge concerning a domain of discourse through a number of *predicates* $\mathcal{O} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$. This representation usually takes the form of a hierarchy of *concepts* sorted according to a subsumption ordering, which is built using the `is-a` predicate. Examples of those concepts could be `Object`, `Appliance` or `Microwave`, and the aforementioned predicate is used to establish that `is-a(Object,Appliance)` and `is-a(Appliance,Microwave)`. Ontologies also include *instances* of those concepts, also called *individuals*, which represent abstract or physical elements within a certain environment. For example, when a new TV is detected, an instance of the concept `TV` is created and named `TV-1`. Such an instance can be further characterized through custom predicates, which are particular to each application. Examples of predicates could be `size`, which expresses the dimensions of an object (*e.g.* `size(TV-1,[0.2,0.2,0.3])`), or `score`, which codifies the certainty of the neural network about the TV detection (*e.g.* `score(TV-1,0.9)`).

To deal with ontologies within Unity 3D we have resorted to RDFSharp.[2] This is an open source C# framework which allows us to manage ontologies coded using the Web Ontology Language (OWL[3]). OWL is a language that aims to facilitate the representation and processing of rich and complex knowledge through the previously mentioned resources (concepts, instances, and properties). OWL ontologies can be codified by means of RDF[4], acronym for Resource Description Framework, a family of specifications from the World Wide Web Consortium (W3C) that standardizes the data coding for information exchange in the semantic web, and that is supported by RDFSharp.

In this way, we have designed an ontology containing a number of object categories/concepts of interest that can be found in human-like environments as offices or houses (see Fig. 3). These concepts, their properties and relations are also called *prior knowledge*, and has been acquired through human elicitation [9]. The construction of the proposed semantic map involves populating this model with the objects detected in the environment. For that, for each detected object, an instance of its respective concept is generated, *e.g.* `TV-1`, `Microwave-3`, `Table-2`, etc. These instances have associated properties specified by means of predicates: `identifier`, `position`, `orientation`, `size`, `score` (its confidence score) and `nDetections` (the number of times that the object it represents has been detected). It should be noted that, when an object previously perceived is detected again, the properties of its corresponding instance in the ontology are accordingly updated (recall Section 4.2). In its turn, for having a record of detections, each one is introduced as an instance linked to the first one by means of the `is-part-of` predicate.

Thereby, the ontology contains both, prior knowledge, and information acquired from the robot perception system and processed by the *Object Manager*. This model, codified through RDF, enables the execution of logical reasoning engines like Pellet [42] or FaCT++ [43] that can perform profitable tasks for an efficient robot operation. For example, they can check the consistency of the codified information [31], infer new information that is not explicitly provided, or perform high-level queries, *e.g.* finding an appliance able to keep food in good condition [9].

4.4. Graphical user interface

The proposed architecture also contemplates a mechanism for human robot interaction, aiming to enhance their collaboration. For that, it has been designed a Graphical User Interface (GUI) consisting of three main elements: (i) floating windows, (ii) the main interface, and (iii) the visualization of the virtual environment. The combination of these elements turns out to be a powerful tool for user interaction, allowing the storage and display of information within the semantic map.

The first of these elements encapsulates options that are set just one or a few times by the user, so there is no need to fix

[2] https://www.w3.org/2001/sw/wiki/RDFSharp.
[3] https://www.w3.org/TR/owl-features/.
[4] https://www.w3.org.

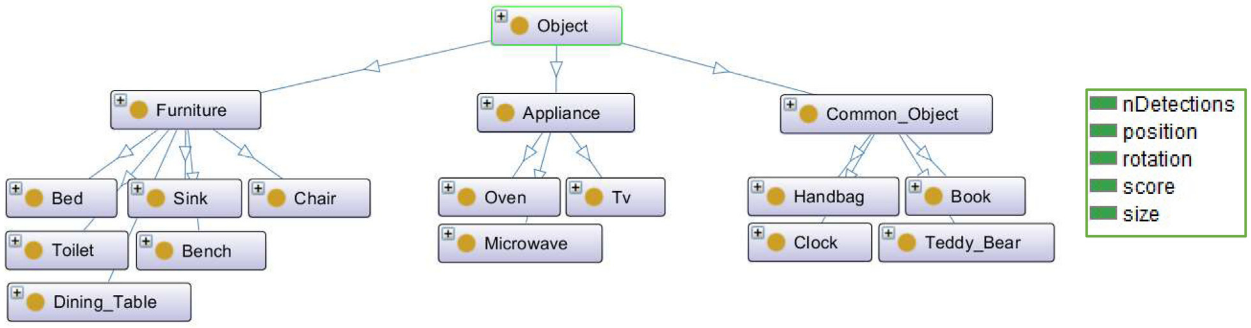
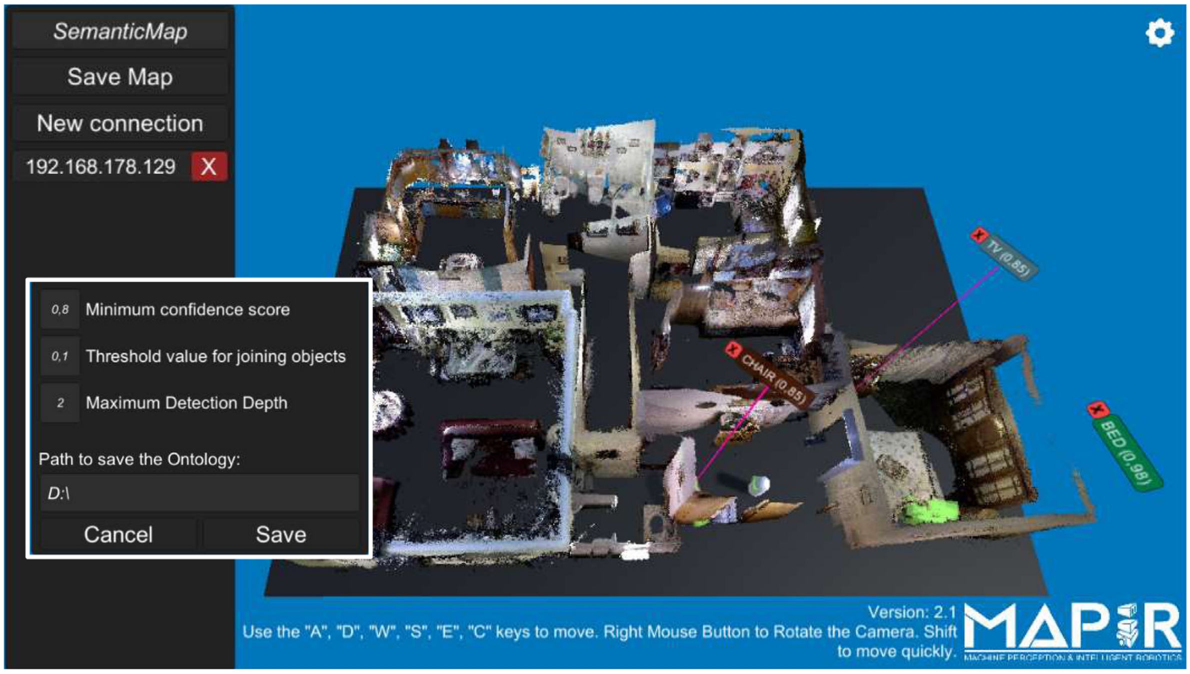


Fig. 3. Left, excerpt of the proposed ontology exhibiting a hierarchy of concepts. Right, properties associated with instances of those concepts (inside a green box).



0,8 Minimum confidence score
0,1 Threshold value for joining objects
2 Maximum Detection Depth
Path to save the Ontology:
D:\
Cancel Save

Version: 2.1
Use the "A", "D", "W", "S", "E", "C" keys to move. Right Mouse Button to Rotate the Camera. Shift to move quickly.

Fig. 4. Graphical User Interface designed to support human–robot interaction including buttons, text fields, and a visualization of the virtual environment with different mechanisms for its modification. Down left, a screenshot of the floating window permitting an advanced configuration of the system.

its visualization in the main screen. Currently there are two of these windows: a first one that permits the user to introduce the information needed to establish a connection with a physical robot (recall Section 4.1), and a second window showing a number of advanced configuration options of the architecture (*e.g.* confidence score threshold to process an object detection, distance threshold to consider that two detections belong to the same object, etc., see left part of Fig. 4).

The main interface consists of a side bar that provides different functionalities. First, it displays, in an editable text field, the name of the current semantic map. If this name is replaced by one belonging to an existing map, it is loaded in the architecture/GUI. Two more buttons enable the user to save the map to disk and to open a new robot connection window. Below them, it is shown a list with the currently connected robots/agents, also permitting the user to cancel those connections. Besides, a small button is also displayed on the upper right side of the GUI, providing access to the aforementioned system configuration window.

As for the interface with the virtual environment, its permits the visualization of pieces of the knowledge acquired from the workspace, and also enables some ways to interact with it. For example, Fig. 4-right shows virtual objects placed in a previously built point cloud representation of an environment. Each virtual object is linked to a small floating panel with a button and a label. The button permits the user to remove that object from both the virtual environment and the ontology (see Section 4.3), thus eliminating the robot's knowledge associated to it. Regarding the label, it shows the category and confidence score of the represented object, helping the user to review at a glance relevant parts of the knowledge acquired from the workspace.

4.5. Report manager

Due to the lack of standard indicators to assess the quality of a semantic map, *ViMantic* integrates a report manager that carries out this task. Concretely, this module compares the ontologies codifying the information within two different semantic maps (see Section 4.3). Since the ontologies populated with *ViMantic* are formal representations of the information contained in semantic maps, in this context comparing ontologies is equivalent to comparing semantic maps. This enables the evaluation of the quality of a generated map w.r.t. the ground truth, or even its relative comparison with a previously built one. The reports generated supply the following general results: (i) number of detections, (ii) number of right objects in the map, (iii) number of wrong objects in the map, (iv) number of detected objects, and (v) number of undetected objects. Notice that (iv) fuses possible multiple detections of the same object in (i). For an object prediction to be considered right, there has to be an object of the same type in the reference semantic map whose distance between the nearest points of their bounding boxes is lower than a given threshold. In addition, in order to provide further debugging capabilities the report includes, for each predicted object in the first semantic map, its properties and the distance to the nearest same-type object in the second one.

4.6. Plugins

Plugins are optional components that can be added to the architecture to exploit and/or expand its functionality. In this way, any instance of *ViMantic* could include an arbitrary number of additional plugins, depending on the requirements of each particular application.

An example of optional component adding extra features could be the "Robot@Home plugin", which permits us to load information of houses from the Robot@Home dataset [30]. This information includes point clouds representing those houses, which are embedded in the virtual environment and graphically shown, helping the user to understand the data acquired by the robot through an immersive experience (recall Fig. 4).

Another example of useful plugin could be an "Object Finder". Let us suppose a scenario where a robot operating in a house receives the order *"bring me a bottle"*. In this context, a plugin could query the *Ontology Manager* the position of previously detected bottles, and send the location of the closest one to the robot in order to navigate there. A more sophisticated approach could employ the detection score associated to each object instance to optimize the possibility to truly find a bottle [44]. If no bottle appears in the semantic map, the plugin could also query their most likely positions (*e.g.* over planar, horizontal surfaces like tables and counters) and command the robot to visit those promising locations to find them [6].

It should be noted that Unity 3D can host complex systems to control robots, hence turning them into mere task executors. For example, plugins could implement a high-level decision-making and task execution system, with access to the available semantic knowledge and capable of controlling the actions of one or more robots [6]. Continuing with the previous example, a plugin could decompose the *"bring me a bottle"* command into a number of simpler tasks affordable by the robot (*e.g.* navigation, fetch and carry, etc.) and, in applications with multiple robots, it could also optimally assign such tasks to robots according to their position, battery level, etc. (see Section 6.4.2).

5. Client side: robots sensing and acting

This section describes the side of the architecture to be run on robots/agents, aiming to support and empower their sensing and actuation skills. Its core components are shown in Fig. 5. For their development we have relied on the widely used Robot Operating System (ROS) [28]. ROS is an open-source collection of tools, libraries and conventions that simplify the task of building complex and robust robotic behaviors, being the default choice in the robotics community for software developing.

Briefly, the *Object Recognition* component is in charge of sensing the robot's surroundings and detecting objects within it (see Section 5.1). The output of this component feeds the *Object Information Packer* one, which packs relevant information about each detected object (Section 5.2). Such information includes: object category, size, 3D position w.r.t. the robot frame, orientation, and confidence score. The resultant packages are sent to the server via *WebSocket* communication [40], which is managed

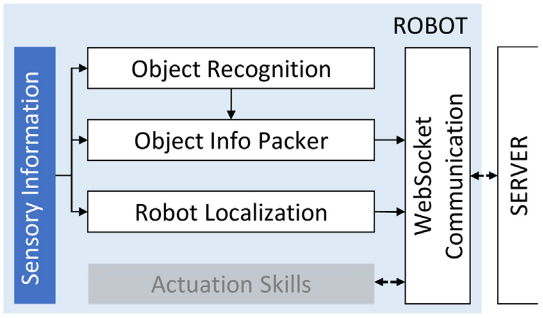


Fig. 5. Overview of the proposed client side of the architecture showing its main components and their interconnections. White boxes are components, while the blue one represents sensors capturing data.

by the *rosbridge_suite* package.[5] In its turn, the *Robot Localization* component is responsible for localizing the robot within a previously built geometric map and sharing such information with the server (see Section 5.3). Finally, and depending on the capabilities of each robot/agent, the architecture also considers an *Actuation Skills* component that permits it to carry out action orders as commanded by the server, *e.g.* navigation to a given location in the map, fetch and carry an object, etc. (Section 5.4). Next sections further describe these components.

It is worth mentioning that, although in our discourse we keep the spotlight on robots, the architecture is designed to accept/provide information from/to any smart device connected to the internet and instantiating a *ViMantic* client. This means that any intelligent agent (running such a client) like smart sensors (*e.g.* IP cameras, presence or humidity sensors, etc.) or devices like intelligent light bulbs, smart TVs, smartphones, or tablets, could generate new information to be inserted in the semantic map, or perform queries about its content.

To enable the implementation of this architecture in a practical way, we have created a library that implements the communication of many of the common message types in ROS. This library can also be used as a template for implementing other custom messages needed in specific applications. The interested reader can find it at: https://github.com/DavidFernandezChaves/ROSUnityCore.

5.1. Object recognition

The purpose of this component is the detection of objects in the robot's workspace from the information provided by its onboard sensors. *Convolutional Neural Networks* (CNNs) have proved to be particularly useful for this purpose in multiple object detection challenges, *e.g.* PASCAL [45] or COCO [46]. This is mainly due to their robustness against challenging factors like changing lighting conditions, occlusions, varying viewpoints, or high intraclass variability (objects belonging to the same category but exhibiting different shapes, colors, sizes, etc.).

The progress in object detection networks is vertiginous, appearing novel designs every year improving the performance of previous works. There are a number of well known networks providing a high detection success, like YOLOv3 [47], Faster R-CNN [48], or Mask R-CNN [49]. In this work we opted for Detectron 2 [50], which integrates an improved version of Mask R-CNN, and that achieved a notorious performance in our previous research [13,16]. The input of this CNN is an intensity image (RGB), and the output is a set of detected objects. Specifically, a detected object includes: the object category, a bounding box containing

[5] http://wiki.ros.org/rosbridge_suite.

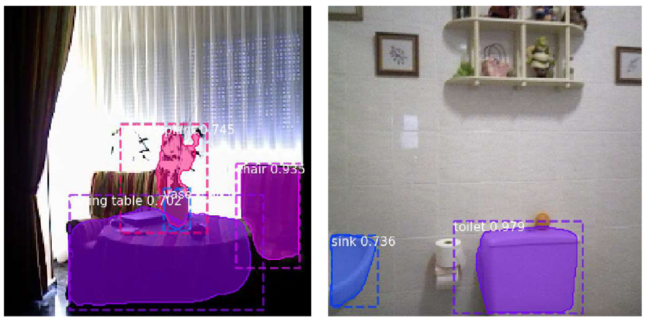


Fig. 6. Object detections as reported by Detectron 2, part of the proposed recognition component. On the left, the CNN recognized a dining table with a confidence score of 0.702, a partially observed chair (0.935) and, even under bad lighting conditions, a potted plant (0.745). On the right, it detected a partially observed sink as well as a toilet with 0.736 and 0.979 confidence scores, respectively.

the object, a mask over the pixels in the image belonging to the object, and a confidence score codifying how confident the network is about the detection. Fig. 6 shows some examples of these detections, illustrating the potential of these network for successfully recognizing objects. To incorporate this CNN into ROS we have created a Detectron2 wrapping node, which is publicly available for any interested reader.[6]

Aiming to be as modular as possible, the architecture has been designed in such a way that the neural network used by the *Object Recognition* component can be replaced by any other technique consuming sensory information and producing a list of detected objects without affecting the other components.

5.2. Object information packer

In order to insert the detected objects in the semantic map, it is needed to transform the output from the CNN, expressed in the 2D image plane, into the robot three-dimensional coordinate frame. Once this transformation is done, the detected object can be placed in the map as described in Section 4.2. The *Object Information Packer* is in charge of doing that by: (i) retrieving the spatial extensions and 3D poses in the robot frame, (ii) packaging together the derived information from each object detected in the same frame, and (iii) sending it to the server. This component defines how and which information is forwarded to the server, helping to the previously mentioned modularity.

To calculate the 3D pose and spatial extension of each object, the *Object Information Packer* transforms the masks of pixels received from the CNN into point clouds. For that we rely on sensory information coming from RGB-D cameras, which provide depth information of the scene in addition to intensities. However, in cases where only intensity information is available, novel techniques are appearing that estimate 3D point clouds or meshes of objects from such data [51]. Let us define the coordinates of a pixel in the intensity image belonging to the mask of a detected object as $p = [u, v]$. Then, the intrinsic parameters of the RGB-D camera can be used to obtain the coordinates of its corresponding 3D point in the sensor frame, that is $P_S = [X_S, Y_S, Z_S]$ (see Zuñiga Noël et al. [52] for more details). Once expressed in the sensor frame, such a point can be transformed into the robot frame by means of the sensor extrinsic parameters, obtaining $P_R = [X_R, Y_R, Z_R]$ [53]. This process is repeated for each pixel in the mask, resulting in a point cloud representing the object. Such point cloud is further processed to remove spurious points as well as erroneous points that do not belong to the object, typically caused by objects with holes or inaccurate object masks. Thus, a filter is applied that removes points not satisfying the condition: $\mu - 2\,\sigma < Z_R < \mu + 2\sigma$, where μ is the mean depth of the point cloud and σ its standard deviation. The remaining points represent the space occupied by the object, and are used to retrieve the position of the object centroid in the robot frame, as well as to fit a 3D bounding box delimiting its extension.

Once the objects detected within an image have been processed, the *Object Information Packer* creates a package containing their categories, 3D poses, spatial extensions and confidence scores. This package is sent to the server in order to be processed, hence fully incorporating the gathered information into the semantic map (recall Section 4.2).

5.3. Robot localization

Another type of information needed to properly build the semantic map is such of the localization of the robot at each time instant. Such localization is expressed w.r.t. a given geometric map of the workspace, and permits the server to properly locate the robot avatar in the virtual environment (recall Section 4.1) as well as the detected objects (Section 4.2). The building of the geometric map is out of the scope of this work, but it is worth mentioning that ROS offers tools for that, like *gmapping*,[7] a ROS wrapper for OpenSlam's Gmapping [54].

Robot localization is a widely researched topic in the robotics community, and ROS provides robust localization packages already built in. Concretely, we have resorted to the *AMCL* package,[8] which implements a localization technique based on the popular Adaptive Monte Carlo Localization method proposed by Fox [55]. Such package relies on measurements from sensors typically mounted on mobile robots: 2D laser scanners, to locate the robot within a previously built geometric map. In this way, in the context of the proposed architecture, laser scans are processed by such package and the obtained robot locations are sent to the server through the *WebSocket* connection.

5.4. Actuation skills

The previous sections describe components where the robots/agents play the role of sensors: they gather raw/processed information and send it to the server. However, agents could also perform actions in the environment, and the *Actuation Skills* component is in charge of encapsulating them. The content of this component is agent-specific, since it depends on their capabilities.

In the case of mobile robots, their essential capability is to navigate, but they could be also able to fetch and carry objects, interact with other devices (*e.g.* pushing a button), play sounds (including words), etc. [24,56]. For that they are needed motorized wheels, robotic arms, speakers, etc., which can be present or not in a given robotic platform. In this way, the *Actuation Skills* component acts as a bridge between the server and the available actuators.

Although the client side could incorporate components to endow the robot to perform tasks of certain complexity, in our proposal this is left to the server side. The server, through the aforementioned plugins, implements the needed logic so a robot could carry out the needed high-level tasks for the application at hand. An example of this is the "Object Finder" plugin (recall Section 4.6), which sends navigation commands to the *Actuation*

[6] https://github.com/DavidFernandezChaves/Detectron2_ros.

[7] http://wiki.ros.org/gmapping.

[8] http://wiki.ros.org/amcl.

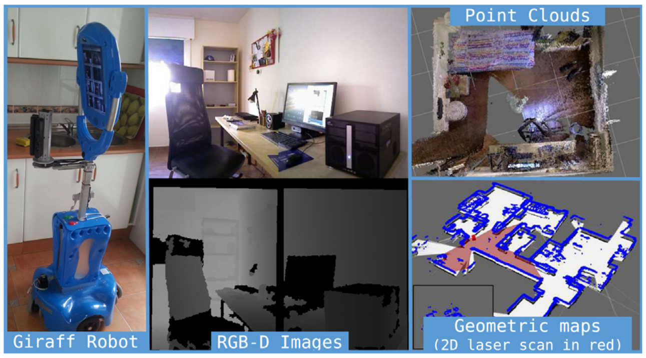


Fig. 7. Mobile robot used to collect the Robot@Home dataset along with some samples of the data it provides.

Skills component in order to find a given object. Another example could be a plugin in charge of gaining in confidence about uncertain object recognition results. Such a plugin could query about the low-scoring detections to the ontology, and send navigation commands to the robot in order to revisit them. Looking at those objects from different points of view clearly helps to disambiguate the validity of their detections.

6. Evaluation

The purpose of this section is to demonstrate the suitability of the proposed architecture for the building of semantic maps. For that, we have carried out a number of experiments using the *Robot@Home* dataset as testbed (see Section 6.1). Such a data repository permits us to consider one or multiple robots collecting data from the same environment, hence enabling the testing of multi-agent scenarios. Section 6.2 describes how the different components/parameters in *ViMantic* have been set up to perform such experiments. In Section 6.3 we comment on the conducted experiments as well as on the reported results. Finally, Section 6.4 discusses two possible use cases of semantic maps built with our architecture.

6.1. Dataset: Robot@Home

Robot@Home [30] is a publicly available repository[9] of raw and processed data collected by a mobile robot Giraff [57] while visiting cluttered houses (see left part of 7). For collecting the dataset, Giraff was equipped with a rig of 4 RGB-D cameras (model Asus XTion Pro Live) and a 2D laser scanner (model Hokuyo URG-04LXUG01). Those sensors gathered 87,000 raw observations divided into 83 sequences (see central part of 7). From them, we have selected the sequences where the robot fully visited four different houses, since they allow us to deeply test our proposal. Additionally, we only considered the images gathered by the RGB-D camera looking ahead, given that it is a more common configuration in robotic platforms. To experiment with realistic sequences of robot operation, we take advantage of the fact that the dataset sequences are also available in the timestamped *rosbag* format, so by means of the *rosbag package*[10] the sequences can be reproduced making sensors' data available at the right time.

Regarding the processed data in the dataset, it includes 2D geometric maps and 3D reconstructions of the visited houses (see right part of 7), both annotated with ground truth categories of the objects appearing therein, as well as the categories of the inspected rooms. Specially relevant here are the 3D reconstructions since, as commented in Section 4.6, they are inserted in the virtual environment so it looks more appealing to users.

6.2. Experimental setup

In order to employ *ViMantic*, some of its components must be instantiated and configured. This includes: the CNN in the *Object Recognition* module, the ontology in the *Ontology Manager*, and the configuration parameters in the server.

Regarding the CNN, we have opted for an instance of *Detectron 2* pre-trained with the *COCO* dataset [46]. Such a dataset includes categories of everyday objects typically found in houses like chair, sofa, potted plant, bed, dining-table, toilet, or tv, among others. As for the ontology, the same object categories considered in *COCO* for indoor environments where converted to concepts and, by human elicitation, classified into three major groups: `Furniture`, `Appliance` and `Common object`. Fig. 3 shows the resultant hierarchy of concepts, as well as the properties used to describe each object instance.

The configuration parameters in the server have also to be fixed. The values used in these experiments are shown in Table 2. They were chosen empirically to: (i) notoriously reduce the number of wrong object detections that result in virtual objects' instances, and (ii) increase the number of detections that are successfully merged. It is worth mentioning that every object detection is recorded in the ontology, independently of whether a virtual object is instantiated or not. It is also specific to each application in which device the server is executed. In the experiments described below it was launched in a computer external to the robots, following the idea of *edge computing*.

In this way, two computers were used during the experiments. The first one, running the server side of *ViMantic*, has an Intel Core i7-5700HQ processor at 2.70 GH, a RAM memory with 2 $\times$ 8 GB DDR3 at 800 MHz, and a graphic card NVIDIA GeForce GTX 960M with a memory of 2 GB. The second computer, running one or multiple instances of the client side of the architecture, is equipped with an Intel Core i7-8750H processor at 2.20 GH, a 2 $\times$ 8 GB DDR4 RAM memory at 1333 MHz, and a graphic card NVIDIA GeForce GTX 1070 with a memory of 8 GB.

6.3. Experiments: putting ViMantic to work

The following sections describe the experiments carried out to validate the instantiated *ViMantic* architecture, which also help us to illustrate its *modus operandi*. Concretely, we have conducted an experiment where a robot explores different houses and, working with a server, builds their respective semantic maps (see Section 6.3.1). In a second experiment, two robots collaborate in the building of such maps (Section 6.3.2). Section 6.3.3 reports and discusses on the obtained results.

6.3.1. Experiment one: building maps with a single robot

The first experiment considers a scenario where a single mobile robot instantiating the *ViMantic* client visits four houses from Robot@Home, namely *alma*, *anto*, *pare* and *rx2*. In each of these houses, the robot navigates until every room is visited while gathering both RGB-D images and 2D laser scanners. It is worth mentioning that the robot has no other purpose than to wander and passively capture data. On the one hand, laser scans are used by the *Robot Localization* component to locate the robot within the provided geometric map of the house (recall Section 5.3). On the other hand, RGB-D images are processed by the *Object Recognition* component in order to detect the objects appearing in them (see Fig. 6), while the *Object Information Packer* extracts

9 http://mapir.isa.uma.es/work/robot-at-home-dataset.

10 http://wiki.ros.org/rosbag.

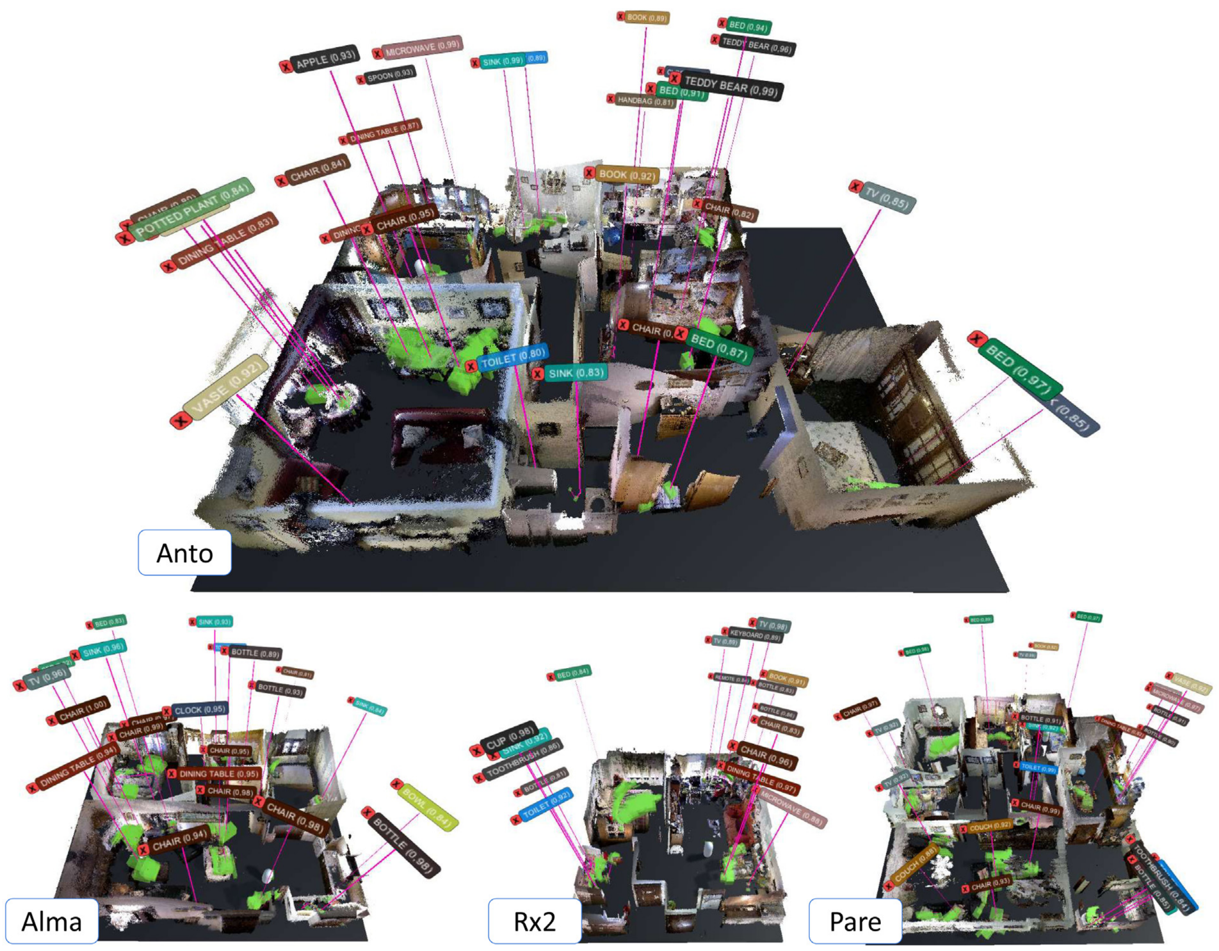
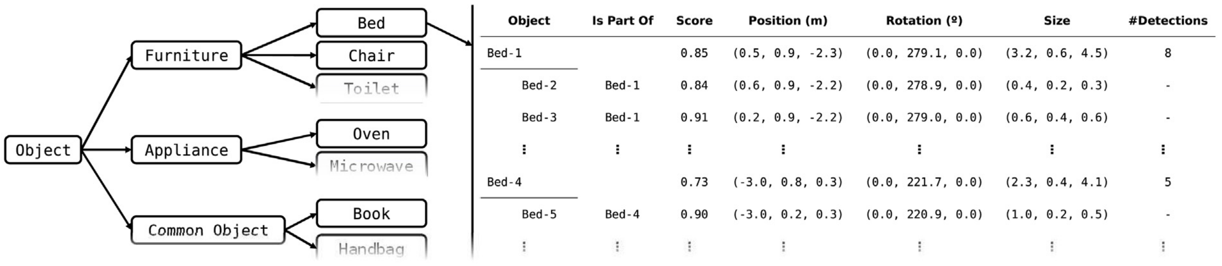


Fig. 8. Views of the virtual environments obtained after the inspection of four houses from the Robot@Home dataset.



Object	Is Part Of	Score	Position (m)	Rotation (º)	Size	#Detections
Bed-1		0.85	(0.5, 0.9, -2.3)	(0.0, 279.1, 0.0)	(3.2, 0.6, 4.5)	8
Bed-2	Bed-1	0.84	(0.6, 0.9, -2.2)	(0.0, 278.9, 0.0)	(0.4, 0.2, 0.3)	-
Bed-3	Bed-1	0.91	(0.2, 0.9, -2.2)	(0.0, 279.0, 0.0)	(0.6, 0.4, 0.6)	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Bed-4		0.73	(-3.0, 0.8, 0.3)	(0.0, 221.7, 0.0)	(2.3, 0.4, 4.1)	5
Bed-5	Bed-4	0.90	(-3.0, 0.2, 0.3)	(0.0, 220.9, 0.0)	(1.0, 0.2, 0.5)	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 9. Excerpt of the content of an ontology after the inspection of the *anto* house by a mobile robot (instantiating a *ViMantic* client). The whole hierarchy of concepts was shown in Fig. 3.

additional information like their spatial extensions and poses (Section 5.1). Then, robot locations and packed detections are sent to the server for their integration in the semantic map.

Then, such information is processed in the server (configured as described in Section 6.2) by: the *General Manager*, which synchronizes the robot avatar location with the arriving ones, and the *Object Manager*, which accordingly creates and updates virtual objects in the virtual environment. Fig. 8 shows the resultant virtual environments of the four houses as visualized in the graphical user interface, populated with the detected objects and the robot avatar (represented by a white oval). Thereby, the user can view at a glance relevant parts of the semantic map and interact with it. Recall that each object detection processed by the *Object Manager* is sent to the *Ontology Manager* for its inclusion in the ontology, even those with a confidence score under the considered threshold, or those that have not yet been detected an enough number of times to be inserted in the virtual environment. Fig. 9 depicts an excerpt of the ontology content once the robot visited the *anto* house, showing on the right side a number of instances of object detections as provided by the *Object Manager*.

Thus, each semantic map built, as defined in Section 3, is composed of: the formal representation of the concepts in the home domain, the linking of those concepts with the spatial (detected) elements in the house, and the virtual model of such environment. Section 6.4 provides some use cases taking advantage of these maps.

6.3.2. Experiment two: collaborative building

This second experiment aims to illustrate the building of semantic maps by means of two agents/robots. This possibility is given by the distributed nature of the architecture. For that,

Table 3
Results obtained from the conducted experiments concerning ontologies and virtual environments.

		Ontology		Virtual environment	
		Size (KB)	# Instances	# Objects	Success
1 Robot	anto	1.082	1.845	29	89,66%
	alma	693	1.166	21	76,19%
	rx2	543	846	22	77,27%
	pare	1.006	1.654	40	85,00%
	Average	831	1.378	28	82.03%
2 Robots	anto	1.346	2.357	35	88,57%
	alma	985	1.685	35	80,00%
	rx2	725	1.177	33	75,76%
	pare	1.072	1.844	51	90,20%
	Average	1.032	1.766	39	83.63%

the same four houses are considered, where two instances of the *ViMantic* client are executed into two different robots. These robots start moving in the same houses at different time instants, and keep navigating until every room is visited. Since both robots are instances of the robot that performed the data collection in Robot@Home, they both follow the same trajectory. To obtain new information with the second robot, we have used the data from the camera looking 90° to the right, resulting in a different point of view of the same scene w.r.t. the first robot.

As before, the first robot considers the RGB-D images coming from the camera looking ahead to feed the *Object Recognition* component. However, for the second one, instead of such a camera it is considered the one on its right, so objects are detected from different viewpoints. Again, 2D laser scans are used to localize robots, sending both clients their locations along with the detected objects to the server, which processes them by means of the *General*, *Object* and *Ontology* managers. The interested reader can check the following video, which illustrates part of the semantic map building process carried out in the *Anto's* house during this experiment: https://youtu.be/3MZgAxxBtKY.

6.3.3. Results

This section discusses the results obtained from the previous experiments. As commented, the built semantic maps consist of links between spatial elements (objects) and concepts defined on an ontology, the own ontology formally defining the knowledge within the home domain, and the representation of those elements in a virtual environment. To evaluate the suitability of such maps, we have considered different aspects like the required size in memory of the final ontology, the performance achieved by object detection-related components, or the computational time demanded by the *ViMantic* critical components. The results shown have been obtained using the Report Manager module, which compared the ontologies obtained after each of the different experiments with ontologies encoding ground truth information. Next paragraphs go into depth on them.

Ontology analysis. The ontology is the core of the semantic map. Having a file encoding it, it is possible to restore a previously built map by loading its ontology through the *ViMantic* graphical user interface, since it also contains the needed information for recovering the virtual objects in the virtual environment. This way, it is relevant to spend some lines analyzing how its size in memory behaves depending on the workspace dimensions. As a starting point in this analysis, the ontology codifying the concepts and their properties has a size of 48 KB. Regarding the visited houses, *anto* and *pare* are large ones, with two bathrooms each, a kitchen, spacious living rooms, and four rooms (master rooms, bedrooms and dressing rooms), while *alma* and *rx2* have a single bathroom, two and one bedrooms respectively, and open concept kitchens-living rooms. Generally, the bigger the space, the more objects appear in it.

Table 3 reports the sizes in memory of the ontologies created in the previous experiments. We can see how such sizes are in line with the houses' descriptions. The lightest ontology is the one built in the *rx2* house, with a size of $\sim$0.5 MB and storing more than 800 instances. The heaviest one, built in the *anto* house in the two robots scenario, exhibits a size of $\sim$1.3 MB and contains more than 2300 instances. We can also check that, on average, an instance requires just $\sim$0.56 KB to be allocated in memory. This is a reduced size enabling the architecture operation in scenarios with thousands of instances, since ontologies efficiently codify such information.

However, for applications with even larger environments, long term operation requirements, or devices with very constrained memory resources, maintenance mechanisms could be implemented if needed to keep affordable the ontology size. For example, similar information could be merged or deprecated knowledge removed [9].

Results of object detection-related components. The performance of the object detection-related components within *ViMantic* is critical for the building of suitable semantic maps. To rely on an object detection system with a low recognition success would lead to unreliable object instances in the ontology and virtual objects in the virtual environment, hence clouding the interaction with the user.

Table 4 shows the performance of these components using different state-of-the-art CNNs when running in the four houses of the first experiment. An object inserted in the semantic map has been considered as right if there is an object in the ground truth of the same type whose distance between the nearest points of their bounding boxes is less than 20 cm.

CenterNet [61] with 240 detected objects was the network with the most detections, while Detectron2 [50] with 206 and an average precision of 0.83 achieved the best trade-off between success and number of detected objects. The number of total objects yielded by each CNN is a good indicator of its execution rate and how prone it is to detect erroneous or poor objects (with few detections). For example, Faster-RCNN [60] runs fast and produced 201 objects in total, but after filtering such objects only 14 of them were considered in the semantic map. Note that objects with multiple detections are more likely to be detected when their associated bounding boxes are large, so CNNs that return small detections (mostly due to blurred images, light effects and other problems) are more likely to perform worse.

It is important to point out that the number of detected objects shown in Table 4 is filtered by the confidence threshold for a detection to be inserted in the map (in our case 0.8). Since some CNNs are more conservative than others, success could be increased by lowering this threshold, especially for those CNNs that have obtained a low average f1 score (*e.g.* Faster-RCNN with an average f1 score of 0.19), as this implies that they have a high recall.

We observe that the objects successfully recognized with a single detection do not represent 10% of the total number of objects recognized in any case, being Faster-RCNN [60] with 9.95% the CNN with more objects successfully detected in this respect. Moreover, in all cases these objects represent the highest percentage of wrong detections with respect to the total number of them. However, as the number of detections increases, so does the percentage of them that are right. In all cases, objects with more than 9 detections have the highest performance.

The reason for this is that single detections tend to occur in blurred images or abrupt lighting changes, and such artifacts are no further detected in the next frames by the CNN. This way, we found it useful to set a threshold (γ) to the number of times that

Table 4
Results obtained from the first experiment using five different state-of-the-art CNNs. The *Objects* row yields the total number of reported objects in all the considered houses, grouping them according to the number of times that they have been detected. In order to compare how profitable each group of objects is for the global performance, the percentage of right/wrong objects (computed by dividing the number of right/wrong objects in a given group by the total number of detections in all groups) is provided in the rows % Right objects w.r.t. the total *(%ROT)* and % Wrong objects w.r.t. the total *(%WOT)*. The final three columns on the right show the success, average precision and average f1-score obtained respectively in the semantic maps after filtering out objects with two detections or less (*gamma* = 2).

		1	2	3	4	5	6	7	8	9	>9	Total	Semantic map success ($\gamma = 2$)	Average Precision	Average f1-score
Detectron2 [50]	Objects	107	24	18	9	5	8	3	3	4	25	206	**75**		
	%ROT	7,28%	5,83%	6,80%	3,88%	2,43%	2,91%	0,97%	0,49%	1,46%	11,17%	43,20%	**82,67%**	0.83	0.73
	%WOT	44,66%	5,83%	1,94%	0,49%	0,00%	0,97%	0,49%	0,97%	0,49%	0,97%	56,80%	**17,33%**		
Yolo3 [58]	Objects	98	39	10	11	6	1	7	1	0	10	183	**46**		
	%ROT	7,10%	6,56%	4,92%	4,37%	2,73%	0,55%	2,73%	0,55%	0,00%	4,92%	34,43%	**82,61%**	0.85	0.53
	%WOT	46,45%	14,75%	0,55%	1,64%	0,55%	0,00%	1,09%	0,00%	0,00%	0,55%	65,57%	**17,39%**		
SSD [59]	Objects	115	22	12	9	3	3	0	1	1	4	170	**33**		
	%ROT	2,94%	3,53%	5,29%	2,94%	1,18%	1,18%	0,00%	0,59%	0,59%	2,35%	20,59%	**72,73%**	0.77	0.53
	%WOT	64,71%	9,41%	1,76%	2,35%	0,59%	0,59%	0,00%	0,00%	0,00%	0,00%	79,41%	**27,27%**		
Faster-RCNN [60]	Objects	174	13	3	4	3	0	2	0	1	1	201	**14**		
	%ROT	9,95%	2,49%	1,00%	1,99%	1,00%	0,00%	1,00%	0,00%	0,50%	0,50%	18,41%	**85,71%**	0.92	0.19
	%WOT	76,62%	3,98%	0,50%	0,00%	0,50%	0,00%	0,00%	0,00%	0,00%	0,00%	81,59%	**14,29%**		
CenterNet [61]	Objects	195	17	11	6	4	0	2	0	0	5	240	**28**		
	%ROT	5,83%	2,50%	3,75%	2,50%	1,25%	0,00%	0,42%	0,00%	0,00%	2,08%	18,33%	**85,71%**	0.91	0.33
	%WOT	75,42%	4,58%	0,83%	0,00%	0,42%	0,00%	0,42%	0,00%	0,00%	0,00%	81,67%	**14,29%**		

an object has to be detected in order to be inserted in the virtual map, where the user can interact with it (recall Table 2).

In Table 4, we can check the success of the semantic map building when filtering out objects that received only one or two detections ($\gamma = 2$). Despite the differences found between the networks, the final success rate of the maps is quite solid at around ∼82% without user influence. It is worth mentioning that human revision, despite the high reliability of the information provided by the semantic maps in *ViMantic*, could help to further improve their quality.

Notice that, although not shown in the virtual environment, every detection is stored in the ontology so new detections of previously observed objects can be identified. It is worth noting that such a threshold also disregard a small number of right detections. Thereby, it sets a trade-off between the number of wrong detections that are visualized in the virtual environment, as well as those right that are omitted.

In the same way, *ViMantic* uses this parameter and those previously mentioned (recall Section 5.2 and Section 4.2) to modify the level of filtering of the detections, allowing to build very populated semantic maps with less success rate, or to increase the success rate at the cost of detecting fewer (but probably true) objects. Nevertheless, the success of the semantic map building could be further improved by increasing the amount of information obtained, for example by using dedicated active perception systems, by revisiting rooms to certify their knowledge, or by adding multiple cameras. At this point it is worth recalling that, as mentioned in Section 4.4, the user can review the semantic map at any moment and remove wrong detections. This process could be enhanced by the automatic proposal by *ViMantic* of uncertain detections to be reviewed (*e.g.* those with score under a certain threshold).

By setting the threshold γ to 3, we obtain the results shown in Table 3. This table reports, for the two conducted experiments, the number of objects inserted in the virtual environments modeling each house along with the percentage of right detections. The achieved success is remarkably high in both cases, ranging from ∼76% to ∼90%. When considering two robots, the number of objects in the virtual environments significantly increases (11 new objects on average), while the percentage of right detections remains similar or slightly increases—the average improvement is

Table 5
Computational time required by the critical components of *ViMantic*.

Client			
Object recognition		Object info packer	
Avg. (ms)	Std (ms)	Avg. (ms)	Std (ms)
105.16	29.99	1.59	0.80
Server			
Object manager		Objects union	
Avg. (ms)	Std (ms)	Avg. (ms)	Std (ms)
2.98	8.35	5.03	6.06

of 1.60%. This is mainly due to the detection by the second robot of objects that went unnoticed by the first one, since it inspected the houses from a different point of view, or to additional detections of a previously observed object but that did not overcome the threshold γ. In conclusion, the collaborative building of the semantic map proved to be profitable for achieving more right detections, and leads us to believe that a similar effect could be achieved by further explorations of the houses by a single robot.

Analysis of computational time. Another factor worth studying is how efficient *ViMantic* is, that is, the time needed to process new information and incorporate it into the semantic map (see Table 5). In this respect, it is interesting for the architecture to build the semantic map in real time, that is, it should be able to process arriving information at the same (or at a higher) rate at which it is available. In our case, such information comes from the RGB-D cameras and the 2D laser scanner.

Regarding laser scans, the *Robot Localization* component is able to process information coming from sensors with a high frequency (*e.g.* 40 Hz) to estimate the robot's pose. In our case, the used laser scanner has a working frequency of 10 Hz. From the server side, new robot locations just imply the update of the robot avatar position, so the information coming from the 2D laser scanner does not impose limitations for such real time operation.

As for RGB-D images, the *Object Recognition* component spends on average 105 ms detecting objects. Then, the *Object Information Packer* extracts additional information and sends it to the server, requiring less than 2 ms for that. Once in the server, the *Object*

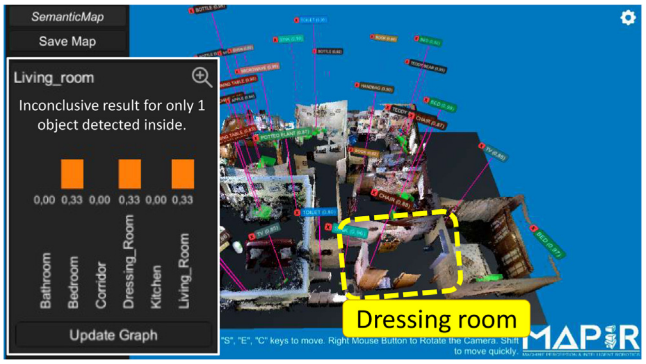

SemanticMap
Save Map
Living_room
Inconclusive result for only 1 object detected inside.
0,00 0,33 0,00 0,33 0,00 0,33
Bathroom
Bedroom
Corridor
Dressing_Room
Kitchen
Living_Room
Update Graph
Dressing room
"S", "E", "C" keys to move. Right Mouse Button to Rotate the Camera. Shift to move quickly
MAPIR

Fig. 10. Use case of *ViMantic* where a plugin calculates the probability for the room to belong to a certain category (bathroom, kitchen, living room, etc.). This is done according to the objects that have been detected inside. In this case, the result obtained for the dressing room is inconclusive due to the lack of detected objects.

Manager employs about 3 ms to process it, while the time required by the *Ontology Manager* to insert new instances or update the already existing ones is negligible. It is also worth mentioning the 5 ms demanded to merge the detections belonging to the same object, which is triggered by Unity 3D within the *Object Manager*. Summing up all the time required by these components we retrieve an average computational time of ∼114 ms, which implies a working frequency of 8.7 Hz. RGB-D cameras usually work at 30 Hz, however, in many applications this frequency is decreased, since it provides a huge amount on information that can hardly be processed online. That is the case of object detection where, considering the robot speed, the processing of images at 30 Hz provides redundant detection results and unnecessarily overloads the limited robot computational resources. Indeed, RGB-D images in the Robot@Home dataset was gathered at a frequency ranging from 1 Hz up to 11 Hz, so *ViMantic* is able to reach a real time operation in it. Nevertheless, if lower execution times are needed, the network used inside the *Object Recognition* component (which is clearly the bottleneck) could be replaced by a faster one (*e.g.* YOLOv3, which works at 20–45 Hz [47]).

This analysis has been done using a single client sending information to the server. To estimate how the addition of more clients affects the computational time on the server side, the time required by the *Object Manager* must be multiplied by the number of clients, while the time needed for fusing detections remains the same. In this way, the addition of a client only demands an extra execution time of 3 ms.

Since both, clients and server, exchange information by means of a *WebSocket* connection, delays in such connection could result in delays while integrating information into the semantic map. However, since the server side is able to process such information at a high frequency (∼124 Hz), it could quickly recover from temporal delays.

6.4. Use cases

This section describes two use cases that pose different scenarios involving robots and semantic maps. In them, the utilization of semantic maps built by *ViMantic* enables such robots to efficiently perform high-level tasks.

6.4.1. Inferring room categories

As introduced in Section 4.6, the functionality of the proposed architecture can be extended in a straightforward way by the addition of plugins. As a first example of what such plugins could do, let us consider a plugin able to infer new knowledge from the one already existing in the map. Suppose a scenario where, in addition to the categories of the objects detected in the environment, it is also needed to know the categories of the rooms where they appear, namely bedroom, living room, bathroom, kitchen, corridor, etc (see next use case). For achieving that, human elicitation could be used to define concepts representing the room types, and to describe them by codifying the objects that can be typically found therein, *e.g.* microwaves in kitchens and toilets in bathrooms [62].

In a previous work [16], we designed a plugin able to exploit such prior knowledge, calculating the probability for each room to belong to the considered categories according to the objects detected therein and their confidence scores. Such a plugin works in a passive way, just processing the available information, hence there are situations where the reported probability is inconclusive due to the lack of detected objects. Fig. 10 shows an example where only one chair was detected inside a dressing room. In that case, the retrieved probability for belonging to categories where chairs typically appear like bedroom, dressing room or living room is the same. This plugin could be extended to exhibit a proactive behavior, *e.g.* by sending to an agent an order to inspect a certain room with inconclusive results. This would permit the detection of new objects or the enhancement of the knowledge about previously detected ones, helping to properly categorize the room.

6.4.2. Towards efficient operation: multiple robots, one choice

The second use case supposes a scenario with multiple collaborative robots/agents. Thus, if the user gives an order, *e.g.* "Please, check the sink in the bathroom near the bedrooms", a logical question arises: *which robot will be best suited for efficiently complete the task?*. In such scenario, it would be convenient to rely on a plugin able to optimally choose among the available robots according to their conditions. Towards such purpose, the information coded into semantic maps results of great utility.

The aforementioned example: "Please, check the sink in the bathroom near the bedrooms", could be interpreted as a navigation plus inspection tasks. A system for interpreting human commands would have to translate them into instructions that the robot can carry out. Next, another system, which could be implemented as a plugin to this architecture, would have to identify the most appropriate robot to perform the tasks. Fig. 11 illustrates this scenario, with two robots operating in the house. The semantic map helps here by providing the locations of both, the two robots, and two previously detected sinks. Notice that, at this point, it serves as a communication channel between the user and the architecture in terms that both are able to understand, *e.g.* check, tap, bathroom or bedroom. The semantic map could be also used to extract new information needed for completing the task. An example of this was illustrated in the previous use case, permitting the architecture to categorize, in this one, two rooms as bathrooms and three of them as bedrooms. In this way, to carry out the commanded tasks, the plugin could check the distance from each bathroom to the categorized bedrooms, obtaining `Bathroom-2` as the closest one and setting it as the navigation goal. Finally, by considering the paths that both robots have to follow to reach such a goal, the plugin could decide which one is closer to the sink (`Robot-2`) and send a navigation command to it. This selection criteria can become more sophisticated by also considering the score of the detected objects/rooms, the robots' workload, the level of their batteries, etc [44].

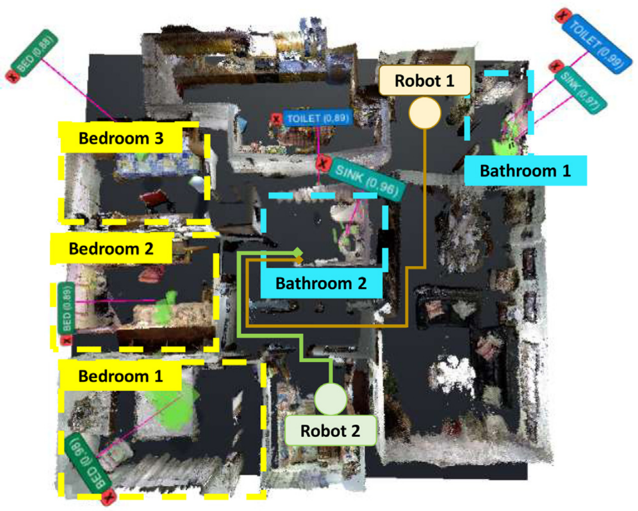


Fig. 11. Example of a use case where a plugin decides which is the best suited robot for accomplishing the command "Check the tap in the bathroom near the bedrooms". The discontinuous lines delimit the rooms, while colors represent their categories. The continuous lines show the path that each robot should follow to reach the target.

7. Conclusions

In this article we have presented *ViMantic*, a novel architecture for semantic mapping by mobile robots. It provides a number of features demanded by modern mobile robotic systems, including map model definition, automatic population, distributed execution, human–robot interface, and public availability. For that, the architecture relies on a client–server design. On the one hand, the server side, built upon Unity 3D, can operate on the own robot or an external device. It is in charge of building and managing the semantic map, also keeping a virtual representation of the environment that allows the user to interact in a friendly way. On the other hand, the client side, developed in the ROS framework, can be run on one or multiple robots/agents simultaneously. It includes components for gathering information from sensory data (detected objects, robot localization, etc.) as well as for acting in the physical environment (navigate, play sounds, fetch and carry objects, etc.). Both sides exchange information by standardized *WebSocket* communication channels. For the sake of modularity, the architecture permits developers to straightforwardly add new plugins/components depending on their needs.

We have reported a number of experiments and use cases supporting the suitability of *ViMantic* for the building and exploitation of semantic maps. For that, we have relied on the Robot@Home dataset, which provides RGB-D images and 2D laser scans from different houses collected by a mobile robot. Concretely, we have performed two experiments: the first one considers a robot operating in such houses, while the second experiment poses a collaborative scenario with two. In both cases, we have described the most relevant parts of the semantic maps building process, the generated information, and the performance of critical components. We conclude that, based on the results obtained in this experiments, *ViMantic* is a good architecture for semantic mapping that meets the expectations of our proposal. To exemplify possible uses, we have also commented on two possible use cases exploiting the outcome of the proposed architecture for categorizing rooms, and for optimally selecting the most suitable robot for performing a task.

In the future we plan to further leverage and develop the features of *ViMantic*. For example, the detection of objects could take advantage of contextual relations to give a sense of coherence to its results, for example, that microwaves are typically found on top of tables and counters but not on the floor. We also aim to incorporate the capability to update the map in order to consider dynamic objects, i.e. those objects that can be moved from the place from which they were previously observed (chairs, bottles, etc.). Another possible extension could be the utilization of the virtual environment for virtual reality purposes, *e.g.* to provide an immersive experience in semantic maps to the user.

CRediT authorship contribution statement

D. Fernandez-Chaves: Methodology, Software, Data curation, Investigation, Writing – original draft. **J.R. Ruiz-Sarmiento:** Methodology, Validation, Supervision, Writing – review & editing. **N. Petkov:** Funding acquisition, Project administration, Supervision, Writing – review & editing. **J. Gonzalez-Jimenez:** Conceptualization, Project administration, Supervision, Writing – review & editing.

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.

Acknowledgments

This work has been supported by the research projects *WISER* (DPI2017-84827-R), funded by the Spanish Government and financed by the European Regional Development's funds (FEDER), *MoveCare* (ICT-26-2016b-GA-732158), funded by the European H2020 program, and by a postdoc contract from the I-PPIT program of the University of Málaga, and the UG PHD scholarship program from the University of Groningen. Funding for open access charge: Universidad de Málaga/CBUA.

References

[1] R. Gehle, K. Pitsch, T. Dankert, S. Wrede, Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, in: HRI '17, 2017, pp. 187—195, http://dx.doi.org/10.1145/2909824.3020219.
[2] H. Khambhaita, R. Alami, Robotics Research, Springer International Publishing, Cham, 2020, pp. 285–300.
[3] R. Chatila, J. Laumond, Proceedings. 1985 IEEE International Conference on Robotics and Automation, 2, 1985, pp. 138–145, http://dx.doi.org/10.1109/ROBOT.1985.1087373.
[4] B. Kuipers, Cogn. Sci. 2 (2) (1978) 129–153, http://dx.doi.org/10.1207/s15516709cog0202_3.
[5] A. Nüchter, J. Hertzberg, Robot. Auton. Syst. 56 (11) (2008) 915–926.
[6] C. Galindo, J. Fernandez-Madrigal, J. Gonzalez, A. Saffiotti, Robot. Auton. Syst. 56 (11) (2008) 955–966.
[7] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, J. Garcia-Rodriguez, A review on deep learning techniques applied to semantic segmentation, 2017, arXiv preprint arXiv:1704.06857.
[8] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A.L. Yuille, Semantic image segmentation with deep convolutional nets and fully connected CRFs, in: International Conference on Learning Representations (ICLR), 2015.
[9] J.R. Ruiz-Sarmiento, C. Galindo, J. González-Jiménez, Knowl.-Based Syst. 119 (2017) 257–272.
[10] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J. Fernandez-Madrigal, J. Gonzalez, 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005, pp. 2278–2283, http://dx.doi.org/10.1109/IROS.2005.1545511.
[11] A. Pronobis, P. Jensfelt, Large-scale semantic mapping and reasoning with heterogeneous modalities, in: Robotics and Automation (ICRA), 2012 IEEE International Conference on, 2012, pp. 3515–3522.
[12] M. Günther, T. Wiemann, S. Albrecht, J. Hertzberg, Building semantic object maps from sparse and noisy 3D data, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013), 2013, pp. 2228–2233.

[13] D. Fernandez-Chaves, J.R. Ruiz-Sarmiento, N. Petkov, J. Gonzalez-Jimenez, in: I. Rojas, G. Joya, A. Catala (Eds.), Advances in Computational Intelligence, Springer International Publishing, Cham, 2019, pp. 313–324.
[14] E. Prestes, J.L. Carbonera, S.R. Fiorini, V.A.M. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Goncalves, M.E. Barreto, M. Habib, A. Chibani, S. Gérard, Y. Amirat, C. Schlenoff, Robot. Auton. Syst. 61 (11) (2013) 1193–1204.
[15] L. Riazuelo, M. Tenorth, D.D. Marco, M. Salas, D. Gálvez-López, L. Mösenlechner, L. Kunze, M. Beetz, J. Tardós, L. Montano, J. Montiel, IEEE Trans. Autom. Sci. Eng. 12 (2) (2015) 432–443.
[16] D. Fernandez-Chaves, J.R. Ruiz-Sarmiento, N. Petkov, J. Gonzalez-Jimenez, International Conference on Applications of Intelligent Systems (APPIS), 2020, http://dx.doi.org/10.1145/3378184.3378230.
[17] J.J. Roldán, E. Peña Tapia, D. Garzón-Ramos, J. de León, M. Garzón, J. del Cerro, A. Barrientos, Studies in Computational Intelligence, vol. 778, Springer Verlag, 2019, pp. 29–64, http://dx.doi.org/10.1007/978-3-319-91590-6_2.
[18] F. Navarro, J. Fdez, M. Garzón, J.J. Roldán, A. Barrientos, Advances in Intelligent Systems and Computing, 694, Springer Verlag, 2018, pp. 606–616, http://dx.doi.org/10.1007/978-3-319-70836-2_50.
[19] R. Capobianco, J. Serafin, J. Dichtl, G. Grisetti, L. Iocchi, D. Nardi, A proposal for semantic map representation and evaluation, in: Mobile Robots (ECMR), 2015 European Conference on, 2015, pp. 1–6.
[20] S. Kaszuba, S.R. Sabbella, V. Suriani, F. Riccio, D. Nardi, Rosmeery: Robotic simulated environment for evaluation and benchmarking of semantic mapping algorithms, 2021, arXiv preprint arXiv:2105.07938.
[21] E. Bastianelli, D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, D. Nardi, On-line semantic mapping, in: Advanced Robotics (ICAR), 2013 16th International Conference on, 2013, pp. 1–6.
[22] I. Kostavelis, A. Gasteratos, Robot. Auton. Syst. 66 (2015) 86–103.
[23] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, IEEE Internet Things J. 3 (5) (2016) 637–646.
[24] J.-R. Ruiz-Sarmiento, C. Galindo, J. Monroy, F.-A. Moreno, J. Gonzalez-Jimenez, Knowl.-Based Syst. 168 (2019) 100–108.
[25] N. Eric Maillot, M. Thonnat, Image Vision Comput. 26 (1) (2008) 102–113.
[26] N. Durand, S. Derivaux, G. Forestier, C. Wemmert, P. Gancarski, O. Boussaid, A. Puissant, Ontology-based object recognition for remote sensing image interpretation, in: Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on, vol. 1, 2007, pp. 472–479.
[27] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, D. Lange, Unity: A general platform for intelligent agents, 2018, arXiv preprint arXiv:1809.02627.
[28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, ICRA Workshop on Open Source Software, 3, (3.2) Kobe, Japan, 2009, p. 5.
[29] J. Han, D. Zhang, G. Cheng, N. Liu, D. Xu, IEEE Signal Process. Mag. 35 (1) (2018) 84–100.
[30] J.R. Ruiz-Sarmiento, C. Galindo, J. González-Jiménez, Int. J. Robot. Res. 36 (2) (2017) 131–141.
[31] C. Galindo, A. Saffiotti, Robot. Auton. Syst. 61 (10) (2013) 1131–1143.
[32] H. Zender, O.M. nez Mozos, P. Jensfelt, G.-J. Kruijff, W. Burgard, Robot. Auton. Syst. 56 (6) (2008) 493–502.
[33] M. Tenorth, L. Kunze, D. Jain, M. Beetz, 2010 10th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2010, 2010, pp. 430–435, http://dx.doi.org/10.1109/ICHR.2010.5686350.
[34] D. Pangercic, B. Pitzer, M. Tenorth, M. Beetz, Semantic Object Maps for robotic housework - representation, acquisition and use, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp. 4644–4651.
[35] J.O. Reinaldo, R.S. Maia, A.A. Souza, Proceedings - 2015 Brazilian Conference on Intelligent Systems, BRACIS 2015, Institute of Electrical and Electronics Engineers Inc., 2016, pp. 210–215, http://dx.doi.org/10.1109/BRACIS.2015.50.
[36] X. Qi, W. Wang, M. Yuan, Y. Wang, M. Li, L. Xue, Y. Sun, Int. J. Adv. Robot. Syst. 17 (1) (2020) http://dx.doi.org/10.1177/1729881419900066.
[37] A. Cosgun, H.I. Christensen, Paladyn 9 (1) (2018) 254–276, http://dx.doi.org/10.1515/pjbr-2018-0020, arXiv:1710.08682.
[38] G. Gemignani, D. Nardi, D.D. Bloisi, R. Capobianco, L. Iocchi, in: A.M. Hsieh, O. Khatib, V. Kumar (Eds.), Experimental Robotics: The 14th International Symposium on Experimental Robotics, in: Springer Tracts in Advanced Robotics, vol. 109, Springer International Publishing, 2016, pp. 339–355.
[39] M. Günther, J.R. Ruiz-Sarmiento, C. Galindo, J. Gonzalez-Jimenez, J. Hertzberg, Robot. Auton. Syst. (2018).
[40] V. Wang, F. Salim, P. Moskovits, The Definitive Guide To HTML5 WebSocket, vol. 1, Springer, 2013.
[41] M. Uschold, M. Gruninger, Knowl. Eng. Rev. 11 (1996) 93–136.
[42] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz, Web Semant.: Sci. Serv. Agents World Wide Web 5 (2) (2007) 51–53.
[43] D. Tsarkov, I. Horrocks, Automated Reasoning: Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 292–297.
[44] J. Monroy, J.R. Ruiz-Sarmiento, F.-A. Moreno, F. Melendez-Fernandez, C. Galindo, J. Gonzalez-Jimenez, Sensors 18 (12) (2018).
[45] M. Everingham, S.M. Eslami, L. Gool, C.K. Williams, J. Winn, A. Zisserman, Int. J. Comput. Vision 111 (1) (2015) 98–136.
[46] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C.L. Zitnick, Microsoft COCO: Common Objects in Context, in: European Conference on Computer Vision, 2014, pp. 740–755.
[47] J. Redmon, A. Farhadi, Yolov3: An incremental improvement, 2018, arXiv preprint arXiv:1804.02767.
[48] S. Ren, K. He, R. Girshick, J. Sun, IEEE Trans. Pattern Anal. Mach. Intell. 39 (6) (2017) 1137–1149.
[49] K. He, G. Gkioxari, P. Dollár, R.B. Girshick, Mask R-CNN, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980–2988.
[50] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, R. Girshick, Detectron2, 2019, https://github.com/facebookresearch/detectron2.
[51] G. Gkioxari, J. Malik, J. Johnson, Mesh r-cnn, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 9785–9795.
[52] D. Zuñiga Noël, J.R. Ruiz-Sarmiento, J. Gonzalez-Jimenez, Computer Analysis of Images and Patterns. Lecture Notes in Computer Science, 11678, Springer International Publishing, 2019, pp. 659–671, http://dx.doi.org/10.1007/978-3-030-29888-3_54.
[53] D. Zuñiga Noël, J.R. Ruiz-Sarmiento, R. Gomez-Ojeda, J. Gonzalez-Jimenez, IEEE Robot. Autom. Lett. 4 (3) (2019) 2862–2869, http://dx.doi.org/10.1109/LRA.2019.2922618.
[54] G. Grisetti, C. Stachniss, W. Burgard, IEEE Trans. Robot. 23 (1) (2007) 34–46.
[55] D. Fox, KLD-sampling: Adaptive particle filters, in: Advances in Neural Information Processing Systems, 2002, pp. 713–720.
[56] J. Alonso-Mora, S. Baker, D. Rus, Int. J. Robot. Res. 36 (9) (2017) 1000–1021, http://dx.doi.org/10.1177/0278364917719333.
[57] J. González-Jiménez, C. Galindo, J. Ruiz-Sarmiento, 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, IEEE, 2012, pp. 827–832.
[58] J. Redmon, A. Farhadi, YOLOV3: An incremental improvement, 2018, arXiv:1804.02767.
[59] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A.C. Berg, Lecture Notes in Comput. Sci. (2016) 21–37.
[60] S. Ren, K. He, R. Girshick, J. Sun, Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, in: NIPS'15, MIT Press, Cambridge, MA, USA, 2015, pp. 91–99.
[61] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, Q. Tian, Centernet: Keypoint triplets for object detection, 2019, arXiv:1904.08189.
[62] J.R. Ruiz-Sarmiento, C. Galindo, J. González-Jiménez, Joint categorization of objects and rooms for mobile robots, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015.