UNIVERSITÀ DEGLI STUDI
DI GENOVA

Semester Project

Software Architecture for Robotics

Human-like Memory for Robots

Tutor: Luca Buoncompagni
Supervisor: Prof. Fulvio Mastrogiovanni

Submitted by:

Bernabei	Federico	Napoli	Giovanni
Cartosio	Luca	Robers	Maren
De Mari	Lorenzo	Salunkhe	Durgesh
Fallica	Francesco	Ventura	Alessandra
Liu	Zijian		

Academic Year 2018/2019

Contents

Abbreviations

SIT	Scene Identification and Tagging algorithm
PITT	Primitive Identification Tracking Tagging
ROS	Robot Operating System
STM	Short-term memory
LTM	Long-term memory
ER	Encoding Reasoner
RR	Retrieving Reasoner
FR	Forgetting Reasoner
ST	Storing Reasoner
CR	Consolidating Reasoner
DM	Dialogue Manager

1 Objective of the Project

This project aims to implement software that enables a robot to have a human-like memory: it is supposed to learn, classify, recall, and remarkably also forget information. To achieve this behaviour we combine and enhance several software components. This includes speech and visual recognition (Sections 2.2.2, 2.2.3), a Scene Identification and Tagging algorithm (Section 2.2.4.1), and an ontology reference (Section 2.2.1).

1.1 Why should a Robot Forget?

Not all information that a robot collects and stores is actually important and relevant. Information might be old and outdated, information can have a value just for a limited amount of time, information might me irrelevant because it is not used more than once. The challenge artificial intelligence faces is to distinguish between relevant and unnecessary knowledge while avoiding known problems like overfitting (storing information in a way to detailed manner [1]) and the so-called catastrophic forgetting (abruptly forget/overwrite data during the learning of new information [2]).

1.2 How does the Human Memory Work?

Basis for the modeling of cognitive memory processes is the Multi-Store Model developed by Richard Atkinson and Richard Shiffrin in 1968 (Figure 1) [3]. The model devides the human memory into three components: a sensory register, a short term store and a long term store. The information stored in the memories is perceived by various sensors (e.g. visual, acoustic, olfactory) and then passed to the short term memory (STM). This type of memory is fist, but has a limited capacity. STM can store 7 ± 2 memory items (=chunks of information) [4]. Though much of the information in the STM is forgotten rather quickly due to a time decay, via rehearsal (continous repitition) it can be held in the STM for a longer period of time. Some of the information of the STM is transfered to the long term memory (LTM) through different transfer mechanisms. Once in the LTM the information is stored more or less permanently. To concousily remember an information, it needs to be recalled and temporarily stored in the STM again. This process is called retrieving. This general model is well suited for the technical adaption and implementation in computer sience and robotics.

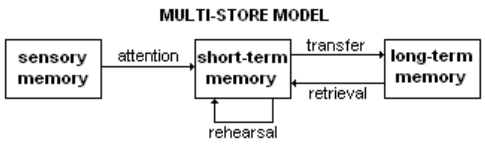


Figure 1: **Multi-Store Model of Memory.** Also Atkinson-Shiffrin or modal model.[5, 3].

The STM and LTM can be further distinguished into subclasses. This project does not adapt all existing memory types, but a subset of existing memory types (Figure 2). The sensory memory (part of STM) perceives the information from the outer world. For this project it is represented by a visual module that provides the visual information and a dialogue module that is equipped with a speech recognition device to enable the user to access the memory. The memory items that enter the LTM are exclusively declarative. Non declarative knowledge like procedural knowledge and non concsious knowledge will not be represented. To keep the memory model as close to the human memory structure, the declarative memory is further devided into semantic and episodic memory.

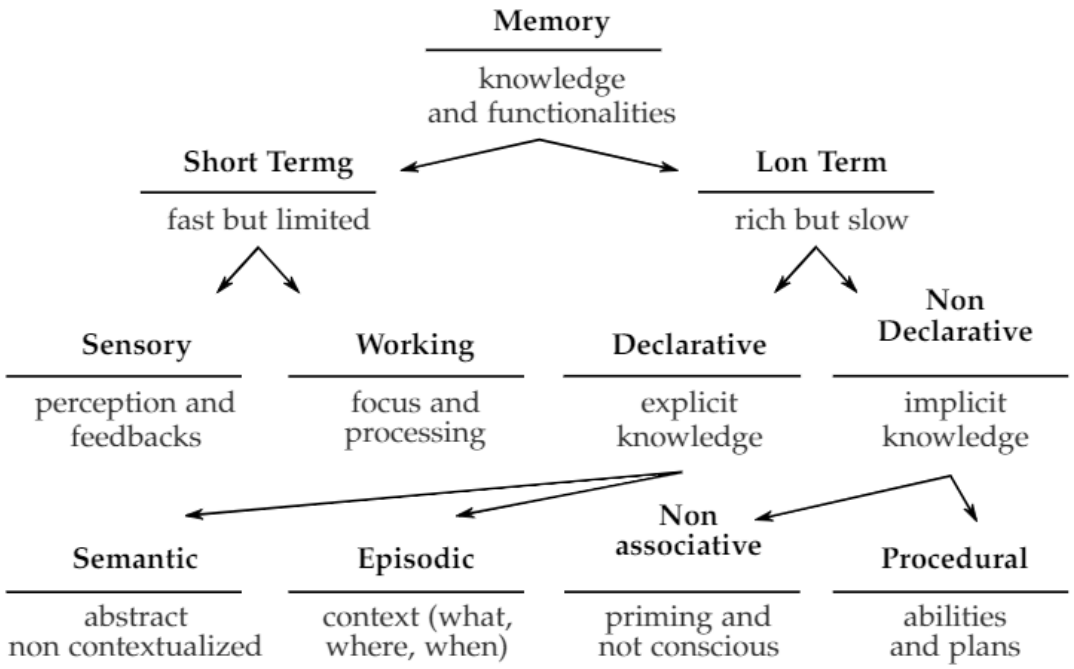


Figure 2: **A simplified taxonomy of memory types.** [6, 7, 8].

2 The System's Architecture

Robotics Engineering is a field of research which is permanently evolving and intersects in multiple areas with various other fields like natural sciences, mechanical engineering, and computer sciences. Many obstacles that need to be tackled by robotics engineers are typically not unique to the field of robotics, but need to be overcome also by other disciplines. A method to address these similarities in tasks, that gains more and more popularity is the modular design. Instead of finding and designing highly specialized solutions for a single project, it is partitioned into smaller workloads. In the following process, a subsystem is designed for each of those packages. In the final step, all subsystems (modules or skids) are combined, and together they form the solution to the complete project. This approach leads to a more efficient use of resources, a shorter learning time, higher flexibility and compatibility [9]. The modular design approach is used by multiple disciplines. In software design this is called modular programming (interface-based programming in object-oriented programming) and it is a main tool for component-based robotic engineering [10, 11].

The model of human-like memory for robots has been designed with the objective to develop reusable modules that can be re-purposed and rearranged for future applications. Hence, the whole system is divided into software modules that can stand for their own and interact with each other through well-defined interfaces. By using this approach we guarantee that if only one specific part of this architecture is useful to a future project it can be implemented with a limited amount of customization, e.g. the modification of interfaces or expansion of some functionalities.

2.1 Overall Architecture

The architecture of the human-like memory for robotic implementation embraces the principles of modularity that permits to reuse, recombine and repurpose single components of a system (Figure 3). An ontology represents the knowledge collected by the perceiving module, in our case a vision module is used (Section 2.2.2). To encode, store and maintain the architecture the ARMOR service (a multi-ontology reference that is compatible with the ROS system) is used (Section 2.2.1). The reasoning module is the core of the robot memory architecture, as it is in charge of evaluating, assessing, and processing the collected information. The user can access the knowledge by interacting with the system via a hybrid perceiving-acting module, here a speech recognition device [6].

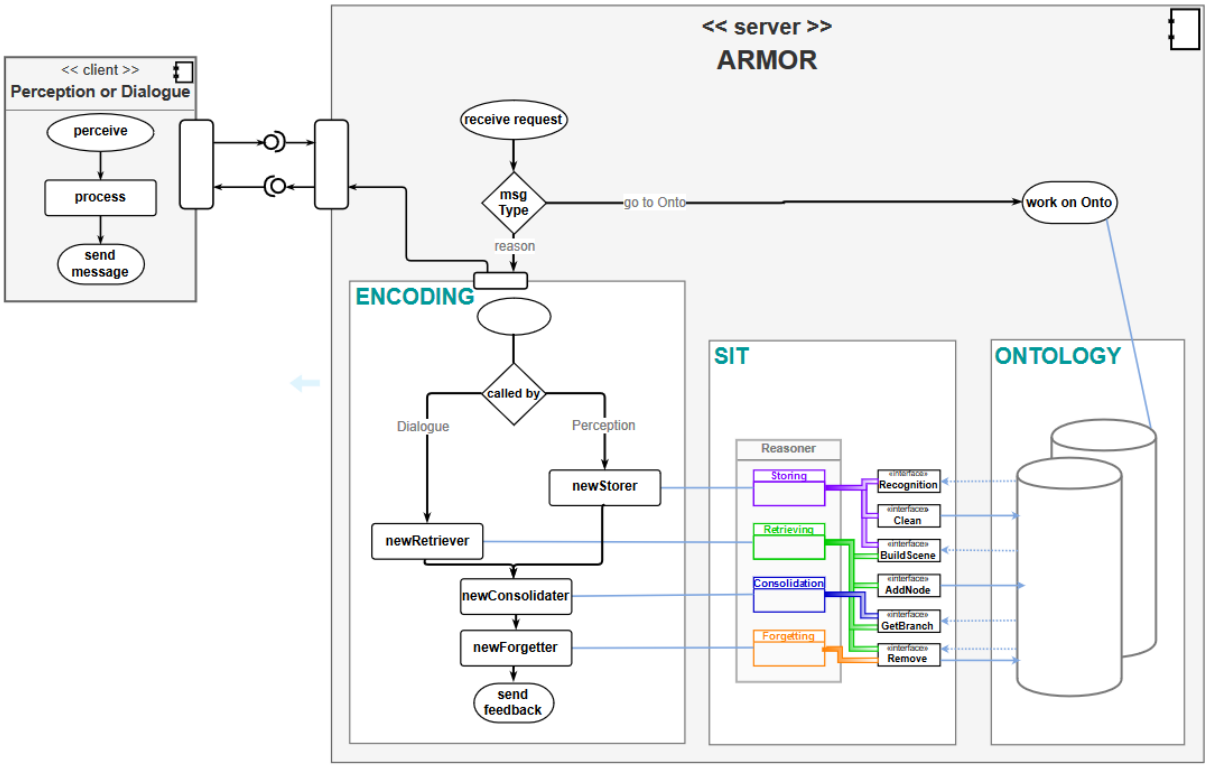


Figure 3: **Architecture of the Human-like Memory for Robots.** The black arrows show the process flow while light blue lines and arrows have a symbolical meaning to visualize the action. For the sake of comprehensibility and simplicity the AMOR and OWLOOP libraries used by SIT and ARMOR are not displayed.

2.2 Description of the Modules

The modules can be categorized into clients and server. Inside the server (ARMOR) different kinds of submodules can be found. The SIT algorithm, that includes the reasoning modules, can not directly write or read from the ontology. It is necessary to use the AMOR library via the OWLOOP and ROS interfaces to manipulate the ontology.
Client and server communicate via a set of messages. The first message is a initialization message while the last signals the end of the row of messages. The messages between initialization and end supply the server with all necessary information (Figure 4).
If the client is a retrieving module, the first interface inside the ARMOR (decision diamond 'msg Type' Figure 3) directs it to the encoding block. If the message of a visual perception device contains the command 'add' the interface permits it to directly manipulate the ontology (via AMOR). Otherwise it will be directed to the encoding block like the retrieving message.

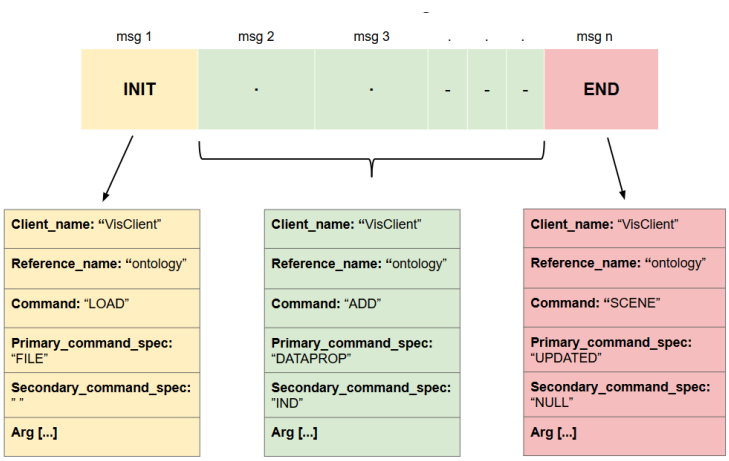


Figure 4: **Structure of the messages between client and server.** Example messages client being a visual perception module.

2.2.1 ARMOR - A ROS Multi-Ontology Reference

ARMOR is a powerful and versatile management system for single and multi-ontology architectures under ROS. It allows to load, query and modify multiple ontologies and requires very little knowledge of OWL APIs and Java. Despite its ease of use, ARMOR provides a large share of OWL APIs functions and capabilities in a simple server-client architecture and offers increased flexibility compared to previous ontological systems running under ROS.
In order to ensure that only one client at a time is altering the ontology it is important to use a service instead of a publish/subscribe architecture. The ARMOR service is used by all modules of the architecture, namely perception, dialogue and reasoners. Both perception and dialogue modules are external modules that act as clients of the ARMOR service, bat are no part of the internal reasoning. In the current configuration we are using the Microsoft KINECT as in- and output device, but it is possible to substitute it by any other perception component or system.

2.2.2 Visual Perception

Zijian Liu, Durgesh Salunkhe, Alessandra Ventura

2.2.2.1 Theoretical background

In the architecture mentioned in Figure 3, the perception module plays a role of interfacing the world environment with the reasoning architecture. The module uses a visual sensor, Microsoft Kinect in our case, to identify the scenes and interprets the data in terms of facts and knowledge relevant to the reasoning entity. In order to fulfill its function, the vision module uses PITT package to receive the data from Kinect in the form of objects and its specific attributes. The PITT package uses RANSAC segmentation to identify the point clouds and fit a specific object shape already known to the algorithm. This enables to get a better values for attributes as the point clouds may change but due to the shape-fitting method, the attributes such as length, radius and centroid remain close to a constant value. This output albeit relevant has noises and few shortcomings. The raw data from Kinect is susceptible to noises and thus results in ambiguous and sometimes wrong segmentation. The RANSAC segmentation relies on point cloud data and due to the loss of points or noise, it may happen that the same object is identified as a different shape or as a new object altogether.

The work presents a structure that acts as an interface between the present PITT output and the ARMOR service. The work details upon the different functions used to store and manipulate the PITT output in order to give a result based on statistics of the observed instances. It further emphasizes on the service implemented to communicate the improved data to the ARMOR service by keeping synchronization in focus.

2.2.2.2 Optimising attributes

The Figure 5 illustrates the architecture implemented in the vision subsystem. The red block in the figure is the pre-existing ROS architecture. In order to understand the architecture, the following important names of nodes, topics, messages and services are mentioned in the table:

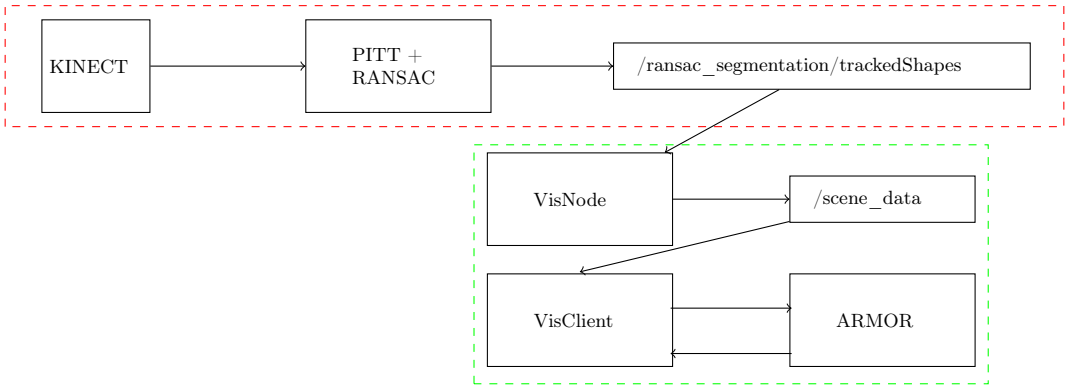


Figure 5: **Schematic diagram for the vision subsystem.**

Table 1: **Overview of the Subsystem.**

Name	Type	Details
ransac_segmentation	node	for interpreting Kinect data
/ransac_segmentation/trackedShapes	topic	of ransac_segmentation
TrackedShapes	message	array of TrackedShape
TrackedShape	message	user-defined message
All_shapes_attributes	message	array of Attribute
Attribute	message	user-defined message
VisNode	node	for manipulating RANSAC data
/scene_data	topic	of VisNode
VisClient	node	acts as a client to ARMOR
ArmorDirectiveReq	message	user-defined message
ArmorDirectiveList	service	user-defined .srv

Newly added nodes and topics In this subsection, we explain in details the working of the nodes we created. There are two nodes added in the package, one that subscribes to the topic /ransac_segmentation/trackedShapes and the other that acts as a client to ARMOR.

1. VisNode:
VisNode is a ROS node that subscribes to the topic /ransac_segmentation/trackedShapes(referred as RANSAC topic) and publishes on the topic /scene_data. The functionality of this node is to manipulate the data published on the RANSAC topic in order to deal with the uncertainity in the detecting the objects as well as to present a statistical analysis of the observed scenes. It is important to note that the VisNode continously hears to the output of ransac_segmentation and

publishes on /scene_data the cumulative analysis of the data. This enables us to create another node that each time receives optimised at that moment.

Averaged attributes and best-chosen shape The assumption in the working principle of this module is that the objects are not replaced faster than the refresh rate of ransac_segmentation. This assumption leads to a conclusion that if the point cloud has not moved beyond a particular threshold then the object_id assigned to the point cloud does not change even if the shape detected is different from the previous one. As mentioned in above sections, the output of ransac_segmentation provides erratic data due to the reliance on the point cloud data. This leads to certain issues such as:

1. The fluctuation in the centroid of the shape
2. Detecting the same object as different shapes
3. The loss of attributes due to the change in shape

Due to the above issues, the attributes changes abruptly. for example: If it was detecting the shape as cylinder with height h, it records the centroid at h/2. In the next scenario, it detects the same object as cone of height h and the centroid suddenly shifts to h/3. In another case, if it detects a cylinder in one instance and sphere in other, then it completely loses the information about the height and the error in the orientation of the cylinder.

The presented work addresses this issue by keeping record of all the shapes detected for the same of object id. The following algorithm details the working principle of the same.

```
function ATTRIBUTES_AVERAGE(trackedShapes)
    subscribe -> /ransac_segmenation/trackedShapes
    for Each object_ID in trackedShapes do
        if detected.shape = cylinder then
            avg_centroid = msg->trackedShapes.centroid + old_data
                                          ▷ Read the centroid and update the new average
            avg_centroid = new_data + old_data
                                            ▷ Read the height and update the new average
            avg_orientation = new_data + old_data
                                        ▷ Read the orientation and update the new average
            Color = msg->trackedShapes.color                    ▷ Update the latest color
            cylinder_count++                                ▷ Increase the count for cylinder
        end if
    end for
    Similarly for other shapes
    MaxShape = Max(cylinder_count,sphere_count, cone_count, plane_count)
return MaxShape
return Average_attributes
end function
```

Figure 6: **Algorithm for updating the attributes.**

In this algorithm, the data is processed at a frequency equal to that of the RANSAC topic. The updated attributes and the best chosen shape is sent as a All_shapes_Attributes type of message. The advantage of introducing this node is that we save the attributes of each instance of the same id and thus there is no loss in the attributes. As the data is stored according to the detected shape, the average of attributes such as height and centroid are stored discretely and thus are not susceptible to high fluctuations due to change in shape detection.

2. VisClient:
VisClient is a ros node that subscribes to the topic, /scene_data and acts as a client to the ARMOR service. The request type in this service is an array ArmDirectiveList. In this node the request is sent inside the callback function of the subscriber. As the request of the client is blocking by nature, the synchronization between ARMOR and Viion subsystems output is ensured. After a response is received by VisClient, it spins back into the callback function and gets the most recent data from the VisNode. The VisClient node also plays an important role of interfacing the messages sent by VisNode that contains object_ids and their attributes with the request compatible with ARMOR. In the existing architecture, the Encoding reasoner plays the role of server and reads the request sent by VisClient. Each request is initialised by a LOAD_-INIT_ command and terminates with SCENE_UPDATED_ COMMAND. This facilitates the Encoding reasoner subsystem to identify the complete information of the scene.

2.2.3 Dialogue

The subgroup working on the dialogue module did not contribute to this report.

2.2.4 Reasoner

Federico Bernabei, Luca Cartosio, Lorenzo de Mari,
Francesco Fallica, Giovanni Napoli, Maren Robers

The reasoning module is the heart of this project. It is responsible for structuring, sorting, storing, and retrieving memory. While the each reasoner structurally is thought of as a service, they are technically implemented as a java class. This is due to the fact that we are using the Scene Identification and Tagging (SIT) algorithm which is written in Java.
The reasoner is divided into two parts, the external and the internal reasoner. The reasoners are grouped based on their interfaces. The external reasoners (encoding reasoner (ER) and retrieving reasoner (RR)) share an interface with the outer world (perception/dialogue modules) the internal reasoners (storing reasoner (SR), consolidationg reasoner (CR), and forgetting Reasoner (FR) only interface with other reasoning modules. By reducing the interfaces to the outer world as far as possible, we enhance the reusability and the customization effort.

Table 2: **Requirements to run the Reasoning Functions.**

Function	Initial State Ontology	Required Interface	Provided Interface	Final State Ontology
Encoding	empty	message from client	message to client	filled with geometric individual
Retrieving	incl. geometric individuals	trigger + ontology instance	trigger + ontology instance	untouched, except for r_j
Storing	incl. geometric individuals	trigger + ontology instance	trigger + ontology instance	node class added to graph or updated s_j
Consolidating	node class added to graph	trigger + ontology instance	trigger + ontology instance	untouched, except for ξ
Forgetting	node class added to graph	trigger + ontology instance	trigger + ontology instance	in case of weak level: deleted node

2.2.4.1 SIT Algorithm

The SIT algorithm was developed to learn and recognize compositions ob objects based on their qualitative spatial relations. It is based on an OWL ontology, the Pellet reasoner and a collection of mapping functions. Its main purposes is to describe primitive objects though geometric coefficients that define their shape. Than, qualitative spatial relations, such as: right/left, front/behind, above/below, parallel, perpendicular and coaxial are computed symbolically. Those are mapped in an concrete scene representation (i.e.: an individual). The recognition phase is based on instance checking by looking for the abstract scene representation (i.e.: classes) that classify the scene individual. If those does not exist the algorithm is able to use the concrete scene as a template to learn its abstract class to be used for further classification. Noteworthy, the system is automatically able to reason about similarity between learned scene. Also, it can be the case that a very complex scene is recognized by a relative small number of relation that hold (i.e.: a sub scene). To discriminate when those differences are too high, and trigger a new learning procedure, the concept of confidence is introduce as a number within $[0, 1]$ [12].
During this project, we extended the existing SIT algorithm with the necessary reasoner classes (Section 2.2.4) as well as the needed functions to interact with the ontology (Figure 7).

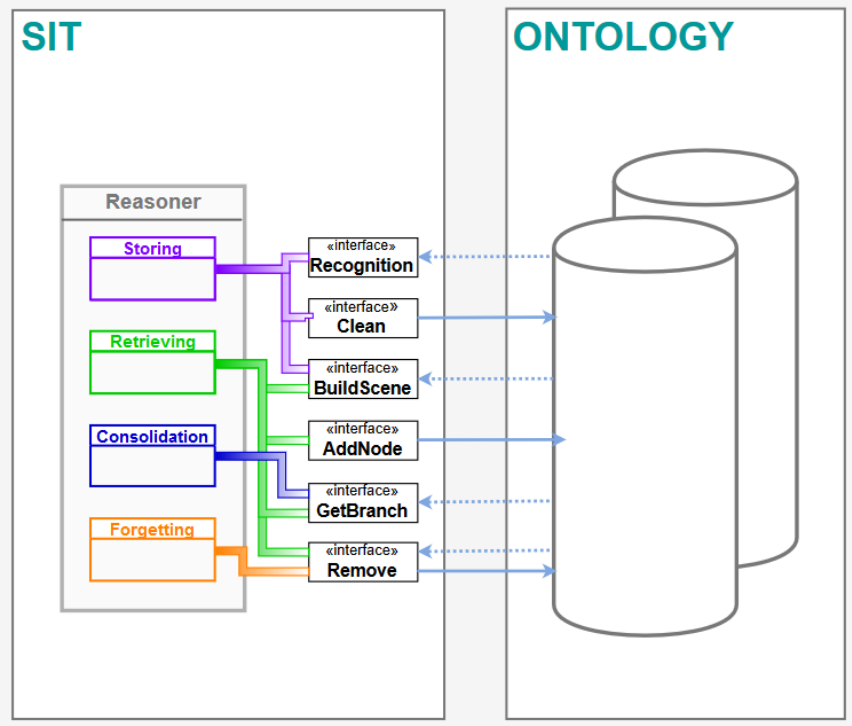


Figure 7: **Edited SIT algorithm.** Colored rectangles: reasoner classes, black rectangles: functional interfaces, colored channels: point to interfaces used by the corresponding reasoner, blue arrows: symbolic representation of reading (dotted) and writing (continuous) on the ontology.

2.2.4.2 Theoretical Background

Encoding The encoding function ($\mathcal{E}$) is equivalent to the perceiving function $\mathcal{P}$ of the SIT algorithm and computes the beliefs of a situation based on the facts. It connects the instances of a situation with a characteristic and has the purpose of generating the ontology facts according to the interfaces [6].

$$\begin{aligned} \mathcal{E}(f_i) &= R(\mathbf{R}_z, \gamma_x)(\epsilon, \gamma_y) \\ &= \mathbf{R}_{zx}(\epsilon, \gamma_y)_r \end{aligned} \tag{1}$$

Storing The storing function ($\mathcal{T}$) is a combination of the three SIT functions classifying, learning and structuring ($\mathcal{C}, \mathcal{L}, \mathcal{S}$). The learning function creates new categories in the ontology by mapping each belief b_r to a feature x_q. The learned category X_e is the definition of a new set that restricts a part of possible beliefs in the experience space, but is not yet a node in the ontology. The structuring function now includes the now node by finding the right position in the graph for the new node. The graph is structured s.t. the parent node's complexity is smaller than the children's. After the node placement the classifying function finds the subgraph X^* (output graph). In total, the storing function maps the beliefs about a situation to the features of an item X_j [6].
Since the episodic score might differ from the semantic score due to the time decay of the episodic memory, the forgetting function might delete a memory item only in one memory type. This might lead to information being stored not in both memories but only in one of them. At the current status of implementation the items of the semantic memory are treated separately from the semantic ones. However, it is generally possible to connect the memory types for instance by using meaningful names. If this functionality is implemented, it is necessary to implement a synchronization between the memory types [6].

$$\begin{aligned} \mathcal{L} &: \mathbb{E} \to \mathbb{X} \\ e &\mapsto X_e \doteq x_q, q \in [1...m] \end{aligned} \tag{2}$$

$$\begin{aligned} \mathcal{S} &: \mathbb{X}, \mathbb{X} \to \mathbb{X} \\ (X^{t-1}, X_e) &\mapsto X \end{aligned} \tag{3}$$

$$\begin{aligned} \mathcal{C} : \mathbb{X}, \mathbb{E} &\to \mathbb{X} \\ X, e &\mapsto X^* \end{aligned} \tag{4}$$

$$\begin{aligned} \mathcal{T} : \mathbb{X}, \mathbb{E} &\to \mathbb{X} \\ X, e &\mapsto X^* \end{aligned} \tag{5}$$

Consolidating Human memory consolidation is a cognitive process that stabilizes the memory trace of an item after it's initial acquisition (learning) [13]. Typically it is divided into two types: synaptic that occurs within a few hours after learning and systems consolidation that is considered a reorganization process of memory that takes place over a period of weeks or years [14]. This type of reconsolidating of memories is not necessary to model in a technical implementation as a memory once physically stored by default is not deleted without the explicit intend to do so. However, some studies suggest that there is another kind of consolidation, the so called reconsolidation. It describes the process to maintain and strengthen the stored knowledge in the LTM. The consolidating function models a specific kind of consolidation, the retrieval consolidation. By retrieving, the memory is retraced and thereby the memory is strengthened [15]. An event that is recalled frequently obviously has a certain importance and thereby should be kept strong and clear.
Analogue to this reasoning, besides a frequently recalled event a frequently experienced event suggests a higher relevance in comparison to an event that was experienced only a single time. As a new node to the ontology is only added, if there is no existing nodes with identical properties, the consolidating function implemented for the robot memory keeps track of reexperienced situations. In addition the reconsolidation of recalled situations is realized withing the consolidating reasoner.

As an extension of the SIT algorithm, the consolidating function $\mathcal{N}$ is defined. It is responsible of computing the score of every $j-$th item u_j (Equation 6). The consolidating function is called every time the memory X is accessed by either the storing or the retrieving function as they update the storing and retrieving score, respectively. Since the nodes are ordered according to their complexity, retrieving a node X_i also affects the children and hence not only the node that exactly matches the retrieved scene needs to be updated, but the whole branch. To make the scores comparable between each other they are normalized [6].

$$\begin{aligned} \mathcal{N} : \mathbb{X}, \mathbb{X} &\to [0, 1] \\ X, X_j &\mapsto u_j \end{aligned} \tag{6}$$

The calculation of the score takes into account a variety of factors, namely the node's complexity (${}^{j}d$), its storing and retrieving scores (s_j, r_j), and the number of children (n_c) as well as their scores (ζ_i). Each element is multiplied with a weight (a) and then summed up to receive the final score (ζ_j)(Equation 7). As the episodic memory is supposed to decay over time, a logarithmic time dependent term is added to take this into account.

Semantic score:

$$\bar{\zeta_j} = a_d^j d + a_c \frac{\Sigma_i \zeta_i}{n_c} + a_s s_j + a_r r_j \tag{7}$$

Episodic score:

$$\bar{\zeta_j} = o_d^j d + o_c \frac{\Sigma_i \zeta_i}{n_c} + o_s s_j + o_r r_j + o_t log(t_o - t_j) \tag{8}$$

Forgetting One of the highlights of the implementation of this architecture is the forgetting reasoner. It enables

As an extension of the SIT algorithm, the forgetting function $\mathcal{F}$ is defined. This function is called by the consolidating function, as soon as it finishes updating the scores of one or more nodes (Equation 9). The forgetting function evaluates the updated scores u_j and assesses if it is necessary to change the level of the node. If the new level of the node is either HIGH or LOW the corresponding node remains in the ontology, whereas a node whose level is changed to WEAK is deleted from the memory [6].

$$\begin{aligned} \mathcal{F} : \mathbb{X}, [0,1] &\rightarrow \mathbb{X} \\ X, d_j &\mapsto X \end{aligned} \tag{9}$$

Retrieving There are different models to understand and reproduce the process the human brain undergoes while trying to remember a specific information or situation. Two quite popular models are the spreading activation [16] and the compound cue [17]. The spreading activation model allows also to retrieve related events and not only a exact match. This is a behaviour analogous to the memory association in the human mind, when we get reminded of another event by a similar event. In artificial neural networks this is usually implemented by a propagation around the primarily activated node, that arbitrarily activates also the nodes in the closer neighborhood (the closer the higher the associated activation of another node). The compound cue in contrast allows the retrieval of the most relevant event. The determination is made on the basis of the degree of similarity by comparing a compound item to all existing items of the memory [18]. Implemented in the SIT algorithm used in this project is the compound cue. The algorithm compares the similarity value ${}^{j}d_\epsilon$ and returns the most suitable results.

2.2.4.3 Technical Realization

Besides the encoding reasoner, all reasoners are implemented as subclasses of the superclass `ReasonerBase` that contains the ontology instances.

Storing Uses the `store()` method that increases the storing counter, if the given node is recognized by the corresponding interface or adds a new node (initialized by `buildScene()`) if it is an unknown scene.

Retrieving Evaluates if the node given by the dialogue module (`recognize()`) is in the graph. In case a complete match is found the retrieving counter of all nodes of the branch are increased. Otherwise it creates a 'dummy node' with `addNode()` and increases the retrieving score for the parent nodes and finally deletes the dummy node `remove()`.

Consolidating Evaluates the scores of the branch (`EvaluateScore()`) of the last modified node, as each nodes score also takes into account the children's scores. Finally calls `normalization()` in order make the scores of the complete tree comparable.

Forgetting Reads all scores of the graph and evaluates their levels. If a node turns out to have a weak level (`EvaluateLevel()`) it is deleted from the graph (`deletenode()`).

The reasoner classes use functional interfaces to query, manipulate and maintain the ontology. Even though technically every reasoner can use every implemented interface, only the `PropertyManager` interface is used by all of them (Figure 7).

`Recognition` Scans the graph in order to find the nodes that match the ones given by the caller. Updates the retrieving and storing counter (r_j and s_j)

`Clean` Removes all the geometric primitive Individuals from the ontology and the scene individuals from the graph.

`BuildScene` Reads all the geometric primitives from the ontology and saves them into a vector of objects.

`AddNode` Add a node to the graph in the correct position (increasing complexity from root to leaves) and number the nodes in a consecutive manner

`GetBranch` Contains functions that scan the graph to either evaluate the score of each node or the normalization factor.

`Remove` Deletes a node from the graph and puts it into the trash bin.

`PropertyManager` Reads the properties (scores and counters), initializes properties to a default value and evaluates the level for a node

3 Implementation

3.1 System Setup

Software In order to successfully run all components of the project, it is neccessary to install and include a number of programms and libraries. In the following table the required software components are listed as well as the webadress under which they can be downloaded and the installing instructions can be found (Table 3). We emphasize that at this moment rosJava is not compatible with newer versions of the operating system Ubuntu than 16.04.

Hardware Besides a computer that meets the requirements of the listed software a visual and auditive perception device that is compatible with the software to be run is needed, such as the 'Microsoft Kinect'.

Table 3: **Software Overview.**

Step	Program	Version	Type	Url
1	Ubuntu	16.04.5 LTS (Xenial)	OS	http://releases.ubuntu.com/16.04/
2	ROS	Kinetic	Framework	http://www.ros.org/install/
3	rosJava		Library	http://wiki.ros.org/rosjava
4	ARMOR	ARMOR2	Library	https://github.com/EmaroLab/armor
5	injected ARMOR		Library	https://github.com/EmaroLab/injected_armor_pkgs
6	SIT		Library	https://github.com/EmaroLab/scene_identification_tagging
7	PITT		Library	https://github.com/EmaroLab/primitive_identification_tracking_tagging
8	protege		Ontology Editor	https://protege.stanford.edu/
	intelliJ IDEA		IDE	https://www.jetbrains.com/idea/

3.2 How to Run the Project

3.2.1 Vision Module - Microsoft Kinect

In order to compile the vision module, the following steps are required.

1. Download the Pitt_msgs, Pitt_object_table_segmentation and pitt_geometric_tracking repositories.
2. Download and install the armor package files.
3. Connect a Kinect or similar vision sensor that provides point cloud data
4. catkin_make
5. roslaunch pitt_object_table_segmentation table_segmentation.launch
6. rostopic list and check for the topic the camera is using. It should be either /camera or /cameraB and the same should be mentioned in the roslaunch file wherever necessary.

3.2.2 Reasoning Module

In order to compile the reasoning module, the following steps are required.

1. Download and install the above mentioned programs and libraries.
2. Create a workspace that contains rosjava.
3. Download the folder injected_armor_pkgs and copy them into the src folder of your rosjava workspace.
4. Open the main '/rosjava/src/injected_armor_pkgs/injected_armor/sit/src/main/java/it/emarolab/sit' with an IDE.
5. Update the ONTO_PATH to the folder of your computer.
6. Change the ONTO_FILE path in the main.
7. Compile the main.
8. Launch the main.

4 Results

Vision Module

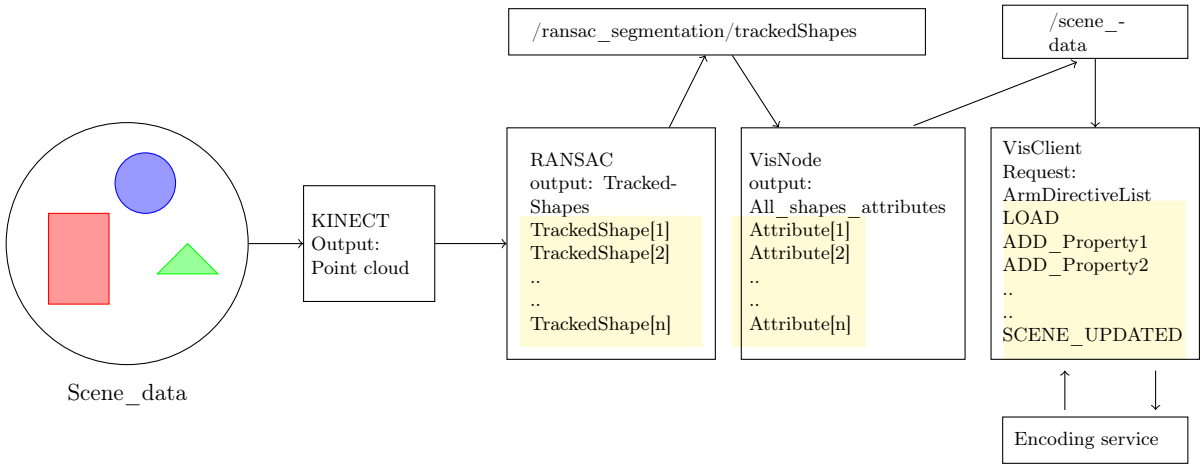
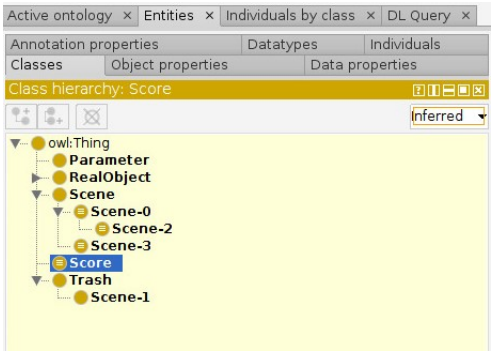


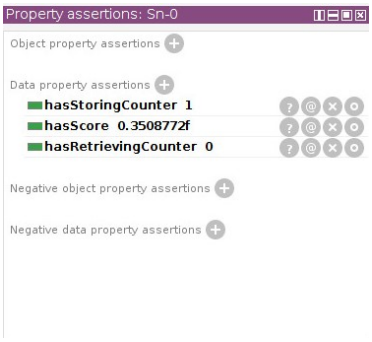
Figure 8: Input output for the vision module

Reasoning Module

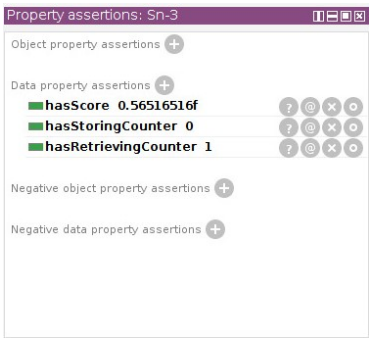
After running the script and updating protege, the ontology is represented and all properties are displayed (Figure 9).



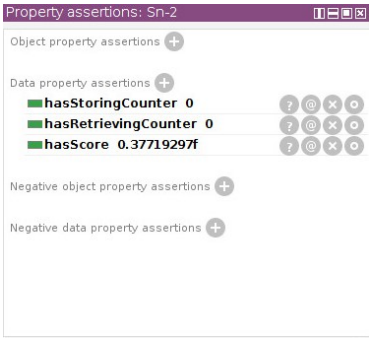
(a) Graph Representation with a deleted scene (Scene-1).



(b) Scene recently perceived and recognized.



(c) Scene recently retrieved.



(d) Scene recently nor recognized nor retrieved.

Figure 9: **Properties of Scenes Displayed in Protege.**

5 Recommendations

So far, the components were tested individually and the necessary provided interfaces were simulated.

Vision Module

- Currently, we only manipulate the data properties common in sphere, cylinder and cone. The generalization of this idea can be taken up as future work in order to send more information about the scene leading to a richer memory.
- The current module can also be implemented after merging the data with the tensorflow structure. This will provide better results from merged data with added capability of statistical analysis and operations.
- The color detection service can detect only red, green, blue, yellow and pink colors. If the object in front of the kinect is not of one of these colors then it is mandatory to comment the use of color_service related code in ransac_segmentation.cpp.
- After roslaunch, the user might see red comments saying cannot choose a shape from one point. This is not an error but the output of RANSAC segmentation signifying that the algorithm has reduced the given point cloud to the threshold set.
- Currently, the VisNode runs only for one scene.This is due to the fact that PITT is not robust and the same scene is interpreted as new scenes because of loss of detection or noise. The user has to reinitialize VisNode if the scene has actually changed. After the PITT is robust enough, we can simply reinitialize the attributes in the VisNode. It is currently commented in the VisNode.cpp.

Reasoning Module

- The retrieving reasoner is not completed, so far it is only implemented for the recognition of a full matching scene. It needs to be completed such that it retrieves also scenes that are similar but not exactly the same.
- Due to an unknown bug that is probably located in the OWLOOP it is necessary to close and re-open the ontology after writing data in the ontology.
- We suspect a second bug in the SIT recognition function that seems to be affect the real-time behaviour.
- Currently, the reasoners are implemented for the semantic memory only. The episodic reasoners need to be created analogously. The modular structure of the code permits to achieve this by simply implementing new interfaces that include the episodic properties. The storing methods have to be extended with a synchronization to prevent forgetting an item only in one memory.

References

[1] Jochen J. Steil. "Memory in Backpropagation-Decorrelation O(N) Efficient Online Recurrent Learning". In: (2005), pp. 649–654. DOI: 10.1007/11550907{_}103. URL: https://link.springer.com/chapter/10.1007/11550907_103.

[2] Michael McCloskey and Neal J. Cohen. "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem". In: *Psychology of Learning and Motivation* 24 (Jan. 1989), pp. 109–165. ISSN: 0079-7421. DOI: 10.1016/S0079-7421(08)60536-8. URL: https://www.sciencedirect.com/science/article/pii/S0079742108605368.

[3] RC Atkinson et al. "Human memory: A proposed system and its control processes". In: *Elsevier* (). URL: https://www.sciencedirect.com/science/article/pii/S0079742108604223.

[4] Jochen Braun et al. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information". In: (). URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.308.8071.

[5] Kurzon. *Multi-store Model*. URL: https://en.wikipedia.org/wiki/Atkinson%E2%80%93Shiffrin_memory_model.

[6] Luca Buoncompagni. "Maintaining Structured Experiences for Robots via Human Demonstrations: An Architecture To Supervise Long-Term Robot's Beliefs." PhD thesis. Genova: University of Genova, 2019.

[7] S. Matthew Liao and Anders Sandberg. "The Normativity of Memory Modification". In: *Neuroethics* 1.2 (July 2008), pp. 85–99. ISSN: 1874-5490. DOI: 10.1007/s12152-008-9009-5. URL: http://link.springer.com/10.1007/s12152-008-9009-5.

[8] Larry R. Squire. "Memory systems of the brain: A brief history and current perspective". In: *Neurobiology of Learning and Memory* 82.3 (Nov. 2004), pp. 171–177. ISSN: 1074-7427. DOI: 10.1016/J.NLM.2004.06.005. URL: https://www.sciencedirect.com/science/article/pii/S1074742704000735?via%3Dihub.

[9] Andrew Kusiak and Andrew. *Engineering design : products, processes, and systems*. Academic, 1999, p. 427. ISBN: 0124301452. URL: https://dl.acm.org/citation.cfm?id=519665.

[10] Brugali, Brugali, and Patrizia Scandurra. "Component-based Robotic Engineering. Part I: Reusable building blocks". In: *IEEE ROBOTICS AND AUTOMATION MAGAZINE* (2009). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.466.3373.

[11] Davide Brugali and Azamat Shakhimardanov. "Component-based Robotic Engineering (Part II)". In: *IEEE Robotics & Automation Magazine* 17.1 (Mar. 2010), pp. 100–112. DOI: 10.1109/MRA.2010.935798. URL: http://ieeexplore.ieee.org/document/5430399/.

[12] *SIT*. URL: https://github.com/EmaroLab/scene_identification_tagging.

[13] Yadin Dudai. "The Neurobiology of Consolidations, Or, How Stable is the Engram?" In: *Annual Review of Psychology* 55.1 (Feb. 2004), pp. 51–86. ISSN: 0066-4308. DOI: 10.1146/annurev.psych.55.090902.142050. URL: http://www.annualreviews.org/doi/10.1146/annurev.psych.55.090902.142050.

[14] Larry R Squire and Pablo Alvarez. "Retrograde amnesia and memory consolidation: a neurobiological perspective". In: *Current Opinion in Neurobiology* 5.2 (Apr. 1995), pp. 169–177. ISSN: 0959-4388. DOI: 10.1016/0959-4388(95)80023-9. URL: https://www.sciencedirect.com/science/article/pii/0959438895800239?via%3Dihub.

[15] Natalie C. Tronson and Jane R. Taylor. "Molecular mechanisms of memory reconsolidation". In: *Nature Reviews Neuroscience* 8.4 (Apr. 2007), pp. 262–275. ISSN: 1471-003X. DOI: 10.1038/nrn2090. URL: http://www.nature.com/articles/nrn2090.

[16] John R. Anderson. "A spreading activation theory of memory". In: *Journal of Verbal Learning and Verbal Behavior* 22.3 (June 1983), pp. 261–295. ISSN: 0022-5371. DOI: 10.1016/S0022-5371(83)90201-3. URL: https://www.sciencedirect.com/science/article/pii/S0022537183902013.

[17] R Ratcliff and G McKoon. "Retrieving information from memory: spreading-activation theories versus compound-cue theories." In: *Psychological review* 101.1 (Jan. 1994), pp. 177–84. ISSN: 0033-295X. URL: http://www.ncbi.nlm.nih.gov/pubmed/8121958.

[18] Mei Yii Lim et al. "Human-like memory retrieval mechanisms for social companions". In: *undefined* (2011). URL: https://www.semanticscholar.org/paper/Human-like-memory-retrieval-mechanisms-for-social-Lim-Aylett/09a5439864584647896264 1af25e1b78c017660.