University of Bremen
Institute for
Artificial Intelligence

Memory modules for a cognitive architecture to execute complex manipulation tasks for Personal Service Robots

Lisset Yazmín Salinas Pinacho

Vollständiger Abdruck der vom Fachbereich 3 (Mathematik und Informatik) der Universität Bremen zur Erlangung des akademischen Grades eines

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Prof. Gabriel Zachmann
1. Prüfer:	Prof. Michael Beetz *Universität Bremen*
2. Prüfer:	Prof. Karinne Ramírez Amaro *Chalmers University of Technology*
Beisitz:	Dr. Diar Abdlkarim *University of Birmingham*

Die Dissertation wurde am 15.12.2023 bei der Universität Bremen eingereicht und durch den Prüfungsausschuss am 27.03.2024 angenommen.

Abstract

This work presents an approach to improving the autonomy of robots. This is done by adding cognitive features with the aim of allowing these robots to be used in homes or care facilities with little supervision. Specifically, my approach is to build knowledge-processing memory modules capable of acquiring, retrieving, storing and updating knowledge. Using modules that simulate memory allows robots to process knowledge and perform better in human environments. In particular, this work focuses on Personal Service Robots executing complex manipulation actions embedded into everyday tasks.

I aim to use techniques of learning from human examples to enable robots to perform everyday tasks in household environments. A human activity recognition system for demonstrations from a Virtual Reality environment was implemented to record and analyze those examples. This system is capable of segmenting trajectories and events into actions and sub-actions. These are stored in a hierarchical data structure called episodes, used by the memory modules to give the robot insight into how to execute such actions.

Also, this research developed knowledge-processing memory modules to extend a cognitive architecture. These memory modules are part of Long Term and Short Term Memory. This architecture is transferable to diverse robotic platforms. It was tested in five robots from different vendors and body configurations. This shows that the knowledge can be used depending on the robot's necessities and capabilities. The memory modules allow the robot to decide whether and how to adapt actions. This way, the robots can perform complex manipulation actions similar to humans.

The experiments in this work were performed in a simulated kitchen environment. There, the five robots performed actions embedded in the cooking tasks. The results show an improvement in the action execution success by eleven percent and a similar percentage reduction in the time required to perform the tasks.

This work draws inspiration from linguistics, psychology, neuroscience, cognitive science and computer science to create Knowledge Bases. They contain concepts from natural sciences, actions, skills, objects, robotic platform capabilities and experience.

Resumen

Este trabajo presenta una propuesta para mejorar la autonomia de los robots. Esto, por medio de agregar capacidades cognitivas para permitir a estos robots ser usados en hogares o instituciones del cuidado de la salud con poca supervisión. En especial, mi propuesta es construir módulos de memoria para procesar conocimiento capaces de adquirir, recuperar, guardar y actualizar el conocimiento. Al usar módulos que simulan la memoria, los robots pueden procesar conocimiento y mejorar la ejecución de tareas en ambientes humanos. En particular, este trabajo se enfoca en Robots Personales de Servicio ejecutando manipulación complejas de objetos relacionados con la tareas diarias.

Mi objetivo es utilizar técnicas de aprendizaje a partir de demostraciones humanas para permitir que los robots realicen tareas cotidianas en ambientes domésticos. Para ello, se implementó un sistema de reconocimiento de actividades humanas en un ambiente de Realidad para registrar y analizar esos ejemplos. Este sistema es capaz de segmentar trayectorias y eventos en acciones y subacciones. Estos se almacenan en bases de datos con una estructura jerárquica llamada episodios, utilizada por los módulos de memoria para darle al robot una idea de cómo ejecutar dichas acciones.

Además, esta investigación desarrolló módulos de memoria para procesar conocimiento que amplian una arquitectura cognitiva. Estos módulos de memoria forman parte de la Memoria de Largo y de Corto Plazo. Esta arquitectura es transferible a diversas plataformas robóticas. Fue probada en cinco robots de diferentes proveedores y configuraciones físicas. Esto demuestra que el conocimiento se puede utilizar dependiendo de las necesidades y capacidades de cada robot. Los módulos de memoria permiten a cada robot decidir si adapta las acciones y cómo. De esta manera, los robots pueden realizar acciones de manipulación complejas de forma similar a la de los humanos.

Los experimentos de este trabajo se realizaron en una cocina dentro de un entorno simulado. Allí, los cinco robots realizaron acciones relacionadas con las tareas de cocina. Los resultados muestran una mejora en el éxito de la ejecución en un once por ciento y una reducción similar en el tiempo requerido para realizar las tareas.

Este trabajo se inspira en la lingüística, la psicología, la neurociencia, las ciencias cognitivas y las ciencias de la computación para crear Bases de Conocimiento. Estas contienen los conceptos de ciencias naturales, acciones, habilidades, objetos, capacidades y experiencia de los robots.

Zusammenfassung

In dieser Arbeit wird ein Ansatz zur Verbesserung der Autonomie von Robotern vorgestellt. Dies geschieht durch Hinzufügen von kognitiven Funktionen mit dem Ziel, dass diese Roboter in Heimen oder Pflegeeinrichtungen mit wenig Aufsicht eingesetzt werden können. Konkret besteht mein Ansatz darin, Gedächtnismodule zu implementieren, die in der Lage sind, Wissen zu erwerben, abzurufen, zu speichern und zu aktualisieren und so bessere Leistungen in menschlicher Umgebung zu erbringen.

Hierzu nutze ich Techniken des Lernens von menschlichen Beispielen. Ein System zur Erkennung menschlicher Aktivitäten für Demonstrationen in einer Virtual Reality-Umgebung wurde implementiert, um diese Beispiele aufzuzeichnen und zu analysieren. Das System ist in der Lage, Trajektorien und Ereignisse in Aktionen und Unteraktionen zu segmentieren. Diese werden in einer *Episoden* genannten hierarchischen Datenstruktur gespeichert, die von den Gedächtnismodulen verwendet wird, um dem Roboter die Ausführung alltäglicher Aufgaben in häuslichen Umgebungen zu ermöglichen.

Diese Gedächtnismodule wurden im Rahmen dieser Forschung entwickelt, um eine kognitive Architektur zu erweitern. Diese Architektur ist auf verschiedene Roboterplattformen übertragbar. Sie wurde in fünf Robotern verschiedener Hersteller und Körperkonfigurationen getestet. Dies zeigt, dass das Wissen je nach Fähigkeiten der Plattform genutzt werden kann. Die Gedächtnismodule ermöglichen es den Robotern zu entscheiden, ob und wie sie ihre Handlungen anpassen sollen. Auf diese Weise können sie komplexe Manipulationsaktionen ähnlich wie Menschen durchführen, wobei sich diese Arbeit insbesondere auf Personal Service-Roboter konzentriert.

Die Experimente in dieser Arbeit wurden in einer simulierten Küchenumgebung durchgeführt. Dort führten die fünf Roboter Aktionen aus, die in Kochaufgaben eingebettet waren. Die Ergebnisse zeigen eine Verbesserung des Erfolgs bei der Aktionsausführung um elf Prozent und eine ähnliche prozentuale Verringerung der Zeit, die für die Ausführung der Aufgaben benötigt wurde.

Diese Arbeit wird von der Linguistik, der Psychologie, den Neurowissenschaften, der Kognitionswissenschaft und der Informatik inspiriert, um Wissensdatenbanken zu erstellen. Sie enthalten Konzepte aus den Naturwissenschaften, Handlungen, Objekte, Fähigkeiten und Erfahrungen von Roboterplattformen.

Acknowledgments

First, I want to thank my supervisor, Prof. Michael Beetz, and my reviewer, Prof. Karinne Ramírez-Amaro, for all their support and advice. I also want to thank Prof. Gabriel Zachman, Dr. Diar Abdlkarim, Susana Simancas and Olivia Kohler, who kindly took the time and consented to serve on my thesis committee.

I would like to thank all of my colleagues from the Institute for Artificial Intelligence at the University of Bremen, with whom I had the great honor of working. My special thanks go to Dr. Hagen Langer, who took the time to review my work and gave me valuable advice. I would also like to thank Alexander Wich, Fereshta Razdani, Feroz Siddiky and Michaela Kümpel, with whom I also found a friendship to last a lifetime. My thanks go to the university's administration team, without whom the process would be more complicated, especially to Andrea Cowley, Silke Völkers and Dr. Marie Saade for all their support.

My gratitude goes especially to Matthias Schneider for his companionship and patience. He read this work many times and his insights made it a beautiful document. My gratitude also goes to my mother and friend, Leticia Pinacho de la O, who gave me the strength to follow my dreams and all possible support I would need; without her, I would not be the strong person I am now.

I had the support of family and friends who live around the world. All of them have an exceptional place in my heart. Thank you to my family and friends in Mexico. My special thanks go to Juan Carlos and Erika, my beloved siblings with whom I still have the best adventures. Thank you, Cinthya Ceja, for the excellent drawings. Thank you, Fernando, Zulema, Carmen, Caleb, Montserrat, Alejandra, for being there for me in the distance. Thank you to my family in England. Thank you to my friends and family in Germany. Mainly thanks to Wolfgang and Ulla for their language and nutritional support. Thank you to the musketeers, who will continue their journey on different paths but always be together in heart. Thank you, my Pomodoros group, for working alongside me, no matter the storm. To my lunch group in Bremen, you made this journey more pleasant with many laughs and great discussions.

April 18, 2024,

Lisset Yazmín Salinas Pinacho

I gratefully acknowledge the German Academic Exchange Service (DAAD) funding, which made my Ph.D. work possible.

Contents

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Problem description . . . 2
  - 1.2.1 Manipulation action knowledge acquisition . . . 6
  - 1.2.2 Knowledge-processing memory modules . . . 7
  - 1.2.3 Knowledge storage . . . 9
  - 1.2.4 Testing scenario . . . 10
- 1.3 Contributions . . . 10
- 1.4 Reader's guide . . . 12

2 Foundations 15
- 2.1 An introduction to Personal Service Robots . . . 16
  - 2.1.1 Body design . . . 16
  - 2.1.2 Personal Service Robot definition . . . 16
- 2.2 Robotic communication software . . . 19
- 2.3 Robotic learning . . . 19
  - 2.3.1 Clustering . . . 21
  - 2.3.2 Decision tree . . . 22
  - 2.3.3 Reinforcement Learning . . . 24
  - 2.3.4 Artificial Neural Networks . . . 26
- 2.4 Robot's action execution control . . . 34
  - 2.4.1 Robotic architectures . . . 34
  - 2.4.2 Cognitive architectures . . . 35
  - 2.4.3 The use of memory and learning in cognitive architectures . . . 37
- 2.5 Receiving instructions in natural language and parsing them . . . 44
- 2.6 Summary of this chapter . . . 46

3 Overview of the knowledge processing memory modules 47
- 3.1 Existing systems . . . 53
  - 3.1.1 KnowRob . . . 53
  - 3.1.2 Episode or NEEM . . . 55
  - 3.1.3 NaivPhys4RP before RoboSherlock . . . 55

3.1.4 MoveIt! . . . 57
3.2 Knowledge processing memory-based models . . . 58
3.2.1 Working Memory to interconnect memories . . . 58
3.2.2 Episodic Memory to get previous experiences . . . 59
3.2.3 Procedural Memory in action representation . . . 61
3.2.4 Semantic Memory representing objects and actions as concepts . . . 61
3.3 Summary of this chapter . . . 62

4 Knowledge handling by Personal Service Robots 63
4.1 Concept of knowledge . . . 64
4.1.1 Knowledge representation . . . 65
4.1.2 Knowledge retrieval and reasoning . . . 74
4.2 Knowledge system for a Personal Service Robot . . . 75
4.2.1 Action representation by a Personal Service Robot . . . 76
4.2.2 Semantic Knowledge Base . . . 80
4.2.3 Procedural Knowledge Base . . . 83
4.2.4 Episodic Knowledge Base . . . 84
4.3 Summary of this chapter . . . 86

5 Human manipulation action recognition from virtual reality data 87
5.1 Manipulation actions . . . 89
5.1.1 Types of actions . . . 90
5.1.2 Actions in real or virtual environments . . . 90
5.1.3 Action segmentation, classification and recognition . . . 91
5.1.4 Learning from Demonstrations . . . 94
5.1.5 Datasets . . . 96
5.2 Human Action Recognition from Virtual Reality data . . . 97
5.2.1 Sub-action segmentation with if-then and if-then-else rules . . . 100
5.2.2 The sub-actions classifier by a decision tree . . . 104
5.2.3 The sub-actions classifier by a Recurrent Neural Network . . . 108
5.2.4 Action and sub-action prediction . . . 113
5.3 Summary of this chapter . . . 115

6 Knowledge processing memory models for Personal Service Robots 117
6.1 Concept of memory . . . 118
6.1.1 Types of memory . . . 118
6.1.2 The use of memory concepts and interconnection models . . . 123
6.2 The use of memory concepts by Personal Service Robots . . . 125
6.2.1 The Working Memory model . . . 126
6.2.2 The Episodic Memory model . . . 131
6.2.3 The Procedural Memory model . . . 135
6.2.4 The Semantic Memory model . . . 138

6.3 Testing the memory modules . . . 143
6.3.1 Table setting experiment . . . 147
6.3.2 Bring a drink to the table experiment . . . 152
6.3.3 Serve and bring soup test . . . 154
6.3.4 Serve a drink test . . . 157
6.3.5 Pour and mix ingredients test . . . 159
6.4 Summary of this chapter . . . 163

7 Conclusions **165**
7.1 Human Action Recognition . . . 167
7.2 Knowledge processing modules . . . 167
7.2.1 Working memory . . . 168
7.2.2 Episodic memory and knowledge base . . . 169
7.2.3 Procedural memory and knowledge base . . . 170
7.2.4 Semantic memory and knowledge base . . . 171
7.3 Summary . . . 172

A Competitions for Personal Service Robots **177**
A.1 Personal Service Robots commercially available . . . 178
A.1.1 Functionalities . . . 180
A.2 Robotic prototypes from research institutions . . . 181
A.2.1 Functionalities . . . 184

B Action and sub-action plans for solving specific tasks **187**
B.1 Pick and place a bowl and a cup task . . . 187
B.2 Bring a drink from the fridge to the table task . . . 189
B.3 Serving soup task . . . 190
B.4 Serving juice task . . . 191
B.5 Mixing ingredients for batter task . . . 192

C List of Publications **193**
C.1 Own published papers . . . 193
C.2 Awards . . . 194

Glossary **195**

Bibliography **199**

List of Figures

1.1 Robot asking itself how to perform an action. . . . 3
1.2 Robot bringing a bowl of soup to the table. . . . 4
1.3 Robot serving juice from a box container into a glass. . . . 4
1.4 Robot mixing ingredients. . . . 4
1.5 Knowledge-processing memory modules. . . . 5
1.6 Use of examples by a robot to perform a task. . . . 7
1.7 Processing memory modules with their Knowledge Bases. . . . 9

2.1 Personal service robots' latest version at present and first release year. . . . 17
2.2 Machine learning process. . . . 20
2.3 Hierarchy of learning methods. . . . 20
2.4 General clustering result example. . . . 22
2.5 Agglomerative hierarchical clustering example. . . . 22
2.6 Decision tree graphic representation. . . . 23
2.7 Neuron process representation. . . . 27
2.8 Activation functions graphs. . . . 30
2.9 Different number of layers with two inputs X1 and X2 and one output Y. . . . 31
2.10 Recurrent Neural Network with one feedforward. . . . 32
2.11 Recurrent Neural Network process. . . . 32
2.12 Recurrent Neural Network with LSTM process. . . . 33
2.13 Representation of layers in a vertical fashion with different components each layer can include. . . . 35
2.14 Cognitive architectures taxonomy based on representation and processing. . . . 36
2.15 Representation of memory types used in hybrid cognitive architectures. . . . 37
2.16 Comparison between hybrid cognitive architectures related to their use of memory. 39
2.17 Comparison between hybrid cognitive architectures related to their learning processes. . . . 42
2.18 Parser example of instruction with an implicit subject or proper noun. . . . 44
2.19 PRAC key concepts and their role in inferring most probable instruction. . . . 45
2.20 PRAC usage example. . . . 45

3.1 Robot pouring juice from a box container into a glass. . . . 47

3.2 Memory-based knowledge processing modules inside the cognitive architecture CRAM. . . . 48
3.3 KNOWROB system overview. . . . 54
3.4 Example of ROBOSHERLOCK execution. . . . 56
3.5 MoveIt! move_group architecture. . . . 57
3.6 HAR process steps. . . . 60
3.7 Sub-action tree representation in episodes with its action and features. . . . 60
3.8 Procedural Knowledge Base representation. . . . 61

4.1 Knowledge types in *Artificial Intelligence (AI)*. . . . 67
4.2 Knowledge representation methods. . . . 68
4.3 Task tree with levels of representation for tasks, actions, sub-actions and movements. . . . 76
4.4 Trajectory extracted from a human demonstration for `putting` cheese on pizza dough. . . . 79
4.5 Epic Kitchen dataset noun additions to *Semantic Knowledge Base*. . . . 82
4.6 Action additions to *Semantic Knowledge Base*. . . . 82
4.7 Simplified example of actions and sub-actions present in the *Procedural Knowledge Base*. . . . 84
4.8 Simplified example of global and local features present in the *Episodic Knowledge Base*. . . . 85

5.1 Example of `Reaching` sub-action for a Glass with the `RightHand`. . . . 91
5.2 Example of data separation for a learning algorithm. . . . 94
5.3 From VR to episodes. . . . 96
5.4 HAR process. . . . 99
5.5 Minimized decision tree result for `OpeningADrawer` and `ClosingADrawer` actions for right hand. . . . 107
5.6 Normalized confusion matrix sub-action prediction for `OpeningADrawer` and `ClosingADrawer` actions for both hands. . . . 108
5.7 RNN with LSTM architecture for sub-action classification. . . . 109
5.8 Sub-actions evaluation for random inputs for `OpeningADrawer` and `ClosingADrawer` actions. . . . 112
5.9 Random input for RNN sub-action classification loss. . . . 112
5.10 HAR representation example. . . . 115

6.1 Memory type division. . . . 119
6.2 Memory architecture for knowledge flow. . . . 125
6.3 Working memory building an execution plan using Procedural and Episodic Knowledge Bases. . . . 127
6.4 Working memory sets actions for execution. . . . 128
6.5 Working memory uses perception information. . . . 129

6.6 Episodic memory asking about specific actions to build recurrent action to the *Episodic Knowledge Base (EKB)* and used by working memory. 133
6.7 Initial Procedural Knowledge Base action structure from Semantic Knowledge Base. 136
6.8 Procedural memory adds a temporal action structure to Procedural Knowledge Base. 137
6.9 Semantic Knowledge Base upper ontology with main classes for physics, temporal and spatial things, actions and sub-actions. 138
6.10 Working memory sends the knowledge to the Semantic Knowledgebase. 140
6.11 Angle for obtaining liquid from a container. 141
6.12 Relation of forces of liquid inside the container. 142
6.13 Angle for obtaining content from a container. 143
6.14 Virtualbox with three virtual machines and operating systems running. 144
6.15 Execution workflow. 145
6.16 Robots performing serving and cooking actions. 146
6.17 PR2 brings bowl and cup to the table. 147
6.18 Executing time when executing the pick and place task with and without the use of knowledge processing memory modules. 150
6.19 Failures presented when executing the pick and place task with and without the use of knowledge processing memory modules. 151
6.20 Robots bring drink to the table results. 154
6.21 Robots bring drink to the table results. 157
6.22 Robots bring drink to the table results. 159
6.23 Robots add and mix ingredients for a batter results. 162

7.1 Extended CRAM with Memory-based knowledge processing modules. 166
7.2 Episode representation example. 169
7.3 Episodic Knowledge Base global and local features. 170
7.4 Simplified example of actions and sub-actions present in the *Procedural Knowledge Base*. 171
7.5 Semantic Knowledge Base upper ontology with main classes for temporal and spatial things, actions and sub-actions. 172
7.6 Robot pick and place bowl from counter and bring them to the table. 173
7.7 Robot bringing drink to the table. 173
7.8 Robot bringing bowl with soup to the table. 174
7.9 Robot serving juice from a box container into a glass. 174
7.10 Robot mixing ingredients inside a bowl. 174

B.1 Task tree of robots picking and placing a cup from the counter to the table. . . . 187
B.2 Actions and sub-actions involved in the task of picking and placing a cup and bowl from the counter to the table. 188
B.3 Task tree of robots bringing a drink from the fringe to the table. 189

B.4 Specific actions and sub-actions to solve the bringing a drink from the fringe to the table task. 189
B.5 Task tree of robots serving soup. 190
B.6 Specific actions and sub-actions required to serving soup. 190
B.7 Task tree of robots serving juice. 191
B.8 Task tree of robots mixing ingredients for batter. 192
B.9 Task tree of robots mixing sub-actions. 192

List of Tables

2.1 Cognitive architectures using Working and Episodic Memory with its types. . . 40
2.2 Cognitive architectures using associative learning and their type. 43

4.1 Comparison between Knowledge Base approaches. 74
4.2 Manual and locomotion verbs. 77

5.1 Manual and locomotion verbs. 98
5.2 Segmentation sub-action rule features. 102
5.3 Comparison of confidence scores between decision trees and RNN. 113

6.1 Actions and sub-actions present in the pick and place task. 148
6.2 Comparison between the use of memory models and no use. 149
6.3 Comparison between two-arm robots task execution. 153
6.4 Comparison between one-arm robots task execution. 153
6.5 Comparison between robots serving soup task execution. 156
6.6 Comparison between robots bringing soup task execution. 156
6.7 Comparison between robots serving juice task execution. 159
6.8 Physical properties of various products. 160
6.9 Friction force of various objects. 161
6.10 Comparison between robots pouring ingredients task execution. 161
6.11 Comparison between robots mixing ingredients task execution. 162

A.1 PSRs developed by industry from 2000 to present order by released year. 179
A.2 PSRs developed by academia from 2000 to present order by release year. 182

Chapter 1

Introduction

The increase in life expectancy and the low birth rate have, as a consequence, an increase in the aging population in the world [Lutz et al., 2008], which brings the necessity for more caregiving personnel. However, this would increase the cost of elder care to provide independent living support for everyone who requires it [Hashimoto et al., 2013]. Furthermore, the COVID-19 pandemic showed the importance of deploying robots in caregiving facilities. However, even though robots have the potential to support or replace human service employees, specifically, Personal Service Robots (PSRs) acting in home environments and care facilities, consumers need to trust them more. This mistrust makes people unwilling to use them to perform such caring tasks [van Pinxteren et al., 2019].

Some robots without a manipulator achieved some of these caring tasks, such as giving reminders and social interaction. However, tasks related to object manipulation, such as getting a drink, reaching things on high shelves or the floor, finding and bringing items, and preparing meals, still need to be completely available. Such manipulation tasks are identified in the Program of All-Inclusive Care for the Elderly (PACE) [Johnson et al., 2017]. In this sense, many robotic platforms can pick and place free-form objects or open and close devices like a fridge. However, most still need some functionalities and autonomy for caregiving work. This lack of functionalities was shown in the three-week trial by Martinez-Martin and del Pobil [2018] in real private homes, where the robot could search, transport and bring small objects, give reminders and detect emergencies. However, robot errors while performing these tasks led users to frustration and mistrust. This mistrust shows that even though a Personal Service Robot (PSR)

can perform specific tasks, their manipulation capabilities must be improved to be used as caregivers.

This thesis focuses on improving the manipulation capabilities of PSRs. First, the chapter presents the motivation for pursuing this research in Section 1.1. Then, Section 1.2 describes the challenges and problems addressed in this work. Next, Section 1.3 expresses the main contributions of this thesis. Finally, Section 1.4 explains the outline of this document.

1.1 Motivation

The motivation of this work is to give a Personal Service Robot (PSR) the capabilities to serve humans. Those capabilities can be similar to the ones of humans, such as perception, attention, action selection, memory, learning, reasoning and prospection. Specifically, this thesis work is interested in how humans use learned knowledge to select actions and adapt to unknown situations. For this, we can look into the research area of cognitive robotics, which arose from applying new methods to model the human brain's cognitive processes. In the same way, in this work, different knowledge-processing memory modules were implemented to extend the existing cognitive architecture, which was later tested on various robotic bodies.

Another motivation is that living beings perform actions that are not fully understood. Let's take the example of peeling an orange; a person could select a knife or a peeler as a tool, depending on what is available. Then that person has to manipulate the orange and tool together to have a specific contact to start the movements required for the peeling action. All these actions are represented somehow in the person's brain. Such representations together provide an action plan. This work looks better at understanding actions and their representation concerning plan construction. Knowledge is represented and stored in memory for that to be possible, so a PSR can use it to build execution plans. This work follows the premise that manipulation depends not only on complex motion control but on other cognitive processes-related elements. One motivation for this work is that when we build and program robots, we better understand their limitations and how their movements are generated.

1.2 Problem description

Regarding robots' missing manipulation capabilities to date, they cannot easily perform actions like hand-wash dishes or peeling vegetables [Billard and Kragic, 2019]. This is because even when robots can handle some variations in their environment regarding object manipulation, these variations are still limited. For example, robots can manipulate objects with specific properties but have yet to handle soft or deformable objects reliably and efficiently. Furthermore, they still need improvement regarding stable and optimal gasp related to the proper object orientation. These issues mainly concern the position of fingertip contact on the object surface

and, importantly, the purpose of that grasp. Finally, some of these limitations come from the robot's perception capabilities. For example, robots still have issues detecting transparent objects, partially hidden ones or if they are transformed as an effect of one action. For this to be overcome, prior knowledge can be handy. One example is information about deformation models produced by manipulation such as inserting, cutting or bending. Robot knowledge can also include object properties, such as shape, weight, material, viscosity, friction coefficient and so forth. For example, a cup has important physical properties, such as it is hard and fragile because it is made of ceramic. With this knowledge, the robot can also improve its perceptual processing by generating and ranking grasp candidates. The same can happen with manipulating tools by using features from the objects.

Figure 1.1: Robot asking itself how to perform an action.

Primarily, this thesis project aims to provide Personal Service Robots (PSRs) with cognitive knowledge-processing capabilities so they can decide if and how manipulation actions should be adapted so that knowledge is suitably integrated and expanded by each execution to improve performance. For example, when the robot is asked to *bring* a bowl of soup. First, it has to *serve* the soup in a bowl by *measuring* an amount that is a sufficient portion for a person and not too much that can spill. Then, when the bowl is served, the robot should add a restriction to the action `carrying`. Finally, the robot must always obey a specific maximum acceleration and `hold` the bowl upright to avoid spilling anything (Figure 1.2). Serving on the table might be easier in some cases, but the robot should know that the pot might be too heavy to be transported to the table and if not, the same constraints apply.

Another example of constraints the robot must follow is when a robot serves a drink (Figure 1.3). When the robot is `pouring` juice from a box container into a glass, it should consider the box's orientation so that the opening faces the glass but is not too close to it to avoid splashing the liquid. Then, while tilting the container, it should consider that the angle change is slow so as not to spill or splash juice, but the box can get enough air in to allow liquid flow. Also, the robot must be aware that the box deforms when the liquid is coming out, and it has to apply a bit more force slowly to keep it stable but not too much. Finally, the robot should detect when it serves

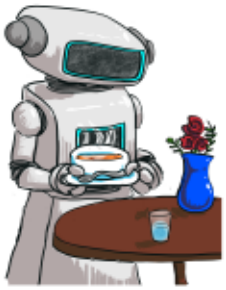


Figure 1.2: Robot bringing a bowl of soup to the table.

the right amount of liquid for a person, but not too much that it will be transportable without spilling it.

Figure 1.3: Robot serving juice from a box container into a glass.

Before baking a cake, the batter must be prepared before it can be put in the oven (Figure 1.4). The constraints, in this case, are related to the speed of the hand. The robot has to `whisk` the ingredients after pouring them into the bowl at different speeds. The first is the speed of the movement of the whisk, such as it must be a little slow as it would splash the ingredients in the kitchen and make a mess, but if it is too slow, it will take very long. The robot must also select the right speed depending on the ingredients already inside the bowl, e.g. the butter with sugar can be mixed at high speed, but the velocity has to decrease when flour and milk are added. Also, the range of movement depends on the size of the bowl.

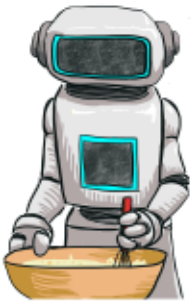


Figure 1.4: Robot mixing ingredients.

Now, think about a robot commanded to *bring* a cup of tea. The robot must solve the task by performing different *actions* in this example. Actions have a hierarchy, e.g. *pouring* hot water into the cup; the robot must *reach*, *grasp*, *lift* and *turn the water kettle*. These so-called basic

or atomic actions are referred here as *sub-actions*. The combination of different sub-actions constitutes one action and the union of actions builds a *plan*. Sub-actions are one level above body part *movements*. Considering that motion involves increasing or decreasing the angle of the joints [Saladin, 2004]. Two examples of body movements are flexion and extension. For example, bending the head or spine is flexion, while any posterior-going movement is an extension. These movements occur at the shoulder, hip, elbow, knee and wrist joints.

With the ideas presented previously in mind, the hypothesis guiding this work is that implementing knowledge-processing memory models using cognitive science, neuroscience and psychology concepts allows a Personal Service Robot (PSR) to perform manipulation tasks in a human-like way. These modules especially support PSRs when performing complex manipulation actions, e.g. using tools in a human-like way, meaning by experimenting and simulating. During manipulation, the robot can reliably handle failures. Furthermore, it can perform actions where the plan is unknown beforehand.

PSRs require to process an enormous amount of knowledge to act in human environments, especially performing caregiving tasks autonomously. This thesis work provides cognitive knowledge-processing capabilities, so PSRs can acquire, store and process knowledge from actions. This allows them to improve their performance and avoid failures. This type of processing capability is only sometimes available in robotic architectures.

This thesis work contributes to cognitive robotics by describing the development of four primary human memory categories shown in Figure 1.5 (blue) and their application to robotic systems. *Semantic Memory (SM)*, *Procedural Memory (PM)* and *Episodic Memory (EM)* are general-purpose storage and retrieval systems. Furthermore, the *Working Memory (WM)* adapts and recalls (semi-autonomously) relevant memories given a current situation, which is a relatively new notion.

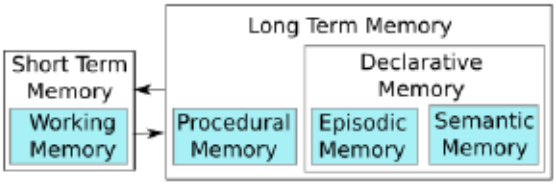


Figure 1.5: Knowledge-processing memory modules.

Regarding planning capabilities, it is important to notice that manipulating objects is easier for robots. We tend to forget that humans are born with basic grasp reflexes [Thibaut and Toussaint, 2010] that improve thanks to imitation and testing of our limbs. The human adult-equivalent object manipulation and planning take around seven years to develop, between 3 and 10 years old. Still, some dexterous manipulations may pose a challenge even to humans. However, one can find ways of achieving such manipulation goals through training and exploration, even if the result is only sometimes optimal. Likewise, a robot could train and explore by testing its joints while manipulating objects to balance between prior and newly acquired knowledge. However, robots must already include some capabilities to perform manipulation actions and learn the

missing ones. For example, if the robot knows only how to grasp a cup, it should be able to expand its knowledge and then pour a liquid from that cup into another container.

When comparing human and robot manipulation capabilities, another big difference appears. While humans have a hand with five fingers that include flexible and soft skin full of sensors, many robots use a parallel jaw gripper instead (see Appendix Table A.1 and Table A.2), often without sensors [Billard and Kragic, 2019]. Only this body part difference makes it much harder for robots to accomplish in-hand manipulation tasks in a human-like way.

It is also important to notice that even when hand control is considerable for manipulation, this act involves not only that type of control. Manipulation also requires the usage of the arm, torso and, in many cases, the entire body. Knowing how to use its own body to act reliably is still an open problem in robotics. To overcome these issues, the robot's cognitive capabilities should not just include those mentioned above but also the robot should be able to know and consider its capabilities and limitations while performing a task. This way, the robot can act more reliably and recover from failures.

1.2.1 Manipulation action knowledge acquisition

If we follow the human approach of imitating and testing, learning to plan can take a long time to develop for a robot, more than ten years. However, in the case of a robot, we can provide already processed knowledge. But what would be a good source of this knowledge? One good example of cognitive capabilities is humans, as they have a remarkable ability to store and retrieve an enormous amount of knowledge needed for a given situation. Furthermore, the robot will act in human environments and has to consider it. For example, if I am carrying objects from an open fridge in both hands, I can use my elbow or another part of my arm to push and close the fridge door instead of my hand. As human is already being taken as an example, this work can also use demonstrations from them. These demonstrations need to include enough information so that the robot can obtain trajectories, orientations, locations, velocities and accelerations. However, more than just having this information is needed; the robot also must know that when a hand approaches an object on a cupboard, that hand is reaching that object. That way, when the robot must reach the same or a similar object, it can know the movement features required to achieve that. But, identifying actions from human examples is not a trivial problem; it requires a system capable of differentiating similar action trajectories, such as reaching for an object and retracting a robotic arm.

This work proposes to use action structures from human examples, such as pouring or putting the dishes inside a dishwasher (Figure 1.6), to improve and generate new combinations of actions. Hence, a Personal Service Robot (PSR) performs caregiving tasks. This use of human examples is known as imitation learning or, more specifically, *Programming by Demonstration (PbD)*. It is based on the idea that humans imitate others by observing and approximating their behavior [Brooks et al., 2004], and robots can do similarly. This work uses it from the robot's viewpoint and is called *Learning from Demonstration (LfD)*.

In the specific case of robots, they can extract the information they need from such demonstrations by identifying actions and extracting some of their features, such as grasping contact points between the manipulator and the object. These features give the intuition of how to perform those manipulation actions as commonsense knowledge. These actions require a structure to be included inside the robot's knowledge, which is represented in a hierarchy referred to as a *taxonomy*.

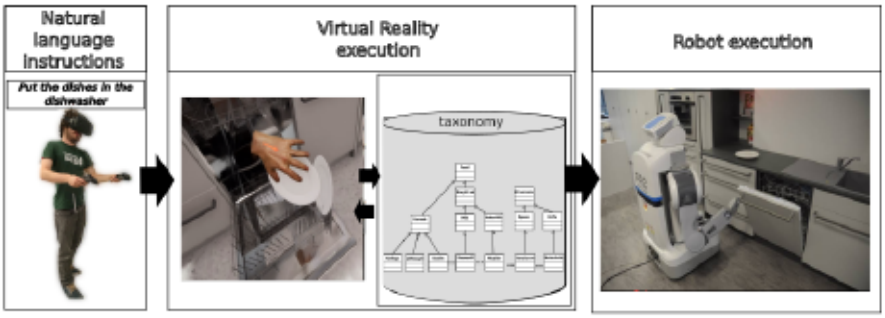


Figure 1.6: Use of examples by a robot to perform a task.

1.2.2 Knowledge-processing memory modules

This work contributes to the cognitive knowledge-processing area of robotic manipulation. In this thesis, the author presents knowledge-processing memory modules integrated into the cognitive architecture CRAM [Beetz et al., 2023]. CRAM is a complete framework for implementing cognitive high-level robot control programs to enable robotic agents to execute their tasks more robustly, reliably and flexibly. However, alongside the reasoning engine of CRAM, PSRs require more knowledge-processing capabilities to integrate more knowledge into their plans. In general terms, robots need advanced cognitive capabilities to predict where, how and why to manipulate objects using prior knowledge. In this sense, generating a well-defined plan structure is essential by using prior knowledge and storing current knowledge for future executions is important. It does not only mean that the robot will call the respective sub-actions at specific points in time, as done in simple scripts. However, it should use more sophisticated reasoning and decision-making systems to determine when and how often to execute them. This includes the preparation for future executions, such as the locations where the robot stands or places objects needed in future actions visible to the robot. For example, not to place objects at locations where they are occluded from others which are required later. More specifically, creating a high-level execution plan means calling the right components in the proper order and finding parameters that maximise the performance of the resulting actions and prevent errors. This also includes prediction mechanisms to integrate the future course of action.

To provide PSRs with the capabilities mentioned before, various memory models extend the cognitive architecture in this work. Cognitive science models, including neuroscience and psychology, inspire these memory models. It is based on the biological idea that cognition is mainly concerned with manipulating and utilising memory [Baxter and Browne, 2011]. This

extension allows the robot to perform tasks in a human-like way by managing knowledge as required, i.e. acquiring, storing and looking for knowledge to act in human environments. With this work, PSRs, with their cognitive architecture, can accomplish vaguely specified tasks such as *bringing me a glass of water* by deciding how to *grasp a bottle*, *open* it and *fill a glass*. They can also choose which type of grip to apply for each specific object, how to reach for it, where to hold it, etc. To accomplish that, robots obtain the missing information from different sources, such as internal (inferring) and external (perception systems) methods, to act according to the combination of all the knowledge. PSRs also produce new knowledge through internal structuring, associations and logical inference mechanisms, similar to the methods presented by Alami et al. [2006]. In this case, memory is not just a repository of sensed data; but includes an active process that transforms factual knowledge into linked structures. As four different types of memory are involved in this work, knowledge should be stored inside the correct type of memory to be managed in a human-like way. Following this idea, this work presents a memory manager based on a model from cognitive science. This model uses the *working memory* as an interface that compares current and previous action executions to provide knowledge about how to act.

As mentioned, PSRs can only serve humans with further expert intervention, especially in handling failures. These errors include the need for more information about executing unknown or known tasks in a human-like way. The memory modules presented in this work handle failures related to unknown tasks by looking for similarities in the structure of actions. Then, the execution plan is built based on the basic actions involved; finally, that plan is simulated and evaluated. In the case of other types of failures, this module uses stored knowledge from previous executions to compare both cases and then decide how to change the plan, if necessary.

One crucial question that has triggered research in cognitive and neuroscience is how humans control movement. It is known that the brain is involved in the production of movement and relies on memory. The brain produces plans to minimize negative consequences (failures). Memory allows people to learn from their surroundings and, more crucially, from previous experiences. So, to optimize cognitive capabilities, memory modules must be constructed appropriately.

The design of such memory modules for a robotic system is detailed in this thesis. These models are based on an accepted division of the human memory system [Tulving, 1985], with *Short Term Memory (STM)* and *Long Term Memory (LTM)* that store the robot's percepts and information gained from previous experience. There are three modules of LTM considered in this work. *Procedural Memory (PM)* stores knowledge needed to perform robot motions. *Semantic Memory (SM)* saves factual knowledge unrelated to a specific time and location. And *Episodic Memory (EM)* preserves individual system experiences. The STM, *Working Memory (WM)*, is likewise intended to have an active storage space containing task-relevant knowledge and holds information representing the current percepts of the system.

Now that the memory modules are defined, it is essential to decide the characteristic of each of them, such that they can process knowledge for a PSR and maximize the potential of the

cognitive architecture that uses them. These modules cannot only recall information that will likely be needed during manipulation but also reduce the execution time.

1.2.3 Knowledge storage

Storage is part of knowledge-processing. The stored knowledge should be rich and deeply represented such that it holds the complete set of beliefs of the robot. In this case, *Knowledge Bases (KBs)* already exist and are used by robots. They exhibit opportunistic behavior to identify unplanned occurrences in the environment and introduce new knowledge directly to the system. However, integrating new knowledge coming from the robot control programs is non-trivial. For example, the detection that results from the perception system needs to be related to object instances inside the *Knowledge Base (KB)*s. Also, using multiple KBs with different representations by the robot is non-trivial. Nevertheless, all of them are working together to accomplish the current goal. This adaptability will be determined by the agents' internal organization and connectivity patterns.

If we still take the human as an example, one issue arises. This issue is that the organization of knowledge in the human brain continues to be under discussion. There are two theories: the object visual property and the connectivity-constraint account. The first ones emphasize the role of non-categorical information in the visual input, such as eccentricity, size, rectilinear, and curvature features (e.g., Levy et al. [2001], Hasson et al. [2002], Konkle and Oliva [2012], Srihasam et al. [2014], Freud et al. [2015], Magri et al. [2019], Yan et al. [2023]). On the contrary, the connectivity-constraint account proposes that the category-specificity in the ventral stream is primarily driven by the innate connections with other brain regions that process nonvisual properties of the corresponding categories (e.g. Riesenhuber [2007], Martin [2009], Mahon and Caramazza [2011], Chen and Rogers [2014], Mahon [2015], Riesenhuber [2020]). Other works suggest that object-directed actions and functional knowledge are represented in different areas of the human brain (e.g. Pobric et al. [2010], Pelgrims et al. [2011], Evans et al. [2016], Andres et al. [2017]). As knowledge related to objects is stored in different areas of the brain, this thesis work implements the four processing memory modules and three KBs; each KB is specific to the types of memory, as shown in Figure 1.7.

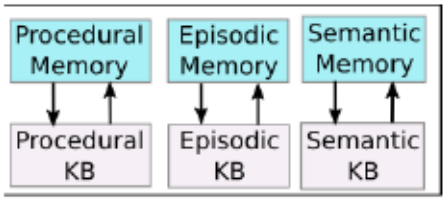


Figure 1.7: Processing memory modules with their Knowledge Bases.

1.2.4 Testing scenario

As manipulations related to caregiving are embedded inside the complexity of preparing and providing a meal, these are the test scenarios used in this work. Pick and place sub-actions were already presented by Flanagan et al. [2006] and are used in this work. That sub-actions list is then extended here in the context of tasks for preparing meals, as these sub-actions were not present in previous work. This extension uses semantic primitives in linguistics, which mean the same in many languages [Wierzbicka, 1992, 2021].

The extended system was tested on five robots with the examples presented in Section 1.2. The robots were also tested on two more tasks: *picking* and *placing* two objects and *bringing* a drink. These examples show capabilities related to serving and preparing meals in human environments. The robots' execution is compared to each other. This comparison includes execution time, number of failures, and number of sub-actions performed (repetition).

1.3 Contributions

In general terms, this work's contributions are mainly in knowledge acquisition and representation, and cognitive robotics. This is done by creating memory-based processing models for Personal Service Robots (PSRs). Specifically, the contributions of this work are:

- The creation and extension of *Knowledge Bases (KBs)* for storing prior, current and conceptual knowledge
  - The creation of a *Procedural Knowledge Base (PKB)* to store action sequences in the form of skills.
  - The creation of an EKB to store the occurrence of events in the form of experience.
  - The extension of a *Semantic Knowledge Base (SKB)* to include more concepts about actions and objects.
- A robust *human activity recognition (HAR)* system to model actions as bottom-up action-sub-action pairs that does not require explicit programming and handles variations.
- The creation of knowledge-processing memory modules extends the CRAM cognitive architecture so that prior knowledge is integrated into plans and current knowledge is stored for future executions.

One first source of knowledge for PSRs is humans when demonstrating how to manipulate objects. For the robot to get knowledge from human demonstrations, a system capable of recognizing actions and representing them in a structured manner is required. In this case, the *human activity recognition (HAR)* system implemented in this work uses machine learning techniques to identify and classify actions into two levels. This hierarchy allows the robot to identify the lowest level as sub-actions. Furthermore, this system also enables the extraction of sub-action features related to object manipulation and body positioning, e.g. distance between the object and body, manipulation type, etc.

Learning from Demonstration (LfD) is strongly linked to the *human activity recognition (HAR)* system as it connects humans' executions (acquisition) and the robot's knowledge-processing system. The data acquisition was made with a 3D *Virtual Reality (VR)* system. The memory modules follow the way humans infer and execute an action. This allows the robot to build execution plans better. Then, those plans are interconnected with the robot's memory to obtain the knowledge required for performing actions.

Memory has a strong connection with cognition and learning. It allows the acquisition, storage, retrieval, use and mix of information, knowledge and experience [Tulving and Szpunar, 2009, Tulving, 2016]. In this thesis, *Working Memory (WM)* denotes a complete implementation that is an interlocutor between the environment and other memory modules. It retrieves knowledge from memory modules based on the current context. This memory model balances flexibility to be specific enough to accomplish retrieval accurately and quickly. It also evaluates heuristic rules such as where to position yourself best to detect an object that is visible or if you should first look for objects at places where you believe they are. In case of failure, it uses prospection to simulate possible actions that can solve the issue. The WM is relatively automatic from the perspective of the systems that manage goals, plans, and task execution. *Episodic Memory (EM)* has a retrieval system for specific experiences, depending on the context. *Procedural Memory (PM)* retrieves knowledge of previously executed actions similar to the learned skills. *Semantic Memory (SM)* retrieves knowledge about concepts, objects and materials. Each memory module has an evaluation method to retrieve the relevant knowledge for the current context.

Part of the knowledge, specifically commonsense knowledge, is used in this work and stored in a low- and high-level hierarchy as *episodes* that include details of performed actions and events. Part of this knowledge is used to extend an *Episodic Knowledge Base (EKB)* with previous experiences. After the performance of actions, a *Procedural Knowledge Base (PKB)* is updated, which includes skills required by the robot to perform tasks. This includes trajectories, used body parts and the effect of action related to the execution goals.

The *Semantic Knowledge Base (SKB)* stores concepts about objects and actions. The PKB and EKB are created and then expanded in this work. On the other hand, the SKB uses KNOWROB [Tenorth and Beetz, 2013, Beetz et al., 2018] as a base and is expanded using interactions with the environment. All this knowledge is then available for the robot.

The memory extension proposed here makes the cognitive architecture transferable to different situations so that various robots can also use it. This is possible since the extracted representations of the observations are given in an abstract form, allowing a better generalization of the demonstrated tasks. This property makes this cognitive architecture superior to classical approaches, where the task is learned for a specific scenario or a particular robot.

Learning capabilities in robotics are linked together with machine learning algorithms. There are two types of learning [Nilsson, 1996, Tapeh and Naser, 2023]. The first one is called *supervised learning*, which uses known values associated with sensor inputs, for example. One use of this would be that if we want to identify the *crack an egg* action, the inputs required are the *contact between the egg and surface* and the *broken shell* to know that the following action is

to *separate the shell*. The output value associated is a *cracked egg* when this information is available. The second type, *unsupervised learning*, has a training set of inputs from which the associated values are unknown. The model tries to classify the inputs meaningfully when the training is performed. Both methods require a training and testing phase. The examples are introduced to the learning implementation during training to produce a useful model.

There is a trend to use an unsupervised learning approach called *deep learning*. In recent years, models have already been pre-trained with large amounts of unlabeled data [Tapeh and Naser, 2023]. This makes it possible to use such models for different applications with less further data for a specified task. However, compared to deep learning, this model prefers to use a combined approach, called *semi-supervised learning*, which uses a small number of labelled examples and many unlabeled ones. This data was collected during this work because the large pre-trained models mostly include text or image data. This approach provides the variability and frequency of specific necessary samples required during training, which include actions and sub-actions. Even more, this memory-based model has an interface capable of collecting knowledge about the hierarchy and features of actions performed by a PSR. Then, when a task is executed successfully, it integrates the knowledge obtained into the permanent memory, called *Long Term Memory (LTM)*. These memory modules and cognitive architecture integration are tested in various simulated robotic platforms presented in Appendix A.1.

1.4 Reader's guide

Inside each chapter, there are some typographical conventions to highlight different elements. One of them is underlined words. This means that the corresponding word definition is included in the glossary at the end of this document in case the reader is interested in this information. The second is *italics* to indicate new terms introduced for the first time. Finally, the reader can find `monospace highlighting` in the following chapters to identify program elements the robot uses. This convention includes program listings and paragraphs to refer to program elements, e.g. variable or function names, databases, data types, environment variables, statements and keywords.

The chapters of this work are divided as follows:

Chapter 2 introduces general notions about robotic systems development. It also presents the foundational state of the art in cognitive systems and the memory modules built and used in this work with their description.

Chapter 3 gives an overview of the implementation presented in this thesis. It describes its components. In the specific case of this work, the module combines *human activity recognition (HAR)* and a memory model for Personal Service Robots (PSRs) to perform complex manipulation tasks.

Chapter 4 presents concepts about knowledge in general and knowledge representation. It also describes the HAR system representation as episodes, including their procedural details and

how they can be used for future executions. Finally, it provides the structure of the sub-action and action plans library and how prior knowledge can be integrated.

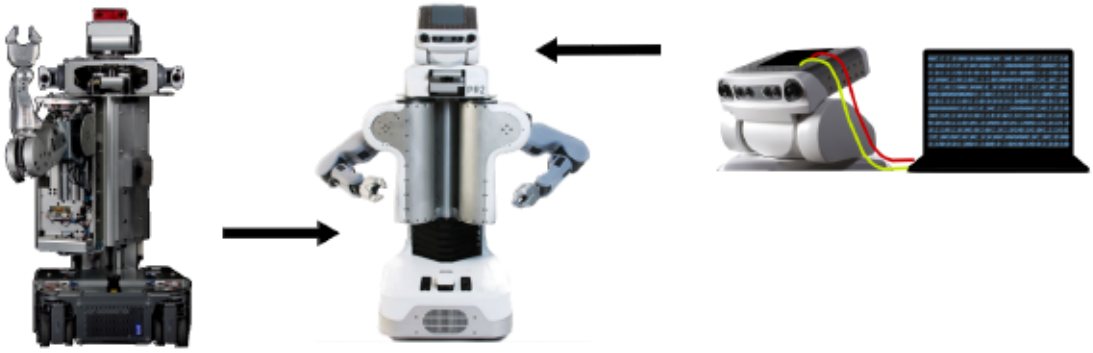
Chapter 5 presents the *human activity recognition (HAR)* state of the art and implementation. The latter uses a combination of machine learning techniques.

Chapter 6 includes the concepts around memory. It also describes the memory management approach used in this work, which consists of implementing a model following the ideas from cognitive science research for memory.

Chapter 7 includes the conclusions and future work.

Chapter 2

Foundations



Robots require specific capabilities to be called Personal Service Robots (PSRs) and assist humans in their daily lives. This work aims to make such a robot capable of performing caregiving tasks in human environments. To achieve that, robots should be capable of gathering the required knowledge to perform tasks in a human-like way. Current robotic platforms use different techniques and are being used in various scenarios, such as nursing homes or households. Therefore, it is of interest for this work to understand how currently available robots represent and perform actions. For this to be possible, it is essential first to introduce a little about the complexity behind building such systems; see Section 2.1. Robotic platforms equipped with the body to perform complex manipulation tasks [Abdo et al., 2016] in households are presented in Figure 2.1. They also require their different components to communicate and interact with each other; see Section 2.2. Some robots can learn; for that, they use machine learning approaches introduced in Section 2.3.

As the robot's control becomes more complex, robots require cognitive capabilities to perform tasks autonomously, which is necessary especially for PSRs if we want to use them to perform caregiving tasks; see Section 2.4. This is why the area of cognitive robotics emerged with the idea of applying models for the makeup of biological brain machines [Christaller, 1999], natural intelligence has only been seen in biological systems, in different living forms and degrees. Therefore, many people have been asking how it might be possible to build an *intelligent* agent that operates in a human-like way. Some approaches are presented in a literature review about cognition in robotics. In addition, cognitive architectures using memory models and cognitive robots as research areas are shown in Section 2.4.3.

2.1 An introduction to Personal Service Robots

A robot has different applications in industry, education and, more recently, households. In the case of this work, the focus is on robots able to manipulate objects in everyday human environments, such as households. These robots are known as service robots and, depending on their use in homes as helpers, collaborators and/or companions alongside other capabilities, can be called more specifically Personal Service Robots (PSRs). Such robots require a body (hardware) with certain features. Also, a formal definition of PSRs is given in Section 2.1.2.

2.1.1 Body design

The design of a robot's body depends on the range of tasks it is expected to solve [Siciliano and Khatib, 2007, p. 67–68]. This includes the robot's mechanical and actuator systems, as well as its geometry, building materials and sensors. An actuator or effector allows robot movement, e.g., a motor inside a joint. As PSRs are required to solve complex tasks, they also need more flexibility, usually more sensors and actuators, which generally increase their price. A robot's design involves engineering, technical, and application-specific considerations regarding task requirements rather than simply a broad specification. The industry is making a noticeable effort to balance flexibility and price, as shown in Appendix A.1. Even though most available robots are still commonly bought by research institutions, the industry is also being directing their production to other customers in recent years.

The robots must also acquire a world model by sensing and interpreting their surroundings. There are various ways to classify sensors, depending on what they measure and how they do it [Siciliano and Khatib, 2007, p. 90]. One class of sensors usually present in robots is *exteroceptive*, which measures information about the external environment, e.g., distance to an object, interaction forces, tissue density, etc. They are classified further into contact or non-contact sensors. *Proprioceptive* sensors measure physical properties in a robot, such as speed, acceleration and joint position. Another way of classification for sensors depends on whether they are passive or active. An *active sensor* emits a signal to the environment and measures the response. On the other hand, a *passive sensor* relies on a signal appearing from the environment. Finally, sensing and estimating refer to the process of transforming a physical sensed measurement into a computer representation for further processing by the perception system. Perception is a higher-level process that allows sensor data requested by the task to be interpreted and integrated to produce a *world model* and then facilitate planning.

2.1.2 Personal Service Robot definition

It is important to note that a Personal Service Robot (PSR) is expected to play two primary roles, according to Chen et al. [2017]. The first is as tools, where they are expected to offer physical

assistance, mainly for the disabled and elderly. Their second role is as partners, where they are expected to interact with people naturally and provide services similar to human caregivers.

The *International Organization for Standardization (ISO)* is responsible for defining functionality metrics. According to the ISO/TC 299 Robotic Technical Committee [Elfving et al., 2012], one classification for PSRs, and interesting for this work, is *robots for domestic tasks*, which includes companions, assistants, humanoid robots, vacuuming, lawn-mowing and floor, pool and window cleaning.

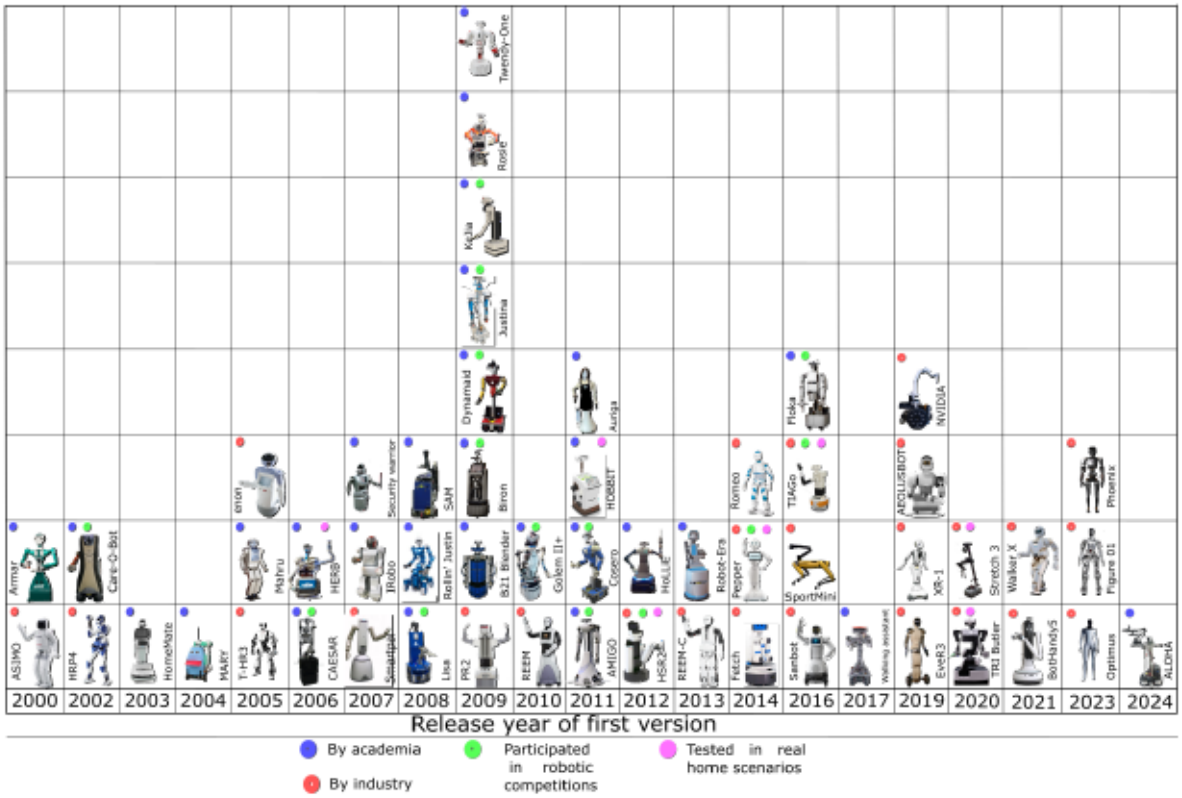


Figure 2.1: Personal service robots' latest version at present and first release year.

As PSRs are planned to be present in humans' everyday life, safety for such robots was first defined by ISO 10218-1:2011. This norm specifies that a robot shall be designed to ensure either a maximum dynamic power of 80W or a maximum static force of 150N, which limits the robot's motor force. This limitation controls the robot's speed, torque and motion, so if there is an impact, it will not hurt or injure a person. For a robot to enter these ranges, its design has to take into account specific factors; for example, if the robot has a 2kg arm and carries an object of 0.5kg while moving at 1m/s, it must decelerate at 60m/s^2 to limit its force below 150N in case of emergency. In other words, knowing that the maximum force of the robot can be $F_{max} = 150\text{N}$, the total mass is the robot's plus the object $2\text{kg} + 0.5\text{kg} = 2.5\text{kg}$ and the force can be calculated with Newton's Law $F = ma$, the deceleration is $a = F_{max}/m = 150/2.5 = 60\text{m/s}^2$.

Requirements for PSRs include more than just features regarding their physical design. Their control systems should allow them to solve everyday chores. Some examples of the chores a PSR should solve are present in the quantitative database stored by the Research Institute of

Human Engineering for Quality Life in Japan (HQL). It includes a set of 44 task types that the robot should be able to perform to provide support on a daily life basis [Iwata and Sugano, 2009]. These tasks include chores such as carrying a bucket filled with water from the floor to a sink, cooking, wiping a window, folding clothes and handling something, etc. Another study in the Program of All-Inclusive Care for the Elderly (PACE) identified 36 high-priority tasks, of which 14 are related to everyday activities that a PSR should be able to perform [Johnson et al., 2017]. For this work, only tasks related to manipulation are taken into account, such as getting a drink, reaching things on high shelves or the floor, finding and bringing items and preparing meals. The robots presented in this chapter should be able to manipulate objects and perform at least one of such tasks; see Figure 2.1. The coverage of this work includes specific actions presented in Section 5.1.3, in which cooking-related tasks mentioned in the database would be solved.

Some interesting ideas are also taken from Kerzel et al. [2017]. Their work mentions that PSRs should have anthropomorphic bodies, mainly to operate in human environments. For this reason, only robots with anthropomorphic features presented as *Degrees of Freedom (DoF)*, e.g.,arms, neck, hip, etc., are considered. Another restriction for this review is regarding the robot's mobility, which excludes robots without a mobile base. Finally, only robots released after 2000 are considered to narrow the search. Robots fulfilling these requirements are presented in Figure 2.1, including colored circles; a blue one marks the robots from industry and a red one the ones from academia. There are also other colored circles. The green circle marks the PSRs that participated in robotic competitions, such as the ones presented in Appendix A. PSRs tested already in real houses deserve recognition as they crossed the bridge between end-users and prototypes; for that, the pink circle is used to mark them. More information on the robots shown in Figure 2.1 is listed in Appendix Table A.1 and Table A.2. In the case of both tables, also command and processing types and middleware are included. The information presented in Figure 2.1 was obtained from an extensive literature survey. Both research institutions and industry have developed robots. Table A.1 and Table A.2 present manipulator features and communication interfaces preferred by robot builders. While industry is moving to use fingers more than grippers as end-effectors of the manipulator, academia seems more looking for functionality to test their algorithms. In the case of this work, the framework proposed is tested in different robotic platforms taken from Table A.1. Those robotic platforms have different configurations, such as one arm, two arms and an end-effector gripper and fingers. These selected platforms use the *Robot Operating System (ROS)* middleware as a communication interface and development platform; see Section 2.2.

Figure 2.1 marks robots participating in robotic competitions. According to my observations, close to the creation of the RoboCup@Home category, there was an increasing number of releases of PSRs. The competition allows the interaction of different robotic developers and, in some cases, collaboration. Even more, if two robots from different teams collaborate while solving the task of one specific test, both teams receive extra points [Matamoros et al., 2019]. However, to my knowledge, no literature conclusively proves that robotic competitions accelerate

the development of such robots. In general, robotic competitions can facilitate testing PSRs while performing specific tasks and potentially trigger collaboration between roboticists.

2.2 Robotic communication software

The robot's control systems inside an architecture must communicate with each other, e.g.,for data and command exchange. The system able to provide such communication is often called middleware. Some examples are the Common Object Request Broker Architecture (CORBA) [Zhen et al., 2009], Real-Time Innovations (RTI) [Castellote and Bolton, 2002], Data Distribution Service (DDS) [Bellavista et al., 2013], Network Data Distribution Service (NDDS) [Pardo-Castellote and Schneider, 1994], Agent Development Environment (ADE) [Mehra and Nissen, 1998], Reconfigurable Context-Sensitive Middleware (RCSM) [Yau et al., 2002], yet another robot platform (Yarp) [Metta et al., 2006], *Robot Operating System (ROS)*, to mention some.

The *Robot Operating System (ROS)* was developed by Willow Garage [Quigley et al., 2009] and is used more frequently in robotics, as shown in Appendix Table A.1 and Table A.2 and this work. Contrary to its name, it is a middleware providing a structured communications layer above a host *Operating System (OS)*. It was designed with the idea of fulfilling service robots' development needs. This middleware supports a large number of software integrations, not just for service robots but for any kind, which increased its usage in academia and industry. ROS is *language-neutral* and *multi-lingual* as it supports and can mix different programming languages such as C++, Python, Octave, LISP and others in various states of completion. ROS already integrates some open-source projects such as drivers, navigation systems, simulators, vision and planning algorithms, among many others. Furthermore, ROS' organization into packages allows researchers to collaborate by sharing their implementations as packages. These are likely to be a few reasons why ROS is used in many robotic platforms today.

2.3 Robotic learning

There are many ways in which machines, such as robots, can learn. These methods are presented in Figure 2.2. A suitable learning method produces hypotheses that predict the classifications of unseen examples [Russell and Norvig, 2003, p. 660]. A prediction is good if it turns out to be accurate; to verify the quality of a hypothesis, the predictions are checked against the correct classification. A set of examples known as the *test set* is required for this verification process. Next, all the collected samples are divided into two disunite sets, the training and test set. Finally, the learning method is applied to the *training set*. When training is finished, hypothesis h is created. Then, we can verify if hypothesis h is correct by measuring the percentage of examples in the test set that is correctly classified by h. The result of this procedure is a trained model that includes a set of data that can be processed to give an average estimated quality as a function of

the size of the training set. This function can be plotted on a graph, showing the *learning curve* for the method on a specific domain that depends on the examples.

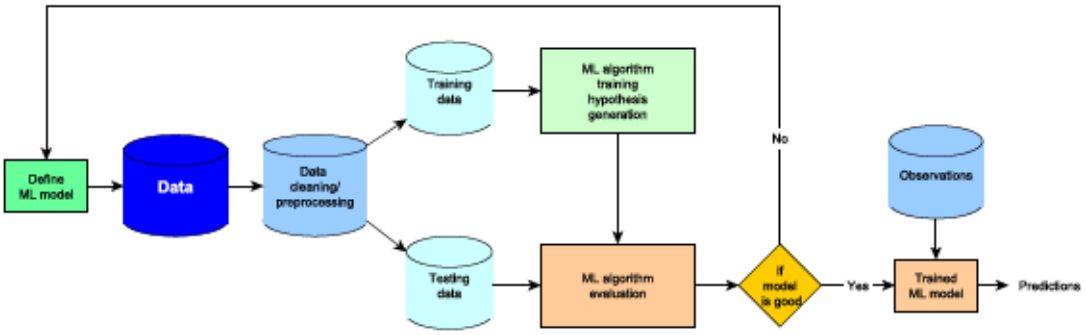


Figure 2.2: Machine learning process.

Machine learning uses computational methods to acquire new knowledge, skills and ways to organize existing knowledge [Tyugu and Tyugu, 2007, p. 80]. The types of these methods are shown in Figure 2.3. Some of them were created in the early days of AI research. These methods were simple systems of *parametric learning* used for building adaptive systems. The role of predefined knowledge and *symbolic learning* was understood later. New approaches to massively *parallel learning* appeared when high-performance computing started and massively parallel hardware was developed.

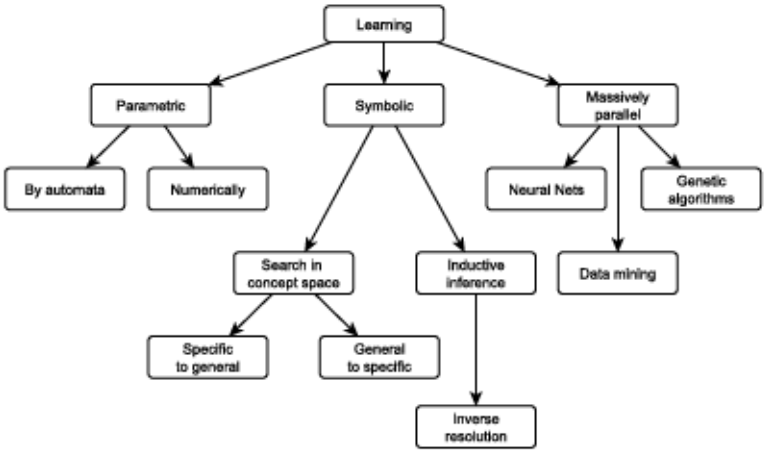


Figure 2.3: Hierarchy of learning methods.

A parametric learning algorithm's behavior can be defined by a simple transition graph with linear branches for external states, known as an *automaton* [Tyugu and Tyugu, 2007, p. 80]. Its output depends on the branch of the graph's current state.

Symbolic learning of concepts is performed by building hypotheses and validating them based on the available knowledge [Tyugu and Tyugu, 2007, p. 84]. Essentially, it is a search in a hypothesis space that includes all of them.

Parallel learning algorithms are suitable for execution on parallel hardware [Tyugu and Tyugu, 2007, p. 98]. The efficiency of a particular parallel algorithm depends on the particular domain

where it will be applied. This type of algorithm aims to develop a hypothesis h that is good in some sense. This means that the hypothesis classifies unknown data with close to 100 % accuracy or has an error score close to 0. An acceptable accuracy or error depends on the domain; e.g., a tumor-classifier model might not be accurate enough.

Selecting which method to apply depends on the nature of the problem and the feedback available from the method [Russell and Norvig, 2003, p. 650]. The field of machine learning usually distinguishes three cases: *supervised*, *unsupervised* and *reinforcement learning*.

Supervised learning involves a function that learns from examples of inputs and outputs. For example, the machine gets a set of images containing people using a mixer and has to build a function to detect the action mixing by the distance between the mixer and the container. The teacher provides the correct output values of the examples. In this case, the environment is fully observable to the machine; this means that it can observe the effects of its actions and can use supervised learning to predict such effects.

On the other hand, if the environment is only partially observable, the problem is easier as the machine might not see the effects of its actions immediately. Then, unsupervised learning can be applied in this case. It involves learning patterns from inputs when no specific output values are supplied. For example, the machine can classify different actions from images depending on the tools used in each of them.

Reinforcement learning is the most general of the three categories. Instead of getting the output values of the input, the machine must learn from reinforcement. For example, if a robot tries to serve a drink without spilling the liquid elsewhere, there is desirable and undesirable behavior. The desirable behavior is to get all the liquid inside the second container, which will be rewarded if achieved. The undesirable behavior is to spill the liquid elsewhere, giving back a punishment or no reward.

In this work, four algorithms of machine learning are used. These algorithms are presented in more detail next.

2.3.1 Clustering

An unsupervised learning method is clustering [Murphy, 2012, Ch. 25]. Its goal is to discover groups of similar examples within some input data; see Figure 2.4. There are some approaches to clustering algorithms. One of them is similarity-based clustering. In this one, the input to the algorithm is an NN dissimilarity matrix or distance matrix D. A *dissimilarity matrix* D measures the distance between objects i and j. This approach allows the inclusion of domain-specific *similarity* or *kernel functions* easily. These functions transform the data into a specific form.

Another approach is feature-based clustering. The input is an ND feature matrix or design matrix X. Its advantage is that it applies to noisy data.

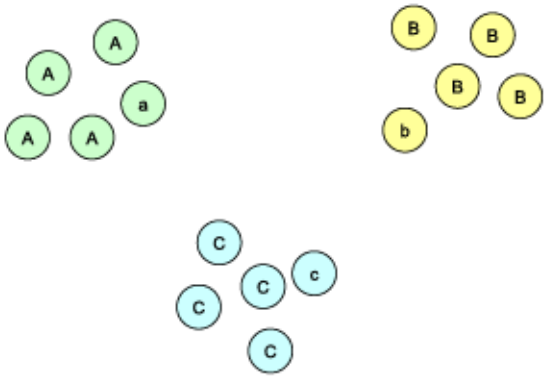


Figure 2.4: General clustering result example.

Another approach used in this thesis work is hierarchical clustering, where the nested tree of partitions is created. This approach is deterministic and does not require the specification of K (number of clusters). In particular, this thesis work uses an agglomerative hierarchical clustering approach. It works in a "bottom-up" manner, as shown in Figure 2.5. This means that each object is initially considered as a single-element cluster (leaf). Then, at each step, two clusters that are the most similar are combined into a new, more significant cluster (nodes). This process iterates over until all points are a member of just one single big cluster (root).

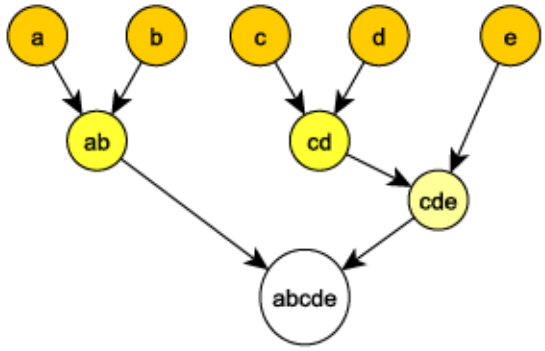


Figure 2.5: Agglomerative hierarchical clustering example.

2.3.2 Decision tree

Decision trees are one of the least complicated yet most successful forms of supervised learning algorithms [Russell and Norvig, 2003, p. 653]. A decision tree returns a decision, which is the expected output value, after receiving as input an object or circumstance characterized by a collection of characteristics. The input and output can be discrete or continuous. A decision tree has two functions: classification and regression. When the output is a discrete value, the tree's function is called classification. On the other hand, learning a continuous function is called regression.

A decision tree has nodes marked by attributes, attribute values or decisions [Tyugu and Tyugu, 2007, p. 116]. An attribute always marks its root. Each path starting from the root passes

through a node with an attribute value and then alternates through the nodes marked by attributes and their values. Finally, it ends with a node labeled by a decision drawn from the values of the attributes met along the path. One graphical representation of a decision tree is given in Figure 2.6.

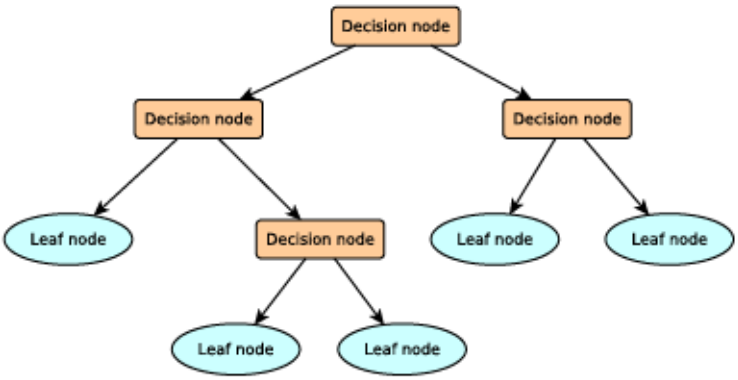


Figure 2.6: Decision tree graphic representation.

Decision trees were developed in the sixties [Russell and Norvig, 2003, p. 674]. The Elementary Perceiver And Memorizer (EPAM) was one of the earliest systems. It was intended as a cognitive simulation model of human concept learning.

The Iterative Dichotomiser 3 (ID3), developed in 1986 by Ross Quinlan, added the idea of using information content to provide a heuristic function [Friedman et al., 2001]. Information content was based on information theory developed by Claude Shannon to aid in studying communication. The algorithm creates a tree with multiple branches and finds for each node, the categorical feature that will yield the most significant information gain for the target categories. The algorithm is based on the observation that a path from the root to a decision is, on average, the shortest if the most discriminating attribute is tested at each step [Tyugu and Tyugu, 2007, p. 117]. The most discriminating attribute can be defined in precise terms as the attribute for which fixing its value changes the entropy of possible decisions at most. The entropy H is calculated to measure the amount of uncertainty in the data S by

$$H(S) = \sum_{x \in X} -p(x) log_2(p(x)) \tag{2.1}$$

where X is a set of classes in S and $p(x)$ is the percentage of elements in class x in the set S. $H(S) = 0$ if the set S is completely classified. The entropy is calculated for each attribute. The smallest $H(S)$ value is used to split the set S on the iteration.

Then, the information gain $IG(A)$ is calculated to measure the difference before and after the set S is divided on an attribute A, defined by

$$IG(S, A) = H(S) - \sum_{t \in T} p(t) H(t) \tag{2.2}$$

where T is the subset created by splitting S by an attribute A, $p(t)$ is the probability of the number of elements in t to the ones in S and $H(t)$ is the entropy of a subset t.

Later, the C4.5 classifier was introduced by Quinlan [Quinlan, 1993]. C4.5 is the successor to ID3. It removed the restriction that features must be categorical by dynamically defining a discrete attribute that partitions the continuous value into a discrete set of intervals. C4.5 converts the trained trees into sets of if-then rules. Then, the accuracy of each rule is evaluated to determine the order in which they should be applied. The set $S = \{s_1, s_2, ...\}$ is the training data of already classified samples. Each sample s_i corresponds to a p-dimensional vector $(x_{1,i}, x_{2,i}, ..., x_{p,i})$, where the x_j represents attribute values or features of the sample and the class in which s_i falls. C4.5 selects the data attribute for each tree node that divides its set of samples into subsets that are enriched in either one specific class or another. Then, the splitting criterion is the normalized information gain (IG described in Equation 2.2). The attribute used to determine the decision is the one with the largest normalized information gain.

The Classification and Regression Trees (CART) is a method introduced by Breiman et al. [1984]. This tree is very similar to C4.5. It is distinguished because it allows numerical target variables (regression) but does not calculate rule sets. Instead, CART builds binary trees by selecting the feature and threshold that yields the most information gain at each node.

Further reading about decision trees and their application in this work can be found in Section 5.2.2.

2.3.3 Reinforcement Learning

As mentioned before, reinforcement learning is based on a machine that receives a reward or punishment depending on its behavior [Russell and Norvig, 2003, p. 763]. Animal psychologists have carefully studied reinforcement for over 60 years. In machines, the idea is that the machine does not know which action to take without some feedback about if that action is good or bad. The machine must know that something is good when receiving positive feedback. This kind of feedback is called a reward or reinforcement. The machine should be able to recognize this reward as such and not as a part of just another sensory input.

Three algorithms can be considered part of this learning type [Russell and Norvig, 2003, p. 764]. A *utility-based* approach is where the machine must have a model of the environment to make decisions, as it must know the states to which its actions will lead. With this approach, the machine learns a utility function on states and uses it to select actions that maximize the expected outcome. The *Q-learning* approach allows a machine to learn an action-value function, or the predicted utility of performing a certain action in a given condition is provided by the Q-function. It can compare the values of its available choices without needing to know their outcomes; because of this, it does not require a model of the environment. This also has a downside, as the machine needs to know where its actions lead; it cannot look ahead. In a *reflex* approach, the machine learns a policy that maps directly from the states to actions. Policies allow the machine to act.

Reinforcement learning can be separated into two types of learning. The first one is *passive reinforcement learning*, for which the task is to learn the utility of states or state-action pairs. The second is *active reinforcement learning*, where the machine has also to learn what to do.

Passive reinforcement learning

In passive learning, the machine's policy π is fixed in its state s so it always executes the action $\pi(s)$ [Russell and Norvig, 2003, p. 765]. The goal is to learn how good the policy is, which means to learn the expected utility function $U^{\pi}(s)$ associated with each state s. The utility is defined as the expected sum of rewards obtained if the policy π is followed as

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s] \tag{2.3}$$

where γ is a discount factor or interest rate, which describes the agent's preference for current rewards over future ones. For example, when γ is near 0, the rewards in the future seem insignificant. Otherwise, when γ is near 1, discounted rewards are equivalent to additive rewards. Discounting is used as it appears to be present in the animal and human decision-making process [Russell and Norvig, 2003, p. 617]. Finally, $R(s_t)$ is a reward function related to the expected agent's reward in its current state. s_t is the agent's state after executing the policy (π) for t steps.

E is the probabilities sum of a current state s' given an executed action and is calculated by

$$E = \sum_{s'} P(s' | s, \pi(s)) \tag{2.4}$$

Active reinforcement learning

A machine using active reinforcement learning must decide what actions to take [Russell and Norvig, 2003, p. 771]. First, the machine will need to learn a complete model with outcome probabilities for all actions, rather than just the model for the fixed policy. It also needs to consider that it has a choice of actions. Then, the utilities it needs to learn are those defined by the optimal policy. They obey the Bellman equations as follows

$$U(s) = R(s) + \gamma \max \sum_{s'} T(s, a, s') U(s') \tag{2.5}$$

where $R(s)$ is the reward function on state s, then an addition is made of the γ is a discount factor times the maximum possible resulting state s'. This is obtained by calculating the transition T from a state s to a resulting state s' when applying an action a, all multiplied by the utility of that resulting state $U(s')$.

To solve this equation, it is required to apply the Bellman update equation infinitely often to reach an equilibrium described by

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max \sum_{s'} T(s, a, s') U_i(s') \tag{2.6}$$

Then, the final utility values must be solutions to the Bellman equations and corresponding optimal policy. This process is performed in the algorithm called Value-Iteration.

If there is no policy π available, the machine can use the Q-learning method. This way, the machine can learn an action-value representation instead of learning utilities [Russell and Norvig, 2003, p. 775]. This approach has the benefit that the machine does not need a model for learning or action selection. In this case, the utility values can be obtained by

$$U(s) = \max_a Q(a, s) \tag{2.7}$$

where $Q(a, s)$ represents the value of acting (a) in the state s.

The Q-values can be calculated using the equation

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s') \tag{2.8}$$

The updated model for Q-values calculates when an action a is executed in a state s that leads to a state s', described by

$$Q(a, s) \leftarrow Q(a, s) + \alpha(R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s)) \tag{2.9}$$

where α is the learning rate.

This approach's drawback is that it learns more slowly than the previous one.

2.3.4 Artificial Neural Networks

An *Artificial Neural Network (ANN)* provides another form of massively parallel learning and comprises simple units called *neurons* capable of generalization [Bishop, 2006, Tyugu and Tyugu, 2007, Tapeh and Naser, 2023]. *Generalization* refers to the network's production of reasonable outputs for the inputs not encountered during training (learning). ANNs can find a good approximate solution to complex (large-scale) intractable problems.

They work well to solve pattern recognition problems and can be implemented in either the hardware or software. They are commonly represented as a graph; see Figure 2.7. Each graph node has an associated state x and threshold t. Each edge has an associated numeric value called weight w. The network learns from the examples by building a mapping between inputs and outputs for the problem. The goal of such a system is classification, which takes an input vector

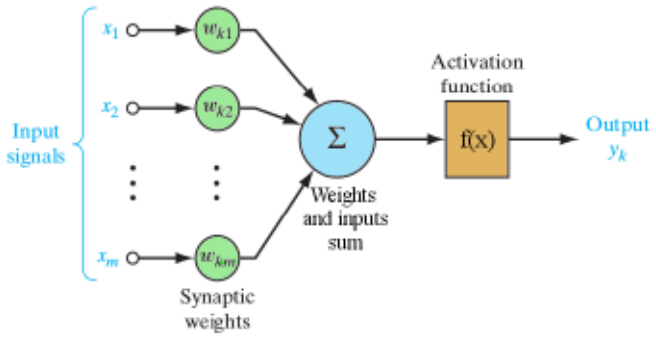


Figure 2.7: Neuron process representation.

x and assigns it to one of K discrete classes C_k where $k = 1, ..., K$. Classifications or learning in different types of ANN usually are performed using *backpropagation* [LeCun et al., 1998]. By adjusting the network's weights (how much a certain input contributes to the outcome) and biases (which characteristics are chosen), backpropagation can identify the circumstances in which mistakes are eliminated from networks designed to imitate human neurons [Orr and Müller, 2003, p. 10]. In this case, the states taken by output nodes are evaluated by each output node. The evaluations are propagated back to other layers. The weights of edges that supported the right decisions are increased, and the weights that kept wrong decisions are decreased. This method applies to ANNs with a small number of layers, 2 or 3.

The objective is to keep adjusting the weights and biases until the intended and actual outputs are identical. The artificial neuron then fires and transmits its resolution to the following neuron in line. One neuron's contribution to the answer is merely a portion of it. Up until the group of neurons produces a final output, each neuron transmits information to the one behind it in the chain. This is called the *gradient-based learning* method. The classes are taken to be disjoint, so each input is assigned to only one class in the most common scenario. For that to happen, ANNs include a *cost* and *activation function*, *weight*, *neurons* and *layers*. All of them are introduced next.

Neuron

An artificial *neuron* approximates the neuron found in the biological brain. They can be either physical devices or mathematical constructions. The development of artificial neurons started with the idea of reverse-engineering how the biological brain processes signals. Its connections and components are based on biological analogies by using brain terms such as neurons and axons as names. However, in practice, they are nothing more than a sophisticated *linear regression*, which is a particular form of function approximation to model a given set of random variables. Still, these algorithms are effective against complex problems and quick for predicting.

The *perceptron* is the most straightforward representation of a neuron and can be seen as a single-layer ANN. It is an iterative algorithm that tries to determine the best values for the weight vector w by successive and reiterative approximations. The importance of this vector is

that it can help predict the class of an example when it is multiplied by the matrix of features x and added to a constant term, called the bias (b). The bias can increase the classification accuracy as it does not depend on the input. The natural specialty of the perceptron is binary classification. The output of the perceptron is obtained by

$$out_i = y_k = \sum_{k=1}^{m}(x_k w_k + b) \tag{2.10}$$

where y_k is the predicted value, w is the weight, m is the number of inputs, k represents the classes, and b is the bias.

The total error of the perceptron is obtained by

$$Error = E^t = -\sum_{i}^{m} y_i(x_i^\top w_i + b) \tag{2.11}$$

where y_i is the expected classification value.

In other types of ANNs, and as mentioned before, the bias b can be a scalar, variable, diagonal matrix or an estimate of the inverse second derivate of the cost function.

Neurons in an ANN are a further evolution of the perceptron as they can take many weighted values as inputs, sum them, and provide the summation as a result, just as a perceptron does. However, they also offer a more sophisticated summation transformation, something that the perceptron can not do. A graph model of this can be seen in Figure 2.7.

The neuron model was developed by observing nature. This way, scientists noticed that neurons receive signals but do not always release a signal of their own; this depends on the amount of signal [Haykin, 2010, p. 17]. When a neuron acquires enough stimuli, it fires an answer. Similarly, artificial neurons give a result after receiving weighted values, sum them and use an activation function to evaluate the result. This is the reason they are called a non-linear method. For example, the activation function can release only zero values until the input reaches a certain threshold.

Interconnected neurons make a network with each neuron's inputs and outputs connected to other neurons. In some cases, they are interconnected by layers.

Weight

The *weight* is a free parameter normally modified to lessen the difference between the desired and actual response of the network generated by the input signal [Bishop, 2006, p. 2]. During training, the weight is repeatedly multiplied by the input examples in the network set. This stops when the network reaches a state where no further significant changes in the weights appear. To

update the weight, it takes a random misclassified input x_t and output y_t values and adds it to the total value by

$$w' = w + \eta(x_t * y_t) \tag{2.12}$$

where η is the learning rate, in the case of the perceptron, which value goes from 0 to 1. The form of the learning rate η needs to be selected carefully. It can be a scalar constant, variable, diagonal matrix or an estimate of the inverse second derivate of the cost function; this depends on the method selected.

Activation function

The activation function is a transformation applied after the weight to each value of the input vector x and then adding each result to obtain an output. The transfer or activation function defines the output of each neuron in terms of the sum of weights w and inputs x product over all incoming edges of the node k defined by

$$x_k = f(\sum_{m}^{i} w_{km} * x_m - t_m) \tag{2.13}$$

where w_{mk} is the weight for the edges, x_m is the current state from the previous layer and t_m is the threshold of the node m. Other common activation functions are sigmoid and hyperbolic tangent.

It can be of type sigmoid, hyperbolic tangent (tanh), Rectified Linear Unit (ReLU), etc. ANNs use the sigmoid or hyperbolic tan functions often.

The sigmoid function has an s-shape [Orr and Müller, 2003, p. 14]. It is described as a strictly rising function with a smooth transition between linear and non-linear behaviour. The logistic function described by is an example of a sigmoid function is presented by

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{2.14}$$

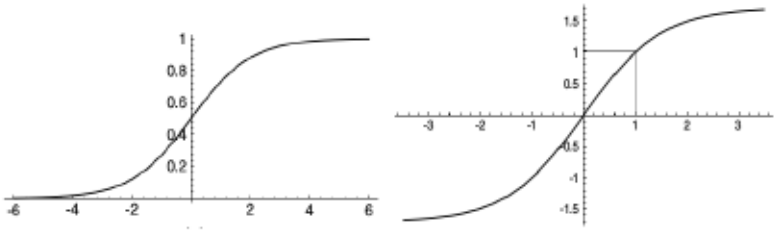
which is an increasing function that asymptotes at some finite value as $\pm\infty$ is approached.

The hyperbolic tangent function also has an s shape and is defined by

$$f(x) = tanh(x) \tag{2.15}$$

To generate an output, the activation function computes the neuron's induced local field. For instance, the activation function can yield zero values unless the input achieves a specific threshold or enhances value by rescaling it closer to the threshold.

The difference between these two activation functions can be seen in Figure 2.8, where the values the functions can take differ. For the sigmoid function the values go between $0 < f(x) < l$ and for the hyperbolic, they go between $-l < f(x) < l$.



(a) Sigmoid (b) Hyperbolic

Figure 2.8: Activation functions graphs.

Cost functions

The *cost function* or loss function, is the predicted error. It measures the loss incurred in taking any available decision or action [Bishop, 2006, p. 41]. In other words, it measures the discrepancy between the predicted value of y_k and the true value of y_t. The goal of the training is to minimize the average loss of the model. Two error types can arise [Bishop, 2006, p. 180]. First, a false positive (aka false alarm) occurs when we estimate a positive value $y = 1$, but the truth is negative $y = 0$. On the other hand, a false negative (aka missed detection) happens when we estimate a negative value $y = 0$, but the truth is positive $y = 1$.

The *Mean Squared Error* is the most widely used cost function [Orr and Müller, 2003, p. 10]. It is used to choose the parameter values so that the hypothesis $h(x)$ is close to the true value y_t for the training examples, where m represents the number of training examples.

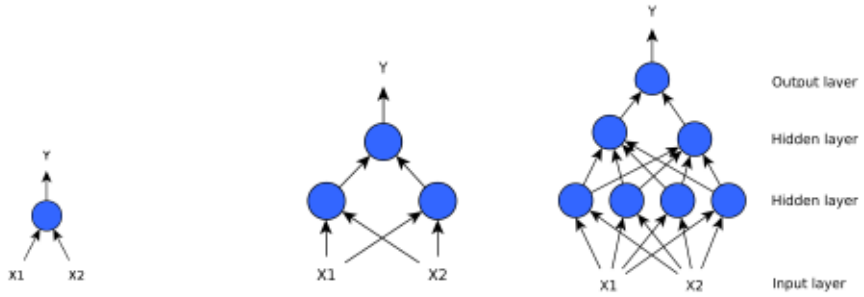
$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_t - y_k)^2 = \frac{1}{m}\sum_{i=1}^{m}(h(x_i) - y_k)^2 \tag{2.16}$$

Layer

An *Artificial Neural Network (ANN)* is a hierarchical model including *neurons* and *layers* to approximate parts of the biological brain. ANN perform operations, also known as layers, on the input data depending on connections, weights and parameters, as well as neurons [Vidal et al., 2017].

A layered neural network organizes its neurons in the form of layers [Haykin, 2010, p. 17]. In its most basic form, a layered network contains an input layer of nodes that is connected directly onto an output layer of neurons (computation nodes), but not vice versa; see Figure 2.9a. In the

case of multilayer networks, the intermediate layers are called *hidden layers*; see Figure 2.9c. The focus of attention of Figure 2.9 is restricted to signal flow from neuron to neuron. It uses a reduced form of representation by omitting the details of the signal flow inside the individual neurons. The input nodes provide the input signals; each neuron is represented by a single node and the links interconnecting the nodes do not show weight. This representation only provides directions for signal flow. The input layer is not counted because there is no processing inside it.



(a) Single layer (b) Two layers (c) Three layers

Figure 2.9: Different number of layers with two inputs X1 and X2 and one output Y.

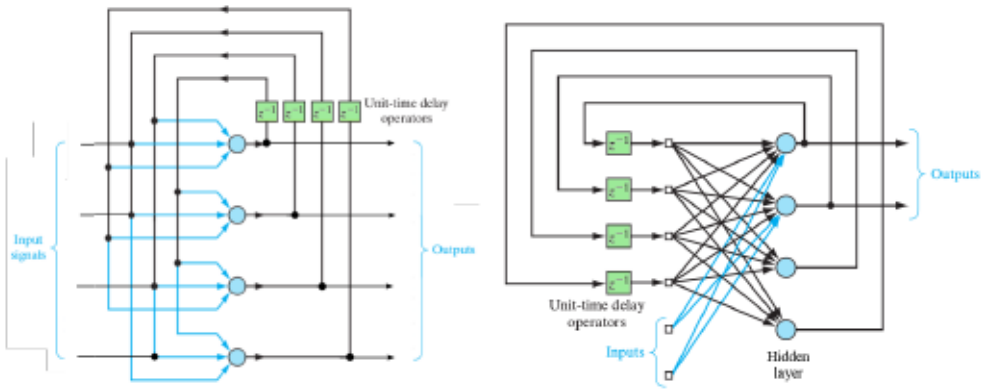
ANNs can be classified based on their general structure into *Forward-pass Neural Networks* and *Layered Neural Networks* [Tyugu and Tyugu, 2007, p. 101]. The first type, *Forward-pass Neural Networks*, can be represented as acyclic graphs. Their nodes can be classified as input, output and internal nodes. It is important to note that input nodes do not have incoming edges, output nodes do not have outgoing edges and internal nodes possess both kinds of incident edges; see Figure 2.7. The second type, *Layered Neural Networks* can be represented as layers, where nodes can be divided into n layers so that each layer contains only nodes of one type. Each node in such a graph belongs exactly to one layer, where they are strongly connected. Both can be combined to create a *Forward-pass Layered Neural Network*, in which the states of output nodes can be interpreted as decisions based on the states of the input nodes.

There are different types of layered networks. For example, the networks presented in Figure 2.9 are of the Feedforward type. On the other hand, the first type of layered neural network is a Single-Layer Feedforward Network [Haykin, 2010, p. 21]. In this case, an input layer of source nodes projects directly onto an output layer of neurons.

The second type is a Multilayer Feedforward Network [Haykin, 2010, p. 22]. In this case, the hidden layers appear. They are computation nodes corresponding to hidden neurons or hidden units, which are called that way because they are not seen directly from either the input or output of the network. Their function is to intervene between the input and output. If one or more hidden layers are added, the network can extract more higher-order statistics from its input.

The third type is the Recurrent Network [Haykin, 2010, p. 23]. A *Recurrent Neural Network (RNN)* is not only of type feedforward as the previous ones. One example is a *Recurrent Neural Network (RNN)* with a single layer of neurons from which each neuron gives its output signal back to all the other neurons' inputs; see Figure 2.10a. In this case, there are no self-feedback loops in the network. This is a circumstance in which a neuron's output gets recycled back

into its input. Another model of a RNN uses hidden neurons; see Figure 2.10b. In this case, the feedback connections originate from the hidden and output neurons. The feedback loops involve using particular branches composed of unit-time delay elements $z - 1$, which results in a non-linear dynamic behavior.



(a) No hidden layer (b) One hidden layer

Figure 2.10: Recurrent Neural Network with one feedforward.

They can maintain a hidden state which keeps track of previous elements in the sequence described by

$$H_t = f(hidden_{t-1}, x_t) \tag{2.17}$$

The steps followed by RNNs are presented in Figure 2.11. In the first step, a hidden state h_t is usually a matrix of zeros so that it can be fed into the RNN together with the first input value of x_t. The hidden state and input data will be multiplied then with weight matrices W_in. The result of these multiplications will then be passed through an activation function. Then the result is fed back into the RNN with the following input value of x_t. This process continues until the model stops producing outputs.

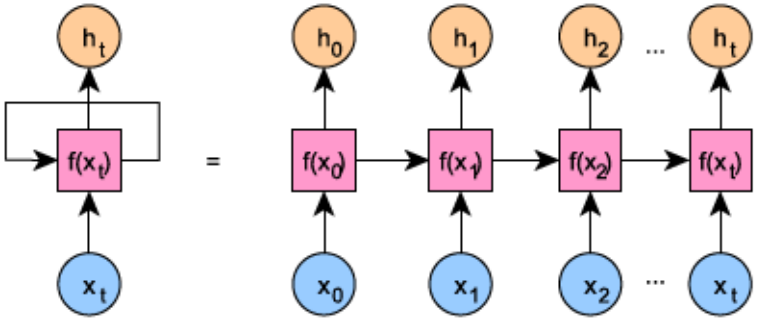


Figure 2.11: Recurrent Neural Network process.

RNNs can use an approach known as *Long Short Term Memory (LSTM)* units to modify the architecture and then work with longer sequences of inputs. For that to happen, the LSTM takes in 3 different pieces of information, including the current input data, hidden states and *Long*

Term Memory (LTM), which includes more information. It requires input and forget gates to control whether information is saved or deleted at each time step before it is sent to memory. The process is presented in Figure 2.12.

The input gate i manages the information to be passed to the LTM by using two layers, i_1 and i_2. The layer has a sigmoid function with a weight W_{i1} and bias b_{i1} described by

$$i_1 = \sigma(W_{i1} \cdot (h_{t-1}, x_t) + b_{i1}) \tag{2.18}$$

The second layer takes the *Short Term Memory (STM)* and current input x_t and passes it through an activation function to regulate the network. The STM is also known as the *hidden state*, which will be used from now on. This function is normally hyperbolic; in this case, it is described by

$$i_2 = tanh(W_{i2} \cdot (h_{t-1}, x_t) + b_{i1}) \tag{2.19}$$

The outputs from these layers are then multiplied by $i_{input} = i_1 * i_2$, and the result represents the information inside the LTM and output. Later, to obtain the output, the RNN takes the current input x_t, the previously hidden state h_{t-1} and LTM to produce the new hidden state (STM). For that to happen, a sigmoid function with different weights will pass the previous hidden state and current input again.

$$O_1 = \sigma(W_{output1} \cdot (h_{t-1}, x_t) + b_{output1}) \tag{2.20}$$

Then, the new LTM passes through an activation function creating O_2. The result of both of them is multiplied to obtain the hidden state and output values h_t, $O_t = O_1 * O_2$.

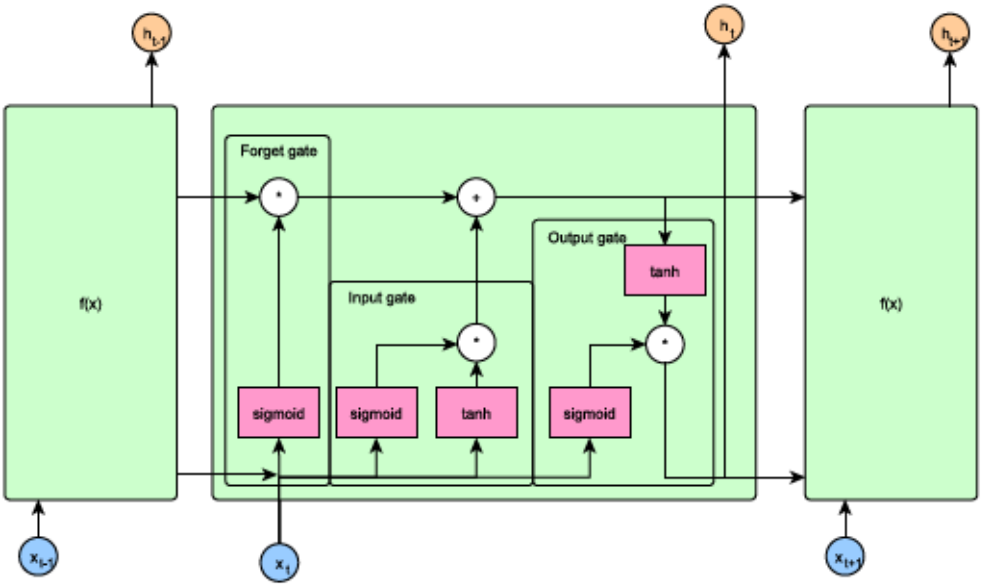


Figure 2.12: Recurrent Neural Network with LSTM process.

Further reading about RNNs and their application in this work can be found in Chapter 5, specifically in Section 5.1.3.

2.4 Robot's action execution control

Robotic action execution requires a sequence of collision-free movements from a start to a goal position across mobile or static obstacles, known as a *plan* and still an open problem [Siciliano and Khatib, 2007, p. 109]. A plan can also be seen as a set of actions with conditions or constraints to pass from one state to another by collecting facts from a KB. In humans, planning is related to thinking of a combination of actions required to achieve a goal or to obtain the desired results [Das et al., 1996]. In this work, a *plan* is seen as in the case of humans. Various actions must be performed in a sequence to complete a task successfully. This sequence of actions constitutes a *plan*.

Robots are required to build and execute plans to accomplish a task. They can achieve that by precisely controlling their body parts, which is accomplished by software commanding the robot's hardware. As expected, robot software systems tend to be complex, normally called architectures, structure design, or frameworks. This complexity is due to the need to control various sensors and actuators (hardware) in real-time in the face of significant uncertainty and noise from the environment [Siciliano and Khatib, 2007, p. 187–188]. Robots must achieve tasks while monitoring for and reacting to unexpected situations. To achieve it, robots require structured software for execution, monitoring and control, an architecture. Different architectures exist, each has advantages and disadvantages. As may be expected, they have changed over the years. For this reason, a bit of history is required to understand how their evolution happened and which features are important while choosing which to use or build.

2.4.1 Robotic architectures

First, it is helpful to know that the first robotic architecture and programming began in the late 1960s with the Shakey robot at Stanford University [Siciliano and Khatib, 2007, p. 89]. Shakey's architecture had three functional elements: sensing, planning, and executing. This approach was called the Sense-Plan-Act (SPA) paradigm. As the plan was built after sensing information was available, the reaction of the robot was slower than it was. In the early 1980s, the reactive planning architecture [Nilsson, 1984] emerged, in which plans were generated quickly and relied more directly on sensed information instead of internal models as Shakey did. The most influential one was Brooks' subsumption architecture [Brooks, 1987], the first one built from layers of interacting finite states. Since then, many other architectures have been created based on the same idea of using layers, usually two or three, to separate processes. This makes the architectures modular and compact, where functionalities are structured and separated. Layers can be represented as a vertical or horizontal stack of sheets with different components; see Figure 2.13. As this model is generalized, elements between layers can repeat. Normally, the

bottom layer receives the information from the environment through the robot's sensors. The high layer usually performs high-level processing.

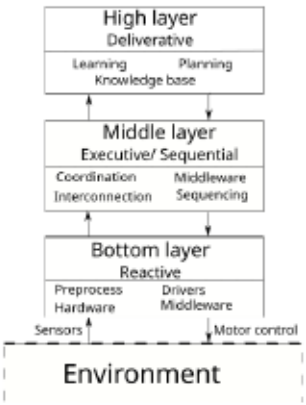


Figure 2.13: Representation of layers in a vertical fashion with different components each layer can include.

2.4.2 Cognitive architectures

In recent years, the idea appeared that to be able to solve more complex tasks, it is necessary to provide a robot with cognitive capabilities. Some research is going in this direction by testing neuroscience models of human cognition in robots; this type of robot is known as a cognitive robot. A cognitive robot does not just increase the capabilities of robots but also helps to understand how such processes work in the human brain.

There is no unified theory or definition of cognition, and it still needs to be better understood how it works. There are some definitions related to human cognition. One example is the work by Postman [1951], which talks about a self-sufficient process providing adaptable behavior in a changing environment. On the other hand, Franklin and Graesser [1996] mention that cognition includes *Short Term Memory (STM)* and *Long Term Memory (LTM)* in the processes of categorizing and building concepts, reasoning, planning, solving problems, learning, and creativity. In the work by Pecher and Zwaan [2005], cognition is considered to be grounded inside the human body, which interacts with the environment to represent and understand the world. Perception and action are considered central in that interaction with the environment.

One definition related to robots is presented in the work by Vernon [2014]. His work defines cognition as a process by which an autonomous system, biological or artificial, is capable of perceiving its environment, learning from its own experience, anticipating the consequences of its actions, acting to achieve goals and adapting to changing circumstances. So, for a robot to be called cognitive, it should be able to adapt, understand, anticipate and act in a changing environment. This definition is the one used in this work.

According to Barsalou et al. [2007], there is a dependency between all the basic processes in the brain, including perception, action, reward and learning. Human cognition emerges from deep dependencies between basic systems and evolves socially. Similarly, Ramírez Amaro

[2014] mentions that when learning a new task, the main source of information for agents is the observation from different sources. That is why it depends on the application and social interactions to develop. However, when humans are in isolation, they can adapt to new situations with the re-use of learned models to infer how to perform an unknown action. This is only possible by understanding what was previously observed by agents while performing a certain action and then extracting semantic information from such observations. Agents, humans or robots, are goal-directed and according to such goals, they manage priorities based also on motivation and opportunity. Then they can perceive, categorize and make inferences from the obtained information, but they also perform memory retrieval. In this sense, they do not learn how to perform an individual task, but how to coordinate tasks. A cognitive architecture is the control system used normally by a cognitive robot to coordinate such tasks.

Since the 1950s, cognitive architectures have been enormously developed, even if not often tested in robots. These approaches model the human mind and build systems capable of reasoning, adapting to new situations and reflecting on their actions [Kotseruba and Tsotsos, 2018]. The work presented by Sun [2004] defines a cognitive architecture as a framework that models cognition through a structure that includes modules division and their relationships. Cognitive architectures are designed for a specific purpose. In this work, the extension of the cognitive architecture has the mission to develop an adaptive Personal Service Robot (PSR).

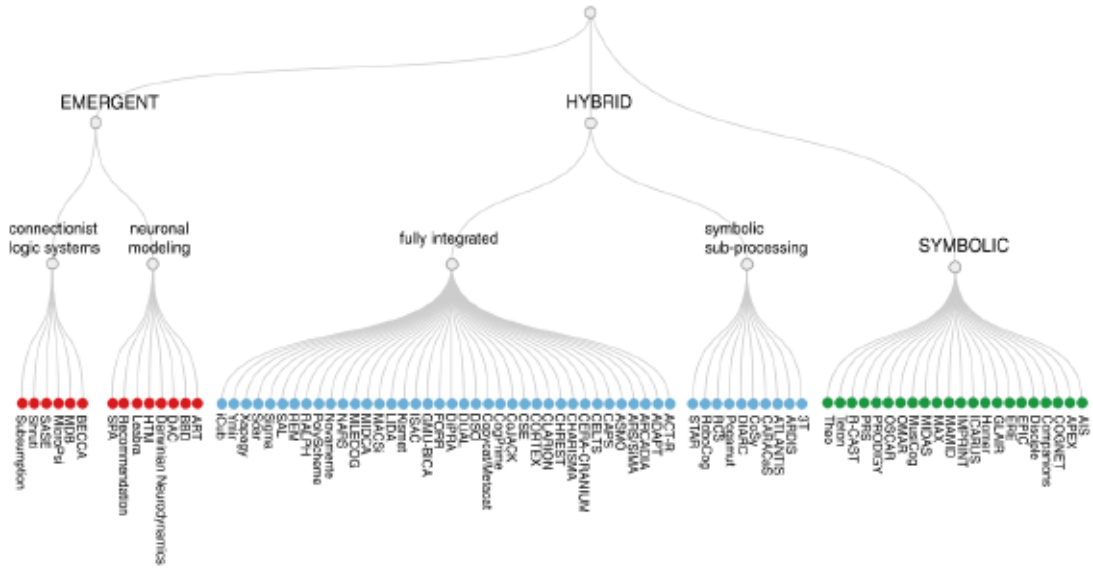


Figure 2.14: Cognitive architectures taxonomy based on representation and processing. Reprinted from "40 years of cognitive architectures: core cognitive abilities and practical applications" by Kotseruba, Iuliia and Tsotsos, John K. 2018, Artificial Intelligence Review, p. 9. Copyright 2018 by Springer.

The work of Kotseruba and Tsotsos [2018] presents 84 architectures classified as symbolic, hybrid and emergent depending on their way of representing knowledge and processing methods; see Figure 2.14. *Symbolic* approaches represent concepts that can be modified using predefined instructions, in many cases if-then rules. That makes them fast to implement. One disadvantage is for these approaches is that the creation of an initial KB requires a long time. On the other hand, many of the *emergent* approaches are biologically inspired. That makes them ideal for

understanding biological systems. One disadvantage is that they require to be trained to produce useful behavior. *Hybrid* architectures integrate world knowledge, such as concepts or sensing information, with reactive actions, normally applied to safety reactions like "move when you are too close to an obstacle while navigating" [Arkin and MacKenzie, 1994]. They emerged as a need to include symbolic knowledge, such as representations of the environment, in the construction of robot behaviors, which means they use prior knowledge for plan formulation. The review presented by Kotseruba and Tsotsos [2018] noted that there was a particular interest in symbolic architectures until the early 1990s. However, after the 2000s most developments went in the direction of hybrid architectures. In the same way, this work uses a hybrid approach for building its framework.

On the robotics side, one of the first steps towards integrating more reactivity and deliberation was the reactive action packages (RAPs) created by Firby in his thesis [Firby, 1990], which included three layers. Independently of Firby's, Bonasso et al. [1996] developed another three-layered hybrid architecture that included robot behaviors in the bottom layer. This layer guaranteed consistent semantics between the agent's internal states and its environment. It was called 3T (three-tiered) and was capable of three control processes: planning, sequencing and real-time control. This architecture has been used on many generations of robots.

Another example of a hybrid approach is the Distributed Integrated Affect Reflection and Cognition (DIARC), which has been under development for more than 15 years [Scheutz et al., 2019]. Compared to other cognitive architectures like *State, Operator And Result (SOAR)* [Laird and Mohan, 2014] or ACT-R [Lebiere et al., 2013], DIARC is a distributed architecture based on a component scheme that allows it to be instantiated in many ways. DIARC is similar to the CogAff [Sloman, 2002] architecture. Both can be used for different cognitive systems, including robots. DIARC has been tested in speaking robots during *Human-Robot Interaction (HRI)*, but only a little in manipulation tasks.

2.4.3 The use of memory and learning in cognitive architectures

This section presents a comparison of hybrid cognitive architectures based on their memory and content, as well as their learning, perception and motor capabilities. Memory can be seen as a cognitive process to encode, store, retrieve and mix information. It is essential for learning [Tulving and Szpunar, 2009]. More details about memory are discussed in Chapter 6.

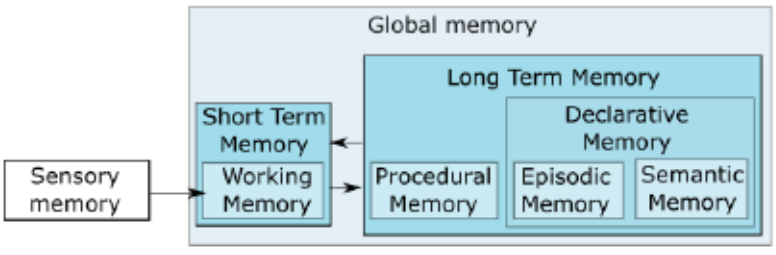


Figure 2.15: Representation of memory types used in hybrid cognitive architectures.

The types of memory considered are sensory, working, semantic, procedural, episodic and global; see Figure 2.16. Kotseruba and Tsotsos [2018] presents the definitions of these types of memories. *Sensory or perceptual memory* catches inputs from the sensors and pre-processes them before sending them to the other memories. *Working Memory (WM)* stores information about the current task temporarily; it is for this reason that it is considered a *Short Term Memory (STM)*. *Procedural Memory (PM)* stores implicit knowledge which is related to motor skills, routine behavior, etc. and is part of the *Long Term Memory (LTM)*. *Episodic Memory (EM)* includes knowledge about experience and autobiography. It is considered to be part of the Declarative Memory, which is part of the LTM. *Semantic Memory (SM)* stores concepts and facts. It is also considered to be part of the Declarative Memory and LTM. Global memory does not separate memories into short- and long-term, instead, it stores all the information together. These memory types are represented in Figure 2.15 with relations and interconnections found in the literature.

Different cognitive architectures use a combination of memories depending on their model of cognition and application. Figure 2.16 shows which specific memory types are used in which cognitive architecture. From them, 20 hybrid cognitive architectures use the same types of memory as this work, e.g.,*Working Memory (WM)*, *Semantic Memory (SM)*, *Episodic Memory (EM)* and *Procedural Memory (PM)*. They differ in the way they use their EM and WM as shown in Table 2.1.

In the case of *Working Memory (WM)*, some cognitive architectures use it as an *activation mechanism* or filter to decide if knowledge should enter to the other memories, similarly to the approach presented in this work, marked with an asterisk in Table 2.1. As WM has a limited capacity, it can update fast and keep a flow of the current information. It connects to other memory types to store information in the long-term and is capable of decision-making. The second usage of WM in cognitive architectures is a processing system for the *world state* built by the sensed input [Kotseruba and Tsotsos, 2018]. This allows the architectures to perform actions depending on the state of the world in a limited time interval. Some other cognitive architectures applied to robots use the WM for a *spatiotemporal integration* to interact with the environment. This allows the robots to have a sense of space and form when interacting with objects, which makes it possible to detect successive changes in a broader time interval. This model is similar to the previous application. The difference is that this model allows robots to reason about the effects of the actions they perform in the environment. Some psychology areas believe that information is added to the WM via a condition-pair pattern matching process, such as if-then rules. These conditions specify what to do in each state [Anderson, 2013].

In the case of *Episodic Memory (EM)*, cognitive architectures do not use it as much as the WM [Kotseruba and Tsotsos, 2018]; see Figure 2.16. The first type of usage for EM is as a *coordinator*. This model uses this type of memory to extract or learn new semantic or procedural knowledge, which saves and uses action experiences for future behavior, similar to this work. Other architectures using EM take advantage of storing previous experiences to improve future executions without sharing such knowledge with other memory types.

Cognitive architecture	Sensory	Working	Semantic	Procedural	Episodic	Global
ACT-R	●	●	●	●	●	○
CHARISMA	●	●	●	●	●	○
CLARION	●	●	●	●	●	○
LIDA	●	●	●	●	●	○
Sigma	●	●	●	●	●	○
Soar	●	●	●	●	●	○
Pogamut	●	●	○	●	●	○
ARCADIA	●	●	○	○	○	○
ASMO	○	●	●	●	●	●
CogPrime	○	●	●	●	●	●
NARS	○	●	●	●	●	●
CELTS	○	●	●	●	●	○
Copycat/Metacat	○	●	●	●	●	○
DUAL	○	●	●	●	●	○
GMU-BICA	○	●	●	●	●	○
ISAC	○	●	●	●	●	○
iCub	○	●	●	●	●	○
MIDCA	○	●	●	●	●	○
MLECOG	○	●	●	●	●	○
Novamente	○	●	●	●	●	○
REM	○	●	●	●	●	○
Ymir	○	●	●	●	●	○
CORTEX	○	●	●	●	○	●
DiPRA	○	●	●	●	○	●
RALPH	○	●	●	●	○	●
RoboCog	○	●	●	●	○	●
3T	○	●	●	●	○	○
ADAPT	○	●	●	●	○	○
ARS/SiMA	○	●	●	●	○	○
ATLANTIS	○	●	●	●	○	○
CAPS	○	●	●	●	○	○
CERA-CRANIUM	○	●	●	●	○	○
CHREST	○	●	●	●	○	○
CoJACK	○	●	●	●	○	○
CoSy	○	●	●	●	○	○
DIARC	○	●	●	●	○	○
DSO	○	●	●	●	○	○
FORR	○	●	●	●	○	○
MACSi	○	●	●	●	○	○
PolyScheme	○	●	●	●	○	○
RCS	○	●	●	●	○	○
SAL	○	●	●	●	○	○
Xapagy	○	●	●	○	●	○
CARACaS	○	●	○	●	○	○
CSE	○	●	○	●	○	○
Kismet	○	●	○	●	○	○
STAR	○	●	○	●	○	○
ARDIS	○	○	●	●	○	○
CRAM	○	○	●	○	●	○
ARMAR-6	○	●	○	○	●	●
IOCA	○	○	○	○	○	●

Figure 2.16: Comparison between hybrid cognitive architectures related to their use of memory.

Most cognitive architectures use *Procedural Memory (PM)* to store long-term facts about how to perform actions [Kotseruba and Tsotsos, 2018].

Semantic Memory (SM) also stores long-term facts, in this case about objects and relationships between them [Kotseruba and Tsotsos, 2018]. Semantic knowledge is typically implemented in a graphic representation inside an ontology, where nodes are concepts and edges are their relationships. This is similar to the use given to the SM in the framework presented in this work.

Table 2.1: Cognitive architectures using Working and Episodic Memory with its types.

Working Memory usage type	Cognitive architecture	Episodic Memory usage type
*Activation mechanism	ACT-R [Anderson, 1996]	*Coordinator
	CLARION [Sun, 2016]	
	Novamente [Goertzel and Pennachin, 2007]	
	SOAR [Laird and Mohan, 2014]	
	CELTS [Faghihi et al., 2013]	Experience
	DUAL [Kokinov et al., 1996]	
	NARS [Wang, 2006]	
	Sigma [Pynadath et al., 2014]	
World state	ARMAR-6 [Asfour et al., 2019]	
	MIDCA [Cox et al., 2016]	
	REM [Murdock and Goel, 2008]	
Spatiotemporal integration	iCub Ruesch et al. [2008]	
	ISAC [Kawamura et al., 2008]	
	Ymir [Alberts, 1995]	
Other	ASMO [Novianto and Williams, 2014]	
	CHARISMA [Conforth and Meng, 2011]	
	CogPrime [Goertzel, 2012]	
	GMU-BICA [Samsonovich et al., 2009]	
	LIDA [Ramamurthy and Baars, 2006]	
	Metacat [Marshall, 2002]	
	MLECOG [Starzyk and Graham, 2015]	

Sensory Memory receives and pre-processes incoming sensory data. This type of memory has been used to solve continuity and maintenance problems, such as identifying and retaining objects. Only the Attention-Driven Cognitive Architecture (ARCADIA) uses this approach [Bridewell and Bello, 2015]. This memory has been also used for perceptual binding and feature extraction for visual data in the Learning Intelligent Distribution Agent (LIDA) architecture [Ramamurthy and Baars, 2006]. It is not clear how other architectures use this type of memory. In this work, the implementation of a Sensory Memory is not included as the work of pre-processing perceptual information is part of the perception system; see Figure 3.2.

Some cognitive architectures using a Global Memory, that represents all knowledge in the same structure, are RoboCog and its successor CORTEX [Romero-Garcés et al., 2015], Distributed Practical Reasoning Architecture (DiPRA) [Pezzulo et al., 2007], Non-Axiomatic Reasoning System (NARS) [Kiliç and Wang, 2015] and Interaction-Oriented Cognitive Architecture (IOCA) [Pineda et al., 2010].

From the 19 cognitive architectures included in Table 2.1, only the fist four share similar usage of memory types related to this work. However, they have yet to be tested in PSRs. The only system using memory concepts tested in one service robot is SOAR [Puigbo et al., 2013]. In this work, memory modules extend the cognitive architecture CRAM [Beetz et al., 2010b, 2023], which is tested in various service robots. CRAM uses a plan-based control system in which a plan executive uses a SM to obtain knowledge about motions. That knowledge is integrated into designators, which store descriptions of entities such as objects, motion, grasps or poses. These designators are parameterized using vague information and then refined when more is available. CRAM also uses EM for improving its plan generation using experience. This work extends the memory capabilities of CRAM. More reading in this extension is presented in Chapter 3 and Chapter 6.

In the case of learning, the classification presented by Kotseruba and Tsotsos [2018] is declarative, perceptual, associative, non-associative and priming. *Declarative learning* refers to a collection of facts and their relationships. *Perceptual learning* changes its output depending on the sensory input, which means that it learns on-line depending on the changes in the environment. This can be seen as a reactive approach, as it modifies the behavior of the agent depending on the sensed information. *Associative learning* makes decisions based on rewards and punishments. Some computational models of associative learning are *Reinforcement Learning (RL)* introduced in more detail previously in Section 2.3.3. *Non-associative learning* does not require a connection between the input and response, i.e. the response does not depend on the stimuli. This type of learning is used in some social robots and *Human-Robot Interaction (HRI)* as it avoids extreme responses. *Priming* learning is based on the idea that the input can affect the identification or classification process. This type of learning is seen more in visual systems.

As can be seen in Figure 2.17, many cognitive architectures do not use learning. This has implications in the adaptability of those architectures to a changing environment, as they require an expert to introduce more features and behaviors. It is the interest of this work to look at the ones using associative learning, and more specifically *Reinforcement Learning (RL)*. They are presented in Table 2.2. One application of RL is a *selector* of actions or behaviors based on their success or failures [Kotseruba and Tsotsos, 2018], which is similar to this work and marked with an asterisk. Another application is as an *associator* for states and actions. In the case of SOAR, it uses RL alongside *chunking*, which is a learning mechanism for acquiring rules depending on experience and goals [Laird, 1988].

This work is interested in using cognitive features for robotic manipulation. It is important to know that object manipulation involves arm control to reach and grasp an object. In this sense, many architectures implement some form of arm control for reaching. However, gripping is more challenging as it depends on many factors, which include the features of the end effector (hand with fingers or gripper) and the properties of the object. Some architectures used for object manipulation are ISAC [Kawamura et al., 2008] for soft objects, DIARC [Scheutz et al., 2019] for objects with different grasping types and iCub [Ruesch et al., 2008] adapting to cans of different sizes, boxes and a ruler.

Cognitive architecture	Declarative	Perceptual	Procedural	Associative	Non-associative	Priming
iCub	●	●	●	●	●	○
MACSi	●	●	●	○	○	○
Novamente	●	●	●	○	○	○
CoSy	●	●	○	●	○	●
ACT-R	●	○	●	●	○	●
CLARION	●	○	●	●	○	●
NARS	●	○	●	●	○	●
CHREST	●	○	●	●	○	○
GMU-BICA	●	○	●	●	○	○
RCS	●	○	●	●	○	○
Soar	●	○	●	●	○	○
CogPrime	●	○	●	○	○	○
DIARC	●	○	●	○	○	○
DSO	●	○	○	●	○	●
CHARISMA	●	○	○	●	○	○
Pogamut	●	○	○	○	○	○
SAL	●	○	○	○	●	○
Xapagy	●	○	○	○	○	○
CRAM	●	○	○	○	○	○
ARMAR-6	●	○	○	○	○	○
LIDA	○	●	●	●	○	●
CARACaS	○	○	●	●	●	○
CSE	○	○	●	●	○	○
CoJACK	○	○	●	●	○	○
DiPRA	○	○	●	●	○	○
FORR	○	○	●	●	○	○
REM	○	○	●	●	○	●
Ymir	○	○	●	●	○	●
CELTS	○	○	●	○	○	○
ADAPT	○	○	●	○	○	○
CORTEX	○	○	●	○	○	○
RobCog	○	○	●	●	○	○
ASMO	○	○	○	●	○	○
MLECOG	○	○	○	●	●	○
ISAC	○	○	○	●	○	○
PolyScheme	○	○	○	●	○	○
RALPH	○	○	○	●	○	○
Sigma	○	○	○	○	○	○
Kismet	○	○	○	○	○	○
ARS/SIMA	○	○	○	○	●	○
DUAL	○	○	○	○	○	○
3T	○	○	○	○	○	○
ARCADIA	○	○	○	○	○	○
ARDIS	○	○	○	○	○	○
ATLANTIS	○	○	○	○	○	○
CAPS	○	○	○	○	○	○
CERA-CRANIUM	○	○	○	○	○	○
Copycat/Metacat	○	○	○	○	○	○
MIDCA	○	○	○	○	○	○
STAR	○	○	○	○	○	○
IOCA	○	○	○	○	○	○

Figure 2.17: Comparison between hybrid cognitive architectures related to their learning processes.

All the cognitive architectures presented have impressive features. However, they cannot re-use their capabilities or accumulate knowledge when applied to new tasks. Instead, every new task or skill is demonstrated using a separate model, a specific set of parameters or KB [Kotseruba and Tsotsos, 2018]. On the other hand, the framework proposed in this work adapts to new situations without the need for new models of the actions. This feature gives an advantage over

other cognitive architectures. Another advantage is the fact that it can adapt to different robotic platforms when other ones either were built specifically for one robotic platform or none at all.

Table 2.2: Cognitive architectures using associative learning and their type.

Associative learning type	**Cognitive architecture**
*Selector	4D/RCS [Albus, 2002]
	ACT-R [Lebiere et al., 2013]
	CHREST [Lloyd-Kelly et al., 2015]
	CoJACK [Ritter et al., 2012]
	FORR [Gordon et al., 2011]
	NARS [Wang, 2006]
	RALPH [Ogasawara, 1991]
	REM [Murdock and Goel, 2008]
Associator	iCub [Ruesch et al., 2008]
	ISAC [Kawamura et al., 2008]
Chunking	SOAR [Laird, 1988]
Other	ASMO [Novianto and Williams, 2014]
	CARACaS
	CHARISMA [Conforth and Meng, 2011]
	CLARION [Sun, 2016]
	CoSy
	CSE [Henderson et al., 2013]
	DiPRA [Pezzulo et al., 2007]
	DSO [Ng et al., 2012]
	GMU-BICA [Samsonovich et al., 2009]
	LIDA [Ramamurthy and Baars, 2006]
	MLECOG [Starzyk and Graham, 2015]
	PolyScheme
	Sigma [Pynadath et al., 2014]
	Ymir [Alberts, 1995]

2.5 Receiving instructions in natural language and parsing them

To make the framework complete, the robot needs to be able to receive instructions in a natural human way, such as language. It is for this reason that a natural language parser is required to transform the input instructions into commands that the robot can understand; see Figure 2.20. A parser usually works in a sequence. First, it scans the inputs and produces tokens or a sequence of characters that match the input by following specified rules. Then it analyzes their grammatical components and produces a result. One example is presented in Figure 2.18, wherein, in this case, each word is classified. In this example, there is only one verb `bring` and three nouns (`me`, `glass` and `water`).

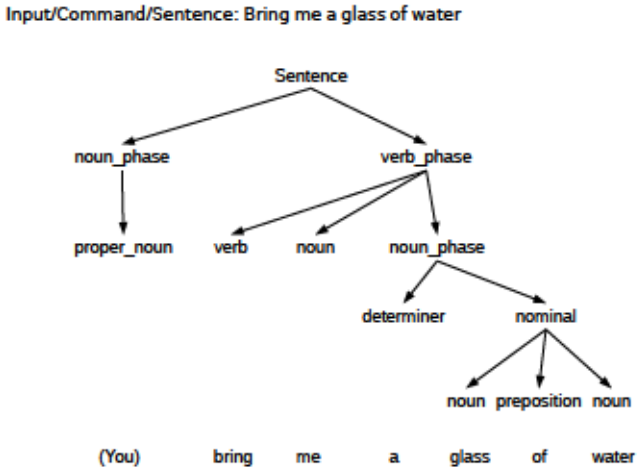


Figure 2.18: Parser example of instruction with an implicit subject or proper noun.

For parsing, a lexicon or an inventory of specific knowledge is required for semantic distinctions inside the grammatical classes. For example, nouns tend to denote objects, verbs actions and adjectives object properties. Even if they are not perfect, these grammatical classes can have the most commonly-shared semantic features of a class [Vinson and Vigliocco, 2002, Laird and Mohan, 2014].

In this work, the Probabilistic Action Cores (PRAC) framework [Nyga and Beetz, 2018] is used for parsing natural language instructions. It is capable of learning specific probabilistic KBs and reasoning about action or hand-labeled instructions; see Figure 2.19.

PRAC can compute the plan instantiation with the best likelihood of achieving the desired action from the provided instruction. Its main components are a specific plan library, KB and dictionary that can be trained off-line. PRAC's dictionary provides all possible word meanings that can occur in natural language instruction. Those meanings are defined in the WordNet [Miller, 1995] dictionary, which comprises more than 117,000 concepts. PRAC's Action Core Library contains a collection of verbs (action cores), which represent how that action can be constructed on a

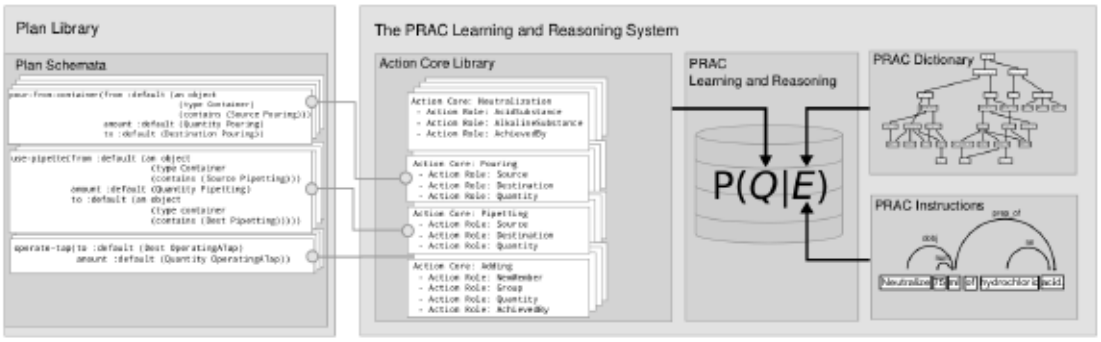


Figure 2.19: PRAC key concepts and their role in inferring most probable instruction. Reprinted from "Cloud-Based Probabilistic Knowledge Services for Instruction Interpretation" by Nyga, Daniel and Beetz, Michael. 2018, Robotics Research 2, p. 653. Copyright 2018 by Springer.

conceptual level. This concept level can be in terms of conjunctions of logical assertions over the predicates as `action core(a, Action)`, `theme(a, t)`, `source(a, s)`, `destination(a, d)`, etc. as seen in Figure 2.20. Additionaly to using probabilistic reasoning methods, PRAC uses the principles of analogical reasoning in semantic networks [Nyga et al., 2017]. This is done by building the mentioned KB of semantically annotated instruction sheets found in the Web.

Finally, PRAC's plan library contains action-specific plans in a computable way from PRAC's Action Core Library. However, it depends on a system having a plan library already, as the plans themselves are considered black boxes in PRAC's reasoning. The plans at the end have to be defined by a human.

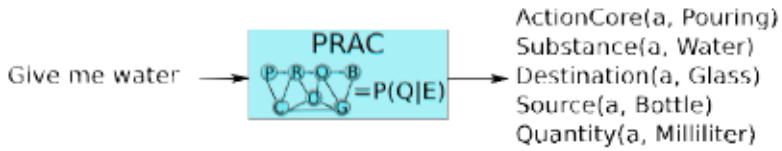


Figure 2.20: PRAC usage example.

As it can be seen in Figure 2.20, PRAC's plan instantiation output for pouring water into glass requires further work to be applied to the robot control. If the robot does not have a specification of the movements required for pouring, it would not perform such action. In this case, sub-actions required by the action pouring are added with the level of specification required by the arm controller. In this case, sub-actions are one level above the muscle or joint movement specification. The *Working Memory (WM)* is responsible for further processing and refinement to formulate an executable plan.

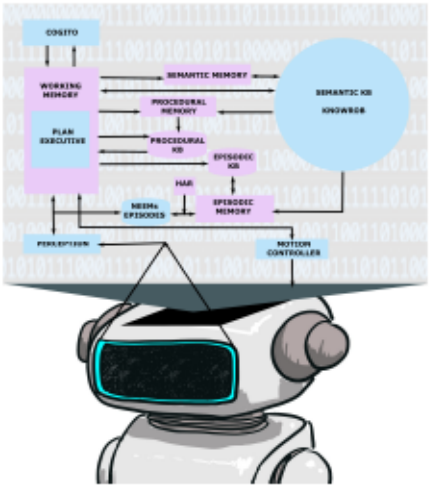
2.6 Summary of this chapter

In this chapter, some requirements that are needed by a Personal Service Robot (PSR) are presented. One of these requirements is complex manipulation capabilities, which is the focus of this work. PSRs need to be tested in scenarios before having them available in homes. Even when robots already have these capabilities, they still lack more complex manipulation skills required for caregiving tasks. By taking this into account, an extensive search of robotic platforms in industry and academia was performed with the idea of looking at the manipulation hardware available and their way of complex handling manipulation. This helped to select the robotic platforms to test the proposed architecture extension.

Finding out that only a few PSRs, a total of 5, were already tested in real homes gives the intuition that manipulation capabilities still require more work. PSRs manipulation requires a solid hardware infrastructure and system features such as knowledge acquisition, planning and driver control. For this reason this thesis work focuses on the high layer to provide and store sufficient knowledge for PSRs to perform complex manipulation actions. The scope of this work includes building a structure of actions in a plan for the robot to perform.

Chapter 3

Overview of the knowledge processing memory modules



As mentioned, particular capabilities are still required to use Personal Service Robots (PSRs) for caregiving. By keeping that in mind, previous Section 2.1.2 presents the physical capabilities needed by PSRs to assist humans in their daily lives. Furthermore, Section 2.4 shows the required control capabilities to solve such tasks.

The type of tasks involved in caregiving include serving food and cooking actions, which are the main focus of this work. However, it is not only about how these actions can be executed but also how their execution can be improved. One example of a serving action, presented before, is pouring a drink (Figure 3.1). The considerations of the robot include the angle of the box container and the container receiving, speed of angle change, box container deformation and amount of liquid.

Figure 3.1: Robot pouring juice from a box container into a glass.

As mentioned, the capabilities required by the robot to solve the task are included in a cognitive architecture. Furthermore, five robots were selected to test the ideas behind this thesis work. Now, it is time to introduce the extended cognitive architecture to enhance the capabilities of these robots.

This work proposes extending the cognitive architecture CRAM [Beetz et al., 2023]; see Section 3.2. This extension adds knowledge processing modules based on memory models of the human brain and a HAR system to introduce the data from demonstrations. These cognitive modules and their interconnections are illustrated next in Figure 3.2. The modules extending CRAM are marked in color pink.

This work uses the ROS middleware shown previously in Section 2.2. The PSR uses all these modules to act in either real or simulated environments.

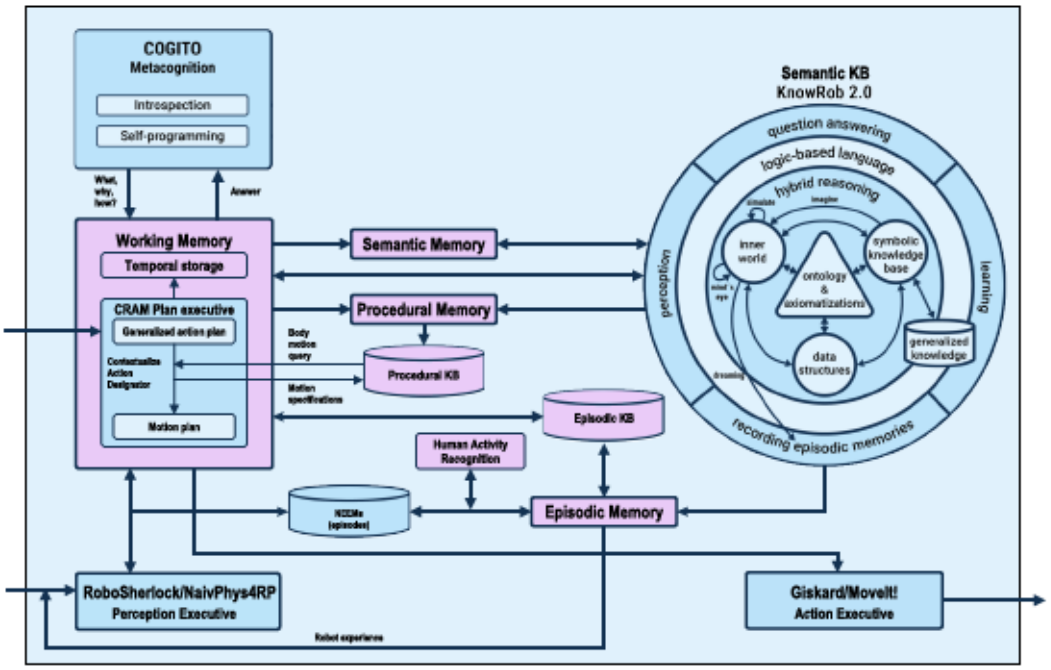


Figure 3.2: Memory-based knowledge processing modules inside the cognitive architecture CRAM. Modified from "The CRAM Cognitive Architecture for Robot Manipulation in Everyday Activities" by Michael Beetz, Gayane Kazhoyan and David Vernon. 2021, IEEE Transactions on Cognitive and Developmental Systems, p. 9. Copyright 2021 by IEEE.

CRAM uses CRAM-PL, a reactive programming language that provides a set of control structures for modular programming. It permits activities to be effectively executed by stating what action must be carried out but not *how*. This action specification is written vaguely. When individuals ask someone to perform something, they usually offer ambiguous directions. When the ambiguity is resolved with more descriptions of the action, the chances of successfully executing it increase. This means that actions are filled with specifications from the incoming information of the world state.

Actions are divided into primitive movements in CRAM-PL, each including parameters whose values define the motion's exact nature. CRAM employs information to translate from desired

Lisp Code 1 Move head designator.

```
1 (an object (type plate) (color white)) ;; Object designator
2 (a location (on table) (in kitchen)) ;; Location designator
3 (an action (to grasp) (an object (type plate))) ;; Action designator
```

action results to the most likely motion parameter values. In the case of objects, such information includes shape, weight, type and components, among others. The action can consist of the grasp type, for example.

CRAM uses *designators* to describe entities such as objects, motions, grasps or poses. A designator is an element of the plan that is a placeholder for yet-to-be-determined information. The designator is resolved and the related information is determined at run time based on the current context of the action. There are four designator types: action, object, location and motion. *Action designators* focus on achieving some goal state, e.g., `setting` the table for a meal or `placing` dirty dishes in the dishwasher. *Location designators* are concerned with poses in general, e.g., a list of positions and orientations where a robot should stand to perform some manipulation. On the other hand, *motion designators* are related to the physical movements and the control of some actuators, e.g., moving the end-effector to a given pose or opening the gripper. Finally, *object designators* are concerned with the properties of objects in the robot's environment, e.g., the pose of the object and its physical characteristics. They provide an interface to the perception system since the pose of an object will typically be determined at run time.

Designators can be seen as objects containing sequences of key-value pairs symbols. Each pair's value element acts as a placeholder for information required by the plan or for the execution of a motor command. As mentioned in previous chapters, the information is acquired by resolving the designator at run time by querying knowledge from *Knowledge Bases (KBs)* and accessing sensor data through the perception executive.

Examples of designators can be seen in the Lisp Code 1. The object designator requires features like type or color. The location designator gets a specification of locations inside a room and the room. Action designators can include other designators inside, such as an object one.

As mentioned, the designator is resolved by querying knowledge from the plan, using the KBs or obtaining sensorimotor data via perception. The robot body moves in a defined trajectory when a motion designator is resolved.

To execute a task, the plan modules communicate with the robot's control system, including functionalities such as object perception, robot navigation and localization. The structure of an action plan created in CRAM can be seen in the Lisp Code 2. This plan was modified by this thesis work to add the sub-actions. First, the plan's name is given (`plan-name`), followed by a list of parameters (`parameters-list`) required by the plan. Then, when the list of preconditions (`required-preconditions`) is satisfied, the plan can be executed with its specific values

Lisp Code 2 Generalized CRAM plan.

```
(def-plan plan-name ( parameters-list )
(when ( required-preconditions )
(perform
(an action
(type ?action-category)
( key-1 value-1 )
( key-2 value-2 )
. . .
)
)
)
)
```

(key-1, key-2, etc.). Finally, plans include an `action-category` corresponding to action verbs, e.g., `fetch`, `place`, `pour`, `cut`.

The structure of the execution plan is generated inside a memory module called *Working Memory (WM)*, which has not been used before. This module chooses the best option by computing the probability of success based on previous episodes in the *Episodic Memory (EM)* and *Procedural Memory (PM)* modules. This thesis work accommodates plans into actions and sub-actions, in which each task has specified action or motion plans. The difference between movements, sub-actions and actions are defined in a taxonomy, where actions and sub-actions are classes. Movements are defined as features of the sub-action. An advantage is that when sub-actions are combined, they provide flexibility to the plan's structure and allow multiple interconnections.

The *Plan Executive* interprets the action plan in a process called *contextualization* into steps [Kazhoyan et al., 2021]. It receives the sequence of actions and monitors the action execution. It can use a Heuristics module to receive a symbolic location description to find a specific pose in the robot's environment. It also communicates with the Perception system to receive symbolic descriptions of objects seen in the environment.

Considering the `pick-up` plan presented in Lisp Code 3, the sub-actions are inside the action `picking-up`. The first step is to insert the arguments required for the `picking-up` action. In this case, it can be the object's type to be manipulated. The argument or arguments are designators.

With the `with-robot-at-location` construct, the plan language ensures that the robot is located appropriately for action execution. This depends on the object, robot capabilities, environment and task context. For this example, the robot should stand in a location where it can perceive and reach for the object. When the robot is appropriately positioned, the plan continues its execution. If the robot is not at the appropriate location, the execution is suspended, repositions itself, and only then continues the execution.

Then, the action plan is instantiated by adding the parameters needed to execute the motion plan, e.g., which arm, trajectory and grasp pose to use. Next, a query is created to retrieve the values for these parameters to the *Procedural Knowledge Base (PKB)*. This query can be seen in Lisp Code 4. This step returns the robot body motions to achieve the goal of the underdetermined

Lisp Code 3 Pick-Up plan.

```
(def-plan pick-up (?object-to-pick)
  (with-robot-at-location (?location-at-which-to-pick)
    (perform
      (an action
        (type picking-up)
        (object (an object (type ?object-to-pick)))
        (arm ?arm-to-be-used)
        ;; Sub-actions
        (perceive ?object-to-pick)
        (reach ?location-at-which-to-grasp)
        (grasp ?grasp-pose)
        (lift ?lift-pose-to-be-used)
        (retract ?arm-used)
      )
    )
  )
)
```

action description and the associated instantiated extended plan. The *Procedural Knowledge Base (PKB)* samples a joint distribution of the motion parameter values and the related outcome from previous executions. The query arguments are the `key-value` pairs that were not passed as arguments to the pick-up plan.

Finally, *Action Executive* executes the plan. In this case, the interface with MoveIt! produces the robot's movements. MoveIt! is explained in more detail in Section 3.1.4.

Additionally, CRAM has an implementation, that was not done in this work, capable of a fast plan projection to predicted parameters that could lead to a successful execution [Kazhoyan and Beetz, 2019]. This projection is used only as a suggestion to the planner as it requires an accurate representation of the world. This is why knowledge processing memory modules were preferred for this work implementation.

Action and sub-action selection is based on relevance, which is how well the action corresponds to the current situation [Kotseruba and Tsotsos, 2018]. This is done by checking pre- and post-conditions of actions before applying them. The action and sub-action selection also depends on a score given to the pair depending on their contribution to achieving the current goal. The score is given by the performance of the action in the past and the potential to improve the behavior in the future via *Reinforcement Learning (RL)*.

Similarly to previous plan creations, the plan is a sequence of calls to perform actions by calling its respective subplan [Kazhoyan and Beetz, 2017, Kazhoyan et al., 2020b]. However, in this thesis implementation the separation between action and sub-actions is defined by a more significant granularity than previously. This is done by using definitions from joint movement and atomic actions instead of force-dynamic events.

Unlike some hybrid approaches introduced in Section 2.4.2, this work assumes that knowledge can be available during plan formulation and execution. The most significant contributions of this work are the biologically inspired memory concepts and their interconnection; refer to

Lisp Code 4 Fetch and place.

```
(query-variables (?location-at-which-to-pick, ?arm-to-be-used, ?grasp-pose
    , ?lift-pose-to-be-used)
  to-succeed (
    (with-robot-at-location (?location-at-which-to-fetch)
      (perform
        (an go
          (type navigate)
          (location (?location-at-which-to-fetch)
        )
        (an action
          (type picking-up)
          (object (an object (type ?object-to-be-fetched)))
          (arm ?arm-to-be-used)
          (grasp ?grasp-pose)
          (lift-pose ?lift-pose-to-be-used)
        )
        (an go
          (type navigate)
          (location (?location-at-which-to-deliver)
        )
        (an action
          (type putting-down)
          (object (an object (type ?object-fetched)))
          (arm ?arm-used)
          (lower-pose ?lower-pose-to-be-used)
        )
      )
    )
  )
)
```

Chapter 6 for more insights. According to work by Vernon et al. [2016], memory keeps what has been achieved through learning and development. It ensures that a cognitive system adapts to new circumstances. Memory allows a cognitive agent to prepare to act.

This work also presents how manipulation actions are represented on two levels, actions and sub-actions. This representation is stored in episodes, an *Episodic Knowledge Base (EKB)*, a *Procedural Knowledge Base (PKB)* and *Semantic Knowledge Base (SKB)*, depending on the level of the representation.

To start, this framework uses concepts of memory from neuroscience. As human examples are used, it is also fair to use an approximation of human memory models. Even more important than the concepts used is how they work together to achieve the expected result.

Section 3.1 presents work that this thesis applied to complement the functionality of the cognitive architecture. Finally, section 3.2 presents the implementation made for this thesis work and is marked in pink in Figure 3.2.

3.1 Existing systems

3.1.1 KnowRob

KnowRob is used as the *Semantic Knowledge Base (SKB)* in this work as it provides an action-centered KB [Tenorth et al., 2014, Tenorth and Beetz, 2017], which integrates various types of knowledge (static encyclopedic, commonsense, task descriptions, environment models, object information, observed actions, etc.) from different sources (manually axiomatized, derived from observations or imported from the web); see Figure 3.3. All representations are combined with semantic properties, such as classes they belong to and features. For example, the cup belongs to the class container and has a handle. In the case of actions, they are represented by their associated motions. Sub-actions are not described as such but as sub-events. These sub-events are defined as instances of an action. For example, the action `PickingUp` has the sub-events `subEvent:Reaching` and `subEvent:TakingSth`. On the other hand, sub-actions are actions in the lower hierarchy and hold all features besides the bottom levels.

The advantage of using *Ontology Web Language (OWL)* [Tenorth et al., 2010b] is that it balances expressiveness and reasoning capabilities well. From a robotic platform perspective, it is possible to translate low-level sensor data into knowledge. OWL is a form of description logic and distinction between classes and instances. General knowledge about object types is modeled into classes. Classes may be hierarchically structured and inherit the properties of their (potentially multiple) parents.

To equip robots with commonsense knowledge, KnowRob uses the Open Mind Indoor Common Sense project (OMICS) was created for mobile robots. OMICS [Havasi et al., 2010] and its previous version ConceptNet [Havasi et al., 2007], contain detailed action-related knowledge about everyday objects. In the case of robots, one way to represent knowledge is presented in

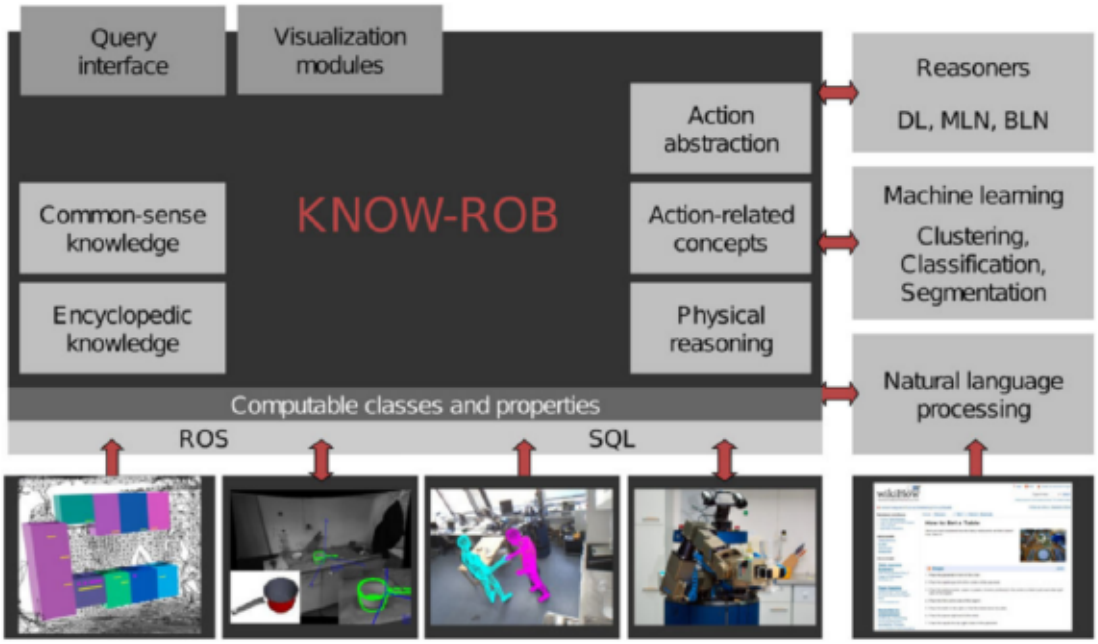


Figure 3.3: KNOWROB system overview. Reprinted from "Knowledge Processing for Cognitive Robots" by Tenorth, Moritz, Jain, Dominik and Beetz, Michael. 2010, KI - Künstliche Intelligenz 24(3), p. 235. Copyright 2018 by Springer.

Beetz et al. [2012] as a procedural attachment extracted from logged data trees. A *Procedural Knowledge Base (PKB)* uses this type of representation in the implementation created for this work.

KNOWROB 2.0 provides a query language to retrieve information from *episodes*. In the KNOWROB 2.0 ontology, all entities of a particular entity category may be retrieved, and each can be described using the characteristics provided for that category. Furthermore, the ontology's relations may be utilized to limit combinations of entities.

KNOWROB uses a weak closed-world assumption. It is a closed-world assumption weak because the robot is still required to detect novel objects from the environment. During its reasoning processes, the robot assumes to know all objects but concurrently monitors its percepts for new objects and updates its belief state whenever a new object is detected. The KB of the robot is populated with object models that consist of CAD models, including the part structure and possible articulation models, a texture model and encyclopedic, commonsense, and intuitive physics knowledge about the object.

Additionally, KNOWROB supports semantic maps [Bozcuoglu et al., 2018]. They are semantic descriptions of environments represented in the ontology format. They include knowledge about objects with their physical properties and pose information. For example, an object A of type Fridge can be inferred to be a container because it is defined as a superclass in the human household knowledge ontology inside KNOWROB.

One disadvantage of KNOWROB is that its classes are mainly created manually and require an expert to provide them. In this work, a different approach is evaluated in which knowledge is added to other KBs automatically and tested by a robot.

3.1.2 Episode or NEEM

The system-generated episodes are known as *narrative-enabled episodic memories (NEEMs)*. A NEEM comprises two parts, which are experience and narrative. The NEEM experience records low-level data such as the agent's sensor information, such as pictures from a camera and forces from a gripper, and the agent's and its observed objects' postures. NEEM experiences are related to NEEM narratives, symbolic tales about the occurrence. These narratives include information on the activities, the context, the intended aims, the observed impacts, and so on. The NEEM experience and NEEM narrative are so rich in information that the agent can repeat an episode to relive the witnessed behavior. NEEMs are experiences gained by experimenting, reading, observation, mental modelling, etc.

The primary purpose of a NEEM is to design a model to represent experience data. Also, it is to provide a standard vocabulary for annotating experience data across diverse activities, scientific fields and acquisition modes. The vocabulary is more than simply a collection of atomic labels; each label is defined in an *ontology* [Euzenat and Shvaiko, 2007]. These definitions are written so that a knowledge base equipped with such *ontology* and a collection of NEEMs can answer a series of competence questions about an activity. The NEEM model is explicitly specified as an OWL ontology [McGuinness and van Harmelen, 2004]; see Chapter 4 for more details.

One good reason to use NEEMs is that they are another source of information besides encyclopedic knowledge, e.g., KNOWROB. They are user for robots to be capable of reasoning about which parameters led to successful executions and which conditions resulted in failures before.

3.1.3 NaivPhys4RP before RoboSherlock

Personal Service Robots (PSRs) need methods for recognizing them and their location (pose) to manipulate these objects. For it, the robot requires perception, which is a process that transforms input information from the sensors into internal representations that can be used by the robot [Kotseruba and Tsotsos, 2018]. Human sensor types are vision, hearing, smell, touch and taste. In robots, standard sensors are related to vision, touch and, in some cases, hearing.

To process the sensor information, CRAM uses the NAIVPHYS4RP system [Kenghagho Kenfack et al., 2022, Beetz et al., 2015]. NAIVPHYS4RP integrates various perception algorithms to answer relevant questions to solve the task. Such questions can be related to objects in the environment. It is also capable of reasoning about those objects. NAIVPHYS4RP can solve some issues associated with the robotic perception mentioned before. It can deal with transformations from the objects (batter to a pancake, for example) by combining perception, representation and reasoning.

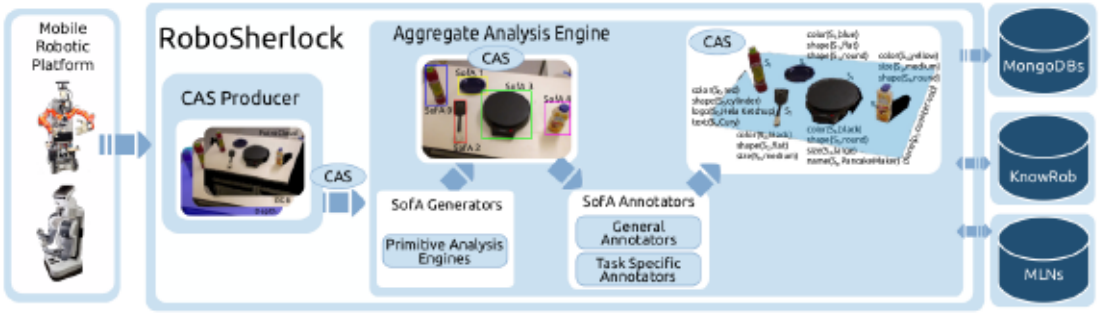


Figure 3.4: Example of ROBOSHERLOCK execution. Reprinted from "RoboSherlock: Unstructured information processing for robot perception" by Beetz, M., Bálint-Benczédi, F., Blodow, N., Nyga, D., Wiedemeyer, T. and Márton, Z. C. 2015, Proceedings 2015 in IEEE International Conference on Robotics and Automation, p. 1552. Copyright 2015 by IEEE.

Lisp Code 5 Perceive query.

```
1 detect (an object
2   (category ?category)
3   ( key-1 value-1 )
4   ( key-2 value-2 )
5   ...
6 )
```

NAIVPHYS4RP uses the Unstructured Information Management Architecture (UIMA) to split processes into several expert modules called *Annotators*, which analyze the incoming data. One of these annotators is capable of combining prior knowledge with results from a CNN to distinguish objects with similar appearance [Richter-Klug et al., 2022].

One of the central data structures from UIMA is the Common Analysis Structure (CAS), which is filled with data during runtime and erased when it ends. When a hypothesis is generated, it is assigned to a data structure called *Subject of Analysis (SOFA)* with n unique name. It collects and organizes all the information coming from different components. NAIVPHYS4RP's output is represented as events. Events represent dynamic situations in which the state of the world can change [Tenorth, 2011, p. 44]; in this case, they represent the robot's belief. Multiple events can be assigned to one perceived object to describe the perception results of the current state compared to the expected condition to achieve an action.

As seen in Figure 3.4, NAIVPHYS4RP uses KNOWROB to obtain features of actions. This represents the importance of the connection between perception and semantic memory in action execution. When NAIVPHYS4RP results are ready, they are sent to the *Working Memory (WM)*.

A general way of sending information requests to NAIVPHYS4RP is presented in the Lisp Code 5. If we take the example of a cup, the result is `(category container)`, `(shape cylinder)`, `(color red)` as part-of the object's description. The `category` property permits the application of self-defined categories, e.g., a container in the case of the cup mentioned before. NAIVPHYS4RP can also complete perceptual tasks such as identifying items that meet

criteria, such as "a container that can store a liter of fluid," by integrating visual detection with knowledge-enabled reasoning and other computations, as computing volumes.

3.1.4 MoveIt!

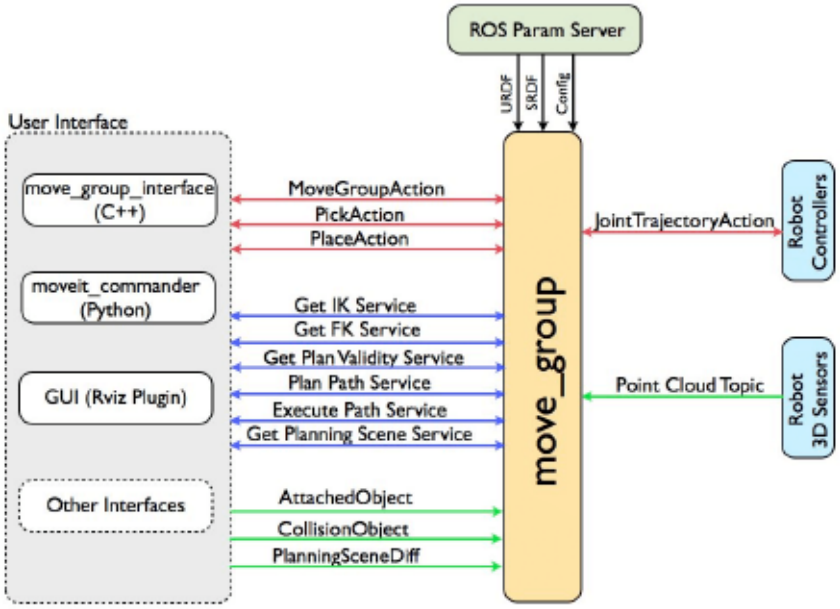


Figure 3.5: MoveIt! move_group architecture. Reprinted from "MoveIt!: An Introduction" by Chitta, S. 2016, Robot Operating System (ROS): The Complete Reference (Volume 1), p. 3–27. Copyright 2016 by Springer International Publishing.

MoveIt! [Chitta et al., 2012] is a system capable of motion planning, generating trajectories and monitoring of the environment for many different robotic platforms. It includes the controllers for the motor systems of the five selected robots in this thesis work.

MoveIt! transforms the plan into motions for the motor system to perform and is available in the middleware *Robot Operating System (ROS)*. This makes possible the connection to the extended cognitive architecture presented here.

The high-level system architecture for MoveIt's principal node, `move_group`, is shown in Figure 3.5. This node acts as an integrator, bringing all the components to give the robot's system a set of ROS actions and services. The user interface allows access to actions and services provided by `move_group`.

MoveIt interacts with various motion planners. The motion planners are accessed via a ROS Action or service (provided by the *move_group* node). This enables MoveIt to employ a variety of libraries as planners, allowing it to be readily extended.

In response to your motion plan request, `move_group` will produce the desired trajectory. The arm (or any set of joints) will be moved to the target place using this trajectory. Note that the output of `move_group` is a `JointTrajectoryAction`, not simply a route; that obeys velocity and

acceleration restrictions at the joint level using the desired maximum velocities and accelerations (if provided).

3.2 Knowledge processing memory-based models

The cognitive architecture CRAM [Beetz et al., 2010b] extension combines theories of cognition and functionality. This extension includes adding memory modules to process knowledge, as shown in Figure 3.2 in pink. This means it covers some of the desired features that Vernon et al. [2016] presented, such as physical embodiment, sensorimotor contingencies, perception, prospective action, declarative and procedural memory, learning, internal simulation and autonomy. However, its main focus is the high layer defined by the memory types used and their interconnections, explained in more detail in Chapter 6.

In the following sections, the modules extending CRAM are presented. They start with the WM in Section 3.2.1. Later, EM is shown, including the HAR implementation for using human examples in Section 3.2.2. Finally, it continues with the PM (see Section 3.2.3) to finalize with the SM (see Section 3.2.4).

3.2.1 Working Memory to Interconnect memories

In this work, the *Working Memory (WM)* is the link between *Long Term Memory (LTM)* and the *Short Term Memory (STM)*. The difference between this approach and other hybrid cognitive architectures is that the WM is not considered to belong only to the *Short Term Memory (STM)* but also to share features of the *Long Term Memory (LTM)*. In the case of *Working Memory (WM)*, some cognitive architectures use it as an *activation mechanism* or filter to decide if knowledge should enter the other memories. Furthermore, this memory can extract knowledge from *Episodic Memory (EM)*, *Procedural Memory (PM)* and *Semantic Memory (SM)* to build an action structure for execution.

WM keeps track of the execution until it is finished and can update fast by keeping current information flowing. It connects to other memory types to store information in the long term. WM is also used as a processing system for the state of the environment built by the perception system, in this case, ROBOSHERLOCK. These features allow the system to perform actions depending on the current state of the world in an adequate time interval. WM includes an integration of spatiotemporal knowledge to interact with the environment. For example, furniture's dimensions, cutlery, dishes, etc.

This allows the robots to have a sense of space and form when manipulating objects and detecting changes. WM will enable robots to reason about the effects of their actions.

3.2.2 Episodic Memory to get previous experiences

This thesis work saves and uses actions executions to use in future behavior. In this case, *Episodic Memory (EM)* is a coordinator of previous executions. EM gets knowledge from the *Semantic Memory (SM)* and shares knowledge with the *Working Memory (WM)*.

In this case, robots build representations of the world and store them as episodes. They include when, what, which and where actions happened in a sequence. Some episodes come from human demonstrations, and others from robot executions.

Human demonstrations are an initial source of knowledge for unknown actions or complex manipulations where the robot can not find a suitable solution, for example, when an object keeps falling, breaking or spilling. In the case of human demonstrations, action-sub-action pairs are segmented from a *Virtual Reality (VR)* environment where humans can perform actions. The segmentation is performed by a HAR system implemented in this thesis work.

Sub-action features can be used during the planning and the actions hierarchy via reasoning using queries. The segmented pairs provide information regarding sub-action features, such as velocities, distances, success in limited cases, objects acted on, etc. The final representation includes those pairs and the events, including other sub-action features, such as hand used, grasp type and object acted on. The WM stores the episodes from robots. The robot can improve further when it uses the episode's information for future performance.

EM can create its *Episodic Knowledge Base (EKB)* by previous actions and their results to the most recent ones. That way, this KB keeps an updated version of the episode's results by including the one that performed best, worst, time of accepted execution, etc. In this KB, the most successful task has a higher priority to be selected.

EM retrieves information from the EKB and the episodes in particular cases required by the WM. EM cannot change episodes but extract their features. This process happens every time the robot finishes executing a task.

Like I mentioned before, human examples are used in this work. The implementation to store them is presented next.

Human Activity Recognition implementation

This work uses the observations of humans while performing actions in VR using the system presented by Haidu and Beetz [2016]. This system records the position and orientation of three body parts, i.e., the head, right and left hand. The recorded data includes raw and high-level data. Inside the raw data, the position and orientation of all objects in the environment are tracked over time. The system is extended by adding more objects to its environment.

A HAR system recognizes and classifies action and sub-action pairs using human examples. This implementation uses machine learning techniques described in more detail in Chapter 5. The implementation can recognize some action features, including—but not limited to—grasping

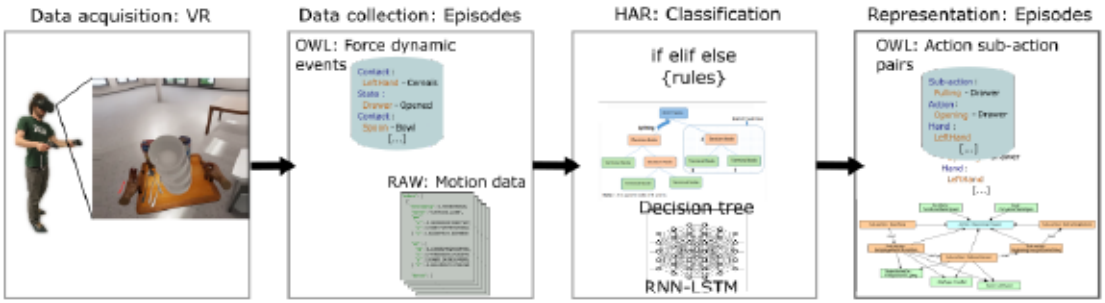


Figure 3.6: HAR process steps.

types as the procedural features from those actions. The process is described in Figure 3.6, from the data acquisition to the result of the action-sub-action pair.

After going through a process of *human activity recognition (HAR)* where actions are recognized, the actions are represented in an execution tree as episodes, see Figure 3.7. In this representation, the sub-action is shown as the center, surrounded by the action it belongs to and its features. Each sub-action includes the action it belongs to and details related to procedural features, such as grasp types and grasp position choice, used when taking an object, source of the object, etc.

The episode representation stores lower-level information like object positions in the environment, their poses, etc., and temporal synchronization using global time. The representation of logged actions builds upon the *Semantic Knowledge Base (SKB)* via a belief state, which provides a structure to represent tasks and their spatial and temporal context. This includes the events, objects, environment maps and body parts.

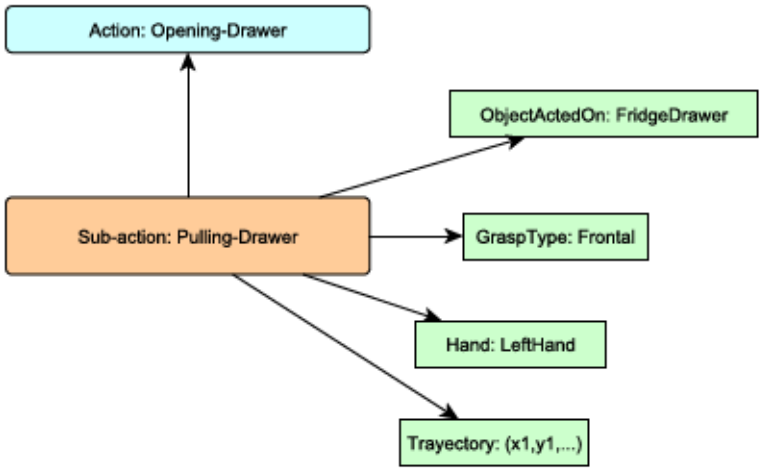


Figure 3.7: Sub-action tree representation in episodes with its action and features.

Compared to previous implementation of planning inside CRAM using brute-force search and heuristics [Koralewski et al., 2019], this work uses imitation learning and learning based on experience.

3.2.3 Procedural Memory in action representation

Procedural Memory (PM) is implemented to store long-term facts about how to perform actions. This knowledge comes directly from the *Working Memory (WM)* after the execution of a task. Then, it is stored inside a *Procedural Knowledge Base (PKB)*. PM updates its KB after every task execution.

PKB's first entry comes from the actions and knowledge present in the *Semantic Knowledge Base (SKB)*. Actions and sub-actions are stored in the PKB. Sub-actions include `properties` such as `bodyPartsUsed`, `subEvents` and `nextMotion`. Actions include features such as `prevAction` and `nextAction`.

The entries of the SKB are improved over time. The information stored afterward takes into account stability by taking into consideration the execution time and success returned by the WM. The representation inside the SKB is defined in three levels. The first is the top-level where objects and their relationship with actions are defined as abstractions and symbols. The second is called a mid-level sequencer, in which actions are chained in a structure for execution.

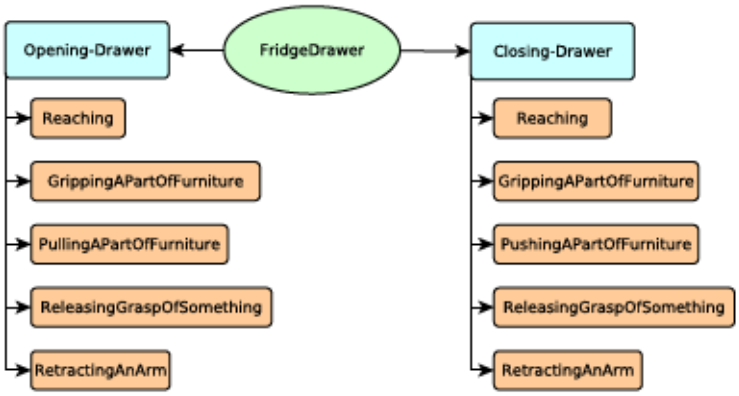


Figure 3.8: Procedural Knowledge Base representation.

This representation allows the *Working Memory (WM)* and *Procedural Memory (PM)* to access procedural knowledge by object, action or sub-action. This access type depends on the requirements of the system during execution time.

3.2.4 Semantic Memory representing objects and actions as concepts

Semantic Memory (SM) stores long-term facts, in this case, especially about actions, objects and relationships. The SKB has a graphic representation inside an ontology, where nodes are concepts and edges are their relationships. This work relies mainly on the work by Tenorth et al. [2010a] regarding KNOWROB inside the SKB.

KNOWROB was missing some information about concepts from fluid dynamics and the naming of cooking-related actions and movements from joints. For this reason, this thesis work updated and extended it by verifying knowledge about physics concepts, actions, sub-actions, motions

and objects, as presented in Chapter 6, Section 6.2.4. In KNOWROB, actions are represented as events initiated by agents to achieve the desired effect. This makes it possible to describe these events using the same structures as exogenous events like sensor readings or the expression of a dialog partner. However, in previous inplementations, intermediate subtasks were defined for `perceiving`, `reaching`, `grasping` and `lifting` an object. Now, they are described as `subAction` and directly associated with the overall goal.

In this work, knowledge obtained from the execution of manipulation actions is compared with the one already present in the SKB, which includes KNOWROB. To be able to add, extend and update knowledge in the SKB, this work created a verification function inside the SM. This function receives the classes and properties and verifies if there are already inside the SKB. In a first instance, they were defined manually offline and integrated. However, this implementation permits that new information coming from the WM or perception system to be added automatically into the SKB. This means that the WM is in charge of submitting new knowledge to the SM to be integrated later into the SKB.

The SM is in charge of managing the knowledge inside the SKB. If the concept is missing, it is added. If the concept already exists and it is the same, nothing happens. However, if the concept exists and is contradictory, a new round of tests is performed to select which one to keep.

When an instruction is received from the parser, *Working Memory (WM)* can read the concepts related to the actions of interest present in the SKB and extracts information about objects and actions. The EM and pm also make use of the knowledge present in the SKB to give meaning to their knowledge.

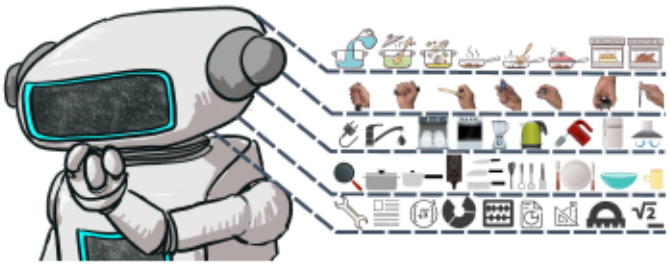
3.3 Summary of this chapter

This chapter presents the memory extension of the cognitive architecture CRAM. It starts by introducing CRAM's functionality and part of the extensions implemented. Then, it explains the functionality of modules used but not implemented in this work. Last, it explains the architecture's modules implemented and how they receive and send information to each other.

This chapter also provides a direction to the other chapters where more information can be found on the specific implemented modules.

Chapter 4

Knowledge handling by Personal Service Robots



As mentioned in previous chapters, a Personal Service Robot (PSR) requires knowledge of how to perform manipulation actions in human environments. In this sense, knowledge is the source of representations of that environment. Such knowledge should be general enough, as it is needed when there is a task in which questions should be answered [Lally et al., 2014]. In human brains, representations support inference and decision-making. For this reason, it is necessary to build representations in robotic platforms, as a programmer can not add every situation they might encounter while executing a task.

There is already some effort to include knowledge in robotic platforms. This happens because knowledge of action outcomes allows agents to perform well in complex environments [Russell and Norvig, 2003]. Agents (robots) using knowledge for this purpose are called *knowledge-based agents*. Knowledge, in that case, can be expressed in very general forms, combining and recombining information to suit different purposes. A knowledge-based agent can combine general and current knowledge about the environment to infer hidden aspects of its current situation before selecting actions. This agent can learn new knowledge about the environment and adapt to changes by updating the relevant knowledge. In some cases, it is by building and storing concepts into knowledge bases; see Section 4.1.1.

For this work, we are interested in representing manipulation actions in an extended manner, referred to as *commonsense knowledge*. How manipulation actions, in particular, are performed in detail corresponds to the commonsense that humans develop since infancy while interacting with objects inside the environment. It is possible to derive this kind of knowledge from activity models from human examples.However, these models still require further observations to be complete by learning relations between them [Tenorth et al., 2010a].

The premise of this chapter is that for high-level cognitive processing, a machine has to be capable of generating, representing and storing knowledge. The works by Newell [1982], Newell et al. [1989], Anderson [1996] and Anderson et al. [2004] provide explanations for the relationship between these processes in human cognition and computer systems emulating them. Their work strongly influenced on recent *Knowledge Representation (KR)* models, as well as this work. This work uses the combination of *Long Term Memory (LTM)* and *Short Term Memory (STM)* to emulate human cognition.

As we want robotic platforms to be able to perform everyday chores, we must equip them with the knowledge and systems to manage this knowledge. This chapter is about using *Knowledge Bases (KBs)* to represent, combine and share knowledge between machines and how robots can act on that knowledge. To represent the required knowledge, a robot first needs to understand the meaning of the type of knowledge. It is essential to define how this representation can be processed and retrieved; see Section 4.1.1.

The ideas from Tulving [2007] about the *hypothesis of indifference* to cognition, behavior, and experience hold that their relationship depends on what they are and the particular circumstances under which they occur.

This chapter discusses concepts about knowledge, its representation, levels of representation and commonsense knowledge. This chapter first introduces definitions from different sources, including roboticists, about knowledge and its types. Specifically, Section 4.1 presents general concepts regarding knowledge and its representation and the available knowledge bases. The purpose of this introduction about knowledge, its types and representation is to provide a background to build the cognitive architecture implemented in this work. This cognitive architecture uses knowledge acquisition and representation to provide tools to a PSR for performing complex manipulations. In Section 4.2, knowledge representation and update are presented for the different types of memory (see Chapter 6).

4.1 Concept of knowledge

There might be confusion about the difference between *information* and *knowledge* as different definitions exist. In a meeting with 57 leading scholars from 16 countries, it was claimed that data, information and knowledge have a sequential order [Zins, 2007], which is used in this work. In the case of the concept of *data*, it is taken as the raw material of information by some people, but in other cases not. For this reason that the concept of raw data appears as source data or atomic data, which has yet to be processed for use, meaning no interpretation is made. When this data is processed, information emerges and has a meaning and purpose. In the same way as data, processed information produces knowledge. Information and knowledge different because the second includes a semantic value. For this reason, it has a higher level in the hierarchy. In the case of humans, knowledge gives them the capacity to understand and explain concepts, actions and intentions.

Other concepts of knowledge used in this work consider the philosophical definition presented by Hobbes [1651] as the evidence of truth, which must have concepts identified by a name used to create concluding propositions. One example is cognitive capabilities, in which ideas picture objects with mental properties. Furthermore, this concept was complemented by the Language of Thought Hypothesis (LOTH) [Fodors, 1975, p. 214]. In this hypothesis, thought representation and validation are supported by the principles of symbolic logic and computability; this means that reasoning can be formalized into symbols or patterns.

In general terms, the implementation made for this work considers that knowledge is composed of basic units called concepts [Ramírez and Valdes, 2012, p. 46]. These concepts are associated with other concepts that consider type, directionality and name. Associations and concepts are built dynamically and become stable through time.

A *concept* represented of a mental object with attributes expressed through a specific language and represented through computable symbols or patterns [Ramírez and Valdes, 2012, p. 53]. The Classic Theory of Concept Representation [Osherson and Smith, 1981] also considers the descriptive capabilities of concepts. Furthermore, *context* is considered to embody semantic knowledge [Ramírez and Valdes, 2012, p. 62]. In other words, it is how groups of concepts are associated. When combining several domains into a mixed context, a context may have contradictory knowledge. In this case, the framework uses a flexible low, restriction model included in the *Ontology Web Language (OWL)* [McGuinness and van Harmelen, 2004]. OWL establishes in its first two levels mainly treelike structures. In the implementations, there are limited searches, which do not compromise the model's flexibility.

According to Collins [1995], commonsense knowledge in humans is inside the body. This means that the way we cut up the physical world around us is a function of the shape of our bodies. For this reason, this type of knowledge, abilities or skills cannot be transferred simply by passing signals between the biological brain and the computer. Then the knowledge from human examples has to be processed, represented and reasoned. Furthermore, the same behavior may be the same in many action executions.

In computer science, it is accepted that knowledge can be processed, which includes storage, change and use by the computer [Tyugu and Tyugu, 2007, p. 6]. One definition says that knowledge is the content of data for a user who understands that data.

For a machine to handle knowledge, a knowledge system is required to represent and make inferences [Ramírez and Valdes, 2012]. This is part of the function of the cognitive framework implemented in this work.

4.1.1 Knowledge representation

Knowledge is available in the environment and the interaction between agents, e.g., humans and robots. One kind of interaction is natural language. However, extracting knowledge can be challenging as it is typically semi-structured or unstructured or needs more information

[Chen et al., 2012]. Another type of interaction between an agent and objects has been recorded in videos where people perform a task, i.e., YouTube. These videos present the problem of having only a specific viewpoint; further, occlusions exist between objects during manipulation. Solving these issues related to the computer vision area is not the focus of this work. For this reason that we use human examples from virtual environments in this work; see Chapter 5.

New knowledge can be acquired by associating previous knowledge with new one [Ramírez and Valdes, 2012, p. 64]. After knowledge is represented from the human examples, new representation units are integrated with the main already known one by following a constructivist principle. This means knowledge is constructed upon more knowledge and retained in a *Long Term Memory (LTM)*. This principle is used in this work.

As mentioned before, *Knowledge Representation (KR)* is required. In the human brain, knowledge is represented inside mental models that structure and organize that knowledge by describing, explaining and predicting its purpose, form, function or state [Phillips et al., 2016]. These models define how people interact with their environment and are continuously modified and updated as new information is acquired, mainly unknown information. There are two theories about how the human brain represents knowledge [Handjaras et al., 2016]. The first is the *modality-specificity theory*, which suggests that concept acquisition is modulated by low-level sensory inputs, e.g., visual, auditory or tactile, combined with motor functions. The second is called *domain-specificity theory* and suggests that knowledge may have an abstract organization of semantic attributes independently of sensorimotor processing. Even when these theories do not agree on the level of abstraction, both agree that knowledge is organized into semantic categories. However, the question of how the organization of conceptual knowledge is scaled in the human brain remains unsolved. Altogether, these results confirmed that category-based information is a dominant component in the definition of concepts of both groups. In the case of domain concepts, they are represented categorically in semantic memory within dedicated neural substrates [Vigliocco et al., 2002].

Knowledge Representation (KR) was initially derived from *Artificial Intelligence (AI)* to represent knowledge symbolically and manipulate it in an automated way by reasoning programs [Paulius and Sun, 2019]. This definition was extended for robots as how they represent knowledge about actions and their environment and enable them to include semantic concepts to their internal components for solving tasks, allowing them to reason and infer. The types of KR models are strongly linked to the kinds of knowledge, such as distributed, symbolic, non-symbolic, declarative, probabilistic and rule-based, among others. Each is suited for a particular type of reasoning, such as inductive, deductive, analogy, abduction, etc. [Russell and Norvig, 2003]. A representational view of the mind is presented by Clark and Grush [1999], in which intelligence is referred to as the *problem-solving computation* of internal representations of real-world structures, facts and hypotheses. In this work, internal states are seen as representations and related to the system's structure and the task performed.

In computational systems, knowledge is commonly represented in three ways. The first one represents knowledge as *symbols*, such as labels, strings of characters, frames, etc. Those symbols

can be manipulated using a predefined instruction set, for example, `if-then` representing the known facts about the world [Kotseruba and Tsotsos, 2018]. The disadvantage of this type of representation is that it needs to deal with a changing environment flexibly and robustly. Another area of improvement is the requirement of an initial knowledge base, which can take a long time to be created. The second type of representation is *sub-symbolic*, which is generally associated with the metaphor of a neuron. In this case, the knowledge is represented as numerical patterns and distributed in neuron-like objects. And the third type of representation combines elements from both and is called *hybrid* representation.

In the computational area of AI, there are mainly five types of knowledge representation, as shown in Figure 4.1. Structural Knowledge represents relations between objects and concepts. Declarative Knowledge represents object facts. Heuristic Knowledge represents the rules of thumb. Meta Knowlege represents knowledge about knowledge. Procedural Knowledge represents procedures as rules that include conditions.

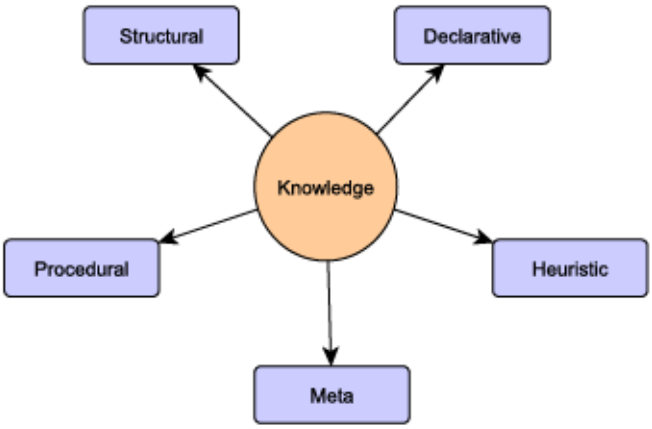


Figure 4.1: Knowledge types in AI.

Knowledge Representation (KR) is reducible (can be transformed) to knowledge objects. These objects can be formulas or texts in a suitable language or by a state of some system [Tyugu and Tyugu, 2007, p. 6]. There are commonsense theories of the world in which objects have properties called *measures* [Russell and Norvig, 2003, p. 329], including height, mass, cost and so on. One type is an *intrinsic* property, which represents the object's substance rather than it as a whole. For example, when an object is cut in half, both new pieces keep the same set of intrinsic properties like density, boiling point, flavor, color, ownership, etc. Another type is an *extrinsic* property, which is not kept when the object is divided, like weight, length, shape, function, etc. In KR, the object's category definition is represented using a class of only intrinsic or extrinsic properties. For example, the weight and volume of that object.

It is also essential to represent actions. Not only single actions but also help to represent and reason about action sequences [Russell and Norvig, 2003, p. 330]. Action representation describes conceptual representations of events in terms of schemata, which are structured knowledge representations of types of things and events one has encountered in the past [Zacks et al., 2009]. Schemata represent typical feature values for an entity type and relations amongst

those features. Activating a schema provides information about what objects will likely be present, what steps will probably be performed, and in which order. If we want a robotic platform to complete everyday chores, we have to equip this system with vast knowledge and systems that can manage this knowledge.

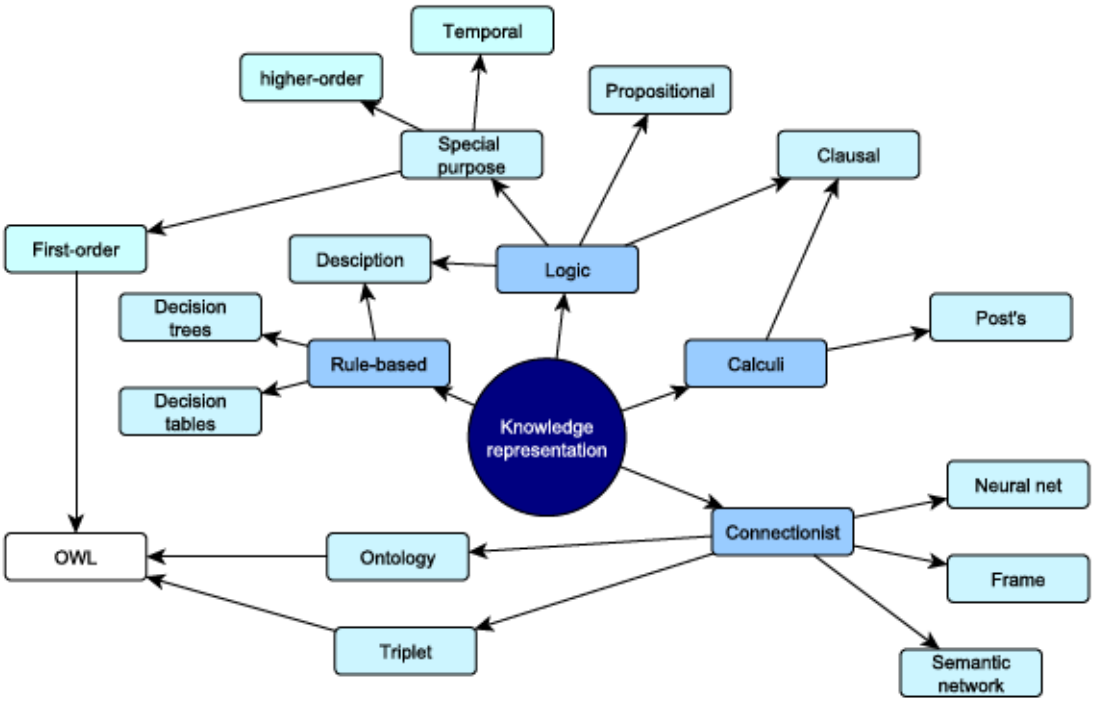


Figure 4.2: Knowledge representation methods.

To represent knowledge, some methods are used for the different types of KR presented in Figure 4.1. They are shown in Figure 4.2. The main classes are *rules-based*, *calculi*, *logic* and *connectionist*. All of them are introduced next. It is crucial to notice that some of them are connected. For example, description connects rule-based and logic, and clausal connects calculi and logic.

Rule-based

Rule-based systems use symbolic representation models focused on *procedural knowledge*. They are usually organized as a library of rules in the form of *condition-action* [Ramírez and Valdes, 2012, p. 50], e.g., if a sub-action is found, *stop*; *else*, keep looking. They are mainly used for representing skills, learning and solving problems, especially when procedural knowledge is present [Newell, 1982].

A robot can use causal rules to represent actions and consequences. Rule systems might also be used for declarative knowledge generally with classification purposes, e.g., if an object uses electrical energy, it is an electrical device. One example of using a rule-based representation is presented by Paulius and Sun [2019].

There is a compact way to represent a small number of production rules called *decision tables*. A decision table requires three kinds of subtables, a table of conditions, a selection matrix and a table of values. The table of conditions includes atomic conditions (predicates) for selecting production rules. The selection matrix combines them into complete conditions. Finally, the table of values contains the results of a selection that can be either values or actions.

On the other hand, a *decision tree* is a simple way to represent knowledge for decision-making [Tyugu and Tyugu, 2007, p. 127]. It is a tree with nodes marked by attributes, attribute values or decisions. An attribute always marks its root. Each path starting from the root passes through a node with a value of the attribute of the root and, after that, through the nodes marked by attributes and their values. It ends with a node labeled by a decision drawn from the values of the attributes met along the path. One example of its use in robotics is the work by Yang et al. [2015a], which represents action sequences as tree data structures a robot can execute. They represent a specific action by the items being used and the consequence of acting. Similar to the implementation made for this work, each action can also be broken down into smaller sub-actions or sub-activities. The second example is the work by Ryoo and Aggarwal [2009], where the representation describes human activities in a hierarchy divided by their temporal and spatial structure. In this case, the activity is decomposed into multiple sub-events and specified by its temporal, spatial and logical relationships. The definition of an event and sub-action differs in this work as they must clearly distinguish between an atomic action and a sub-event; here, it is called a sub-action because it implies various movements.

Logic

Logic is the primary vehicle for representing knowledge [Russell and Norvig, 2003, p. 4]. As this representation is always definite, each proposition is either true or false in the world. Logic has the advantage of being a simple example of representation. At the same time, it is general and handles formalism [Tyugu and Tyugu, 2007, p. 12]. However, it has some limitations discussed later with the specific types of logic. Logic must define the semantics of the language [Russell and Norvig, 2003, p. 200]. Semantics is associated with the *meaning* of sentences. In logic, the language's semantics defines each sentence's truth for each possible world. The logic language uses expressions to represent computations and denote objects that are called *terms*. Terms are built from variables and function symbols. *Atomic formulas* represent relations between objects. A collection of atomic formulas form *clauses*.

There are different types of logic. A very simple logic type is *propositional logic* [Russell and Norvig, 2003, p. 204]. The syntax of propositional logic defines the allowable sentences. The atomic sentences (indivisible syntactic elements) represent a single proposition symbol. Each symbol stands for a *proposition* that can be true or false. Complex sentences are constructed from more straightforward sentences using logical operators, such as negation, (not), conjunction (and), disjunction (or), exclusive disjunction (xor) and implication (if then). The semantics in propositional logic define the rules for determining the truth of a sentence for a particular model [Russell and Norvig, 2003, p. 206]. A model fixes the value-*true* or -*false* for every proposition

symbol. Unfortunately, there are better languages than propositional logic to concisely represent knowledge of complex environments [Russell and Norvig, 2003, p. 210].

Other examples are *special-purpose logic*, such as *temporal logic*, which assumes that facts hold at particular times and that those times (which may be points or intervals) are ordered. *First-order logic* is part of special-purpose logic and is expressive enough to represent commonsense knowledge [Russell and Norvig, 2003, p. 240]. *First-order logic* assumes that the world consists of objects with certain relations between each other that do or do not hold. The essential syntactic elements of first-order logic are the symbols for objects, relations and functions. The symbols can be of three types, constant (standing for objects), predicate (standing for relations) and function. First-order logic has functions STORE and FETCH to inform and interrogate a KB, the first one stores a sentence and the second returns all found sentences for a given rule [Russell and Norvig, 2003, p. 278]. The weight of each feature is the one corresponding to each first-order clause.

Thus, special-purpose logics give certain kinds of objects (and the axioms about them) "first-class" status within the logic rather than simply defining them within the knowledge base. Another one is *higher-order logic*, which allows assertions about all relations, including first-order logic sentences as objects in themselves. Unlike most special-purpose logic, higher-order logic is more expressive because any finite number of first-order logic sentences cannot express some sentences of higher-order logic.

Connectionist

According to connectionist theories, semantic knowledge representations have their roots in the description of interconnected concepts connected through associations [Ramírez and Valdes, 2012, p. 45]. These theories focus only on the presence or absence of associations and their quantity. This work uses mainly a connectionist approach.

A connectionist way of representation is a knowledge *triplet*, which can represent three kinds of knowledge [Tyugu and Tyugu, 2007, p. 33]. The first are facts, which are ground formulas whose meaning may vary. The second is patterns, which are, in essence, an atomic formula with variables. The last ones are rules, which include a condition and action parts.

On the other hand, linguists noticed that the structure of a sentence could be represented as a *semantic network* [Tyugu and Tyugu, 2007, p. 36]. The network represents the sentence's meaning in terms of the definition of words and the relations between the words. Words of the sentence are nodes bounded by arcs expressing relations between the words. This meaning approximates the meaning that people can assign to the sentence.

One example of visualization as a semantic network is used by the *Memory Map (MM)* system [Ramírez and Valdes, 2012, p. 56]. This map represents the interaction between concepts and skills in different contexts; for example, a concept's meaning may change according to the context (semantic environments). The implementation of the MM is a directed graph, very

similar to the more flexible types of semantic networks and ontologies, the last ones introduced later on.

Marvin Minsky postulated a hypothesis that knowledge could be represented in bundles which he called *frames* [Minsky, 1974]. A frame is a data structure for representing a stereotyped situation. They can be considered a type of semantic network which mixes declarative and procedural knowledge. Frames are capable of including procedures within each symbol, which makes them different from other networks. This means that each symbol contains a procedure called a *demon* and a group of attributes describing of the situation. Frames try to emulate human memory by storing situations that include procedural and declarative knowledge. Their properties were described informally [Tyugu and Tyugu, 2007, p. 38]. The first is a *concept*, a knowledge module about something such as a situation, an object, a phenomenon and a relation. *Frames* contain little knowledge, such as components, attributes and actions taken when defined conditions are met. These pieces may be concrete values of attributes, more complicated objects, or even other frames. A slot is filled when a frame is applied to represent a particular situation, object or phenomenon. An essential idea developed in connection with frames was inheritance. Inheritance is a convenient way of reusing existing knowledge in describing new knowledge.

There are other ways of knowledge representation, for instance, an *Artificial Neural Network (ANN)* [Tyugu and Tyugu, 2007, p. 37]. Knowledge in an ANN is partially encoded in the structure of the net and partially in the weights of connections. For more details, refer to Section 2.3. One example of this representation is the work by Cruz et al. [2015], where the effect of actions is represented to detect failures and improve the robot's behavior. Affordances anticipate the impact of the ANN. Affordances are action possibilities between the agent and its environment, presenting relationships between the agent and the object's components.

Ontology

Knowledge is richly and explicitly interconnected in human memory rather than structured as a set of independent or only implicitly connected facts. This knowledge requires a language for defining objects and interpreting their meanings. This can be done by ontology. One definition of an ontology is a system of interrelated concepts used to present knowledge in some knowledge domain [Tyugu and Tyugu, 2007, p. 34]. Another definition relevant to AI considers an ontology to be an explicit specification of a concept, an abstract, simplified view of the world, i.e., what exists can be represented [Gruber, 1993]. Ontologies are not about truth or beauty; they are agreements made in a social context to understand and accomplish some objectives and be guided by them [Gruber, 2003]. Aristotle defined an ontology in philosophy as the study of attributes that belong to things because of their nature [Guarino et al., 2009]. In experimental sciences, ontology focuses on the nature and structure of items independently of further considerations and their actual existence. Their nature and structure can be described in general categories and relations. In the case of Computer Science, an ontology is described as a special kind of information object or computational artifact. Computational ontologies formally model the structure of a system, i.e., the relevant entities and relations that emerge from its

observation, which is used in this work. The ontology engineer analyzes relevant entities and organizes them into concepts and relations represented by unary and binary predicates.

The purpose of an ontology is then to define a scope of concepts and terms used to label and describe the robot's working space in a format that humans also understand. An ontology typically provides a vocabulary to describe a domain of interest and a formal specification of the meaning of the terms in that vocabulary [Euzenat and Shvaiko, 2007]. An ontology, or part of it, could be visualized in a graph form. However, an ontology can capture very complex relationships between categories and individuals that graphs cannot [Paulius and Sun, 2019]. For example, being a subconcept, being a part of, being a property of or being a value of a property. The ontology organizes everything in the world into a hierarchy of *categories* or *classes*. The organization of objects into categories is a vital part of KR [Russell and Norvig, 2003, p. 322]. This is because even though interaction with the world occurs at the level of individual objects, much reasoning occurs at the level of categories. Categories also help to make predictions about objects once they are classified. It represents categories in two ways, predicates and objects. A *category* or *class* is a set of members or a more complex object with *Member* and *Subset* relations defined. Categories serve to organize and simplify a *Knowledge Base (KB)*, presented in the next section, through inheritance. For example, if we consider all instances of the category `Food` are edible and we assert that `Fruit` is a subclass of `Food`, and `Apples` is a subclass of `Fruit`, then we know that every apple is edible. In this case, the individual `Apples` inherit the property of edibility, in this case from their membership in the `Food` category. Subclass relations organize categories into a *taxonomy* or taxonomic hierarchy. The theory of intelligent reasoning is based on insights about human cognition and the organization of knowledge in memory comes from Minsky's frame idea [Minsky, 1974], which is primarily an ontological commitment. This commitment is a view of the world in terms of stereotypical descriptions, e.g., concepts are described in terms of what is typically true about them.

Ontologies can be written down in various languages and notations; their content is more important than their structure, which is a set of concepts about the world. Ontologies are commonly used in developing web services and a semantic web-based on knowledge about services [Tyugu and Tyugu, 2007, p. 45]. Ontologies can express formal inference rules. When the system's actions are consistent with the ontology's rules, it makes an *ontological commitment*. Ontologies use *triplets* of the type subject-predicate-object, e.g., `Apple` is `subClassOf` `Fruit`. There already exist some standards to create ontologies. The *Resource Description Framework (RDF)* standard has a vocabulary for reifying triples, which is the process of making a subject-predicate-object statement into a subject [Segaran et al., 2009, p. 134]. Another standard is the *Ontology Web Language (OWL)*, which is an RDF language developed by the *World Wide Web Consortium (W3C)* for defining categories and properties [McGuinness and van Harmelen, 2004]. They can also enable more powerful reasoning and inference over relationship categories. OWL is the current W3C standard for defining semantic web schemes. OWL is also a subset of first-order logic.

In this work, OWL and KNOWROB are used for building KBs. As mentioned before in Section 3.2, KNOWROB provides an action-centered KB [Tenorth et al., 2010a, 2014, Tenorth and Beetz, 2017, Beetz et al., 2018], which integrates various types of knowledge (static encyclopedic, commonsense, task descriptions, environment models, object information, observed actions, etc.) from different sources (manually axiomatized, derived from observations or imported from the web). All representations are combined with semantic properties. The OWL descriptions are downloaded from the database and parsed by KNOWROB's knowledge processing engine. Tenorth et al. [2013] provide a representation language to describe actions and their parameters, object poses in the environment, and object recognition models. It can also access meta-information about the exchanged data, e.g., types, file formats, units of measure, coordinate frames, self-models of a robot's components and capability configuration. Additionally, the RoboEarth language [Tenorth et al., 2013] is designed to describe task specifications for service robots from a high-level view (i.e., without considering hardware or environment details that are not interesting for the task at hand). The concepts can represent either basic sub-actions, e.g., navigation or grasping, or other task descriptions. KNOWROB includes the linguistic KB WorldNet [Miller, 1995], which provides a dictionary of more than 117,000 concepts.

Other systems also use KNOWROB. One example is the *automated probabilistic model of everyday activities (AM-EvA)* [Beetz et al., 2010a], which analyzes daily activities and represents human actions in raw poses, motion trajectories and activities in a symbolic relational knowledge base. Similar to this work, it uses the KNOWROB framework to define the hierarchy in the levels of abstraction of the actions of the observed sequences of motion. However, it differs in the exclusive use of motion sequences and not other features, such as acceleration and distancing between body parts and objects. A second example is the work by Ramírez Amaro [2014], which uses KNOWROB's ontology to enhance a *human activity recognition (HAR)* system to identify human motions, focusing on the hands. They analyzed activities such as *reach*, *cut*, *unwrap*, *take*, *idle motion*, *put* something somewhere, *release*, *spread* and *sprinkle* from a sandwich-making dataset. It is done by finding the tools used, e.g., knife or spoon, with their corresponding class and then infers the human activity associated with that class. In a further implementation, robotic platforms use human examples from virtual environments to perform everyday activities [Ramirez-Amaro et al., 2014]. Similarly to this work, the action classes are retrieved from KNOWROB. However, the knowledge goes through a preprocessing phase in this work, and the final result is obtained.

On the other hand, the framework presented by Chen et al. [2012] is similar to KNOWROB. The difference is that this one is open-knowledge-centric and focuses on automatically acquiring and utilizing open knowledge for online planning. This approach also includes different inference techniques depending on the type of knowledge.

Similar approaches to KNOWROB are presented in Table 4.1. The comparison is made based on the language used, in this case, OWL or XML. The table also considers if they are used in robotics or not.

Table 4.1: Comparison between Knowledge Base approaches.

Name	Representation	Used in PSR
KNOWROB	RDF and OWL	Yes
OpenRobots Ontology (ORO)	RDF and OWL	Yes
Hierarchical Task Network (HTN)	OWL	Yes
cognitive knowledge base (CKB)	Triple	No

There are three approaches applied to any Personal Service Robot (PSR), Hierarchical Task Network (HTN) [Di Marco et al., 2013], *OpenRobots Ontology (ORO)* [Lemaignan et al., 2010] and KNOWROB. The three are very similar in technology, representing knowledge in first-order logic formalism as RDF triples and OWL. This is because they were built in collaboration. As the cognitive knowledge base (CKB) [Wang, 2015] has not been applied to PSRs; to my knowledge, it is not used in this work.

One issue found in the use of ontologies is that there is no universal one and different specific ontologies have been difficult to use together [Ramírez and Valdes, 2012, p. 52]. However, this work uses some ontologies and overcomes this issue by comparing the classes of each of them. If there is any contradiction, it is excluded and put on a list for manual verification by an expert.

The limitation of connectionist methods is that they cannot explain higher cognitive processes and other types of knowledge [Ramírez and Valdes, 2012, p. 45]. Constructivist theories can overcome this issue with more complex reasoning, such as causality, probability and context. This is why the same is applied in this implementation in a verification process. This is done by integrating various layers with a group of associations. The difference between each layer is the strength of the associations. In this case, the top layer has the concept of an organized structure and the lowest has ideas.

4.1.2 Knowledge retrieval and reasoning

After acquiring and representing knowledge, it is crucial to access it and decide if it should be kept as it is or modified. The use of queries can do knowledge retrieval. A query is seen as an operation able to return information about a theory without modifying it [Darwiche and Marquis, 2002]. For knowledge management, knowledge can be transformed into a processed theory, which can then be operated with queries. One example of transformation is forgetting, which is an important tool to take into account.

KR provides descriptions with different abstraction levels and from various sources and assigns meaning to those descriptions allowing their combination to perform inference [Tenorth et al., 2010a]. KR enables an agent to determine consequences by reasoning about the world before acting on it [Davis et al., 1993]. All representations are imperfect and that imperfection can be a source of error. To overcome those errors, the reasoning is required. In this sense, reasoning is a process that goes on internally about things in the world.

In the action domain, category boundaries could be better defined. However, semantic features describe the inherent characteristics of activities [Ziaeefard and Bergevin, 2015]. The human body (pose), attributes, related objects and scene context are those features. For this work, it is essential to acquire and represent knowledge about actions in an extended manner. How actions are performed in detail corresponds to the commonsense that humans develop since infancy while interacting with objects inside the environment. For example, some of a physician's knowledge is in the form of rules learned from textbooks and teachers, which are patterns of association that he may not be able to describe consciously. So, this kind of knowledge is not expressed explicitly at the level of detail a robotic platform requires, and even more, the system should be able to develop representations after receiving these inputs to build its concepts.

In this case, reasoning intelligently means reasoning in the fashion defined by first-order logic [Davis et al., 1993]. First-order logic and its subsets ignore that most knowledge is uncertain, severely limiting its applicability. However, KNOWROB [Tenorth and Beetz, 2013] can overcome this issue by considering an open-and-closed-world assumption. The closed-world assumption states that everything that is not known is false. On the other hand, the open-world assumption states that everything that is not explicitly known is considered valid. With this combination, an object can be described in various ways. For example, when the robot has to act and some component is missing, it can simply check whether all required, known components are available and decline otherwise.

4.2 Knowledge system for a Personal Service Robot

As mentioned before, knowledge is essential in robots. For example, robots need the knowledge to reason about the world and make decisions based on their information. This includes knowledge about their environment, its objects, and the tasks they are requested to perform. Knowledge also allows robots to adapt to new situations and learn from experience. For example, they can use their knowledge to recognize patterns and make predictions, allowing them to improve their performance over time. Furthermore, robots that have knowledge can operate autonomously without constant supervision or control. They can use their knowledge to plan their actions and make decisions independently, essential for tasks that are too dangerous or complex for humans to perform. Also, robots can perform tasks more efficiently and accurately by using their knowledge to optimize their actions, avoiding mistakes and minimizing the time and resources required to complete a task.

Overall, knowledge is essential for robots to operate effectively in the real world. With knowledge, robots can reason, learn, adapt, and interact with humans and other robots. By incorporating knowledge into their design and operation, robots can become more intelligent, capable, and, ultimately, more valuable to society.

One way to store knowledge in robots is in memory, used in this thesis work. Chapter 6 discusses the use of different memory types. Memory requires a mixture of concepts and associations that

can be added and modified. For a Personal Service Robot (PSR) to handle them, a knowledge system must represent and make the inference. One significant element of this system is the *Knowledge Base (KB)* that stores such knowledge. In this section, different KBs are explained. These KBs are *Procedural Knowledge Base (PKB)*, *Episodic Knowledge Base (EKB)* and *Semantic Knowledge Base (SKB)*.

The *Procedural Knowledge Base (PKB)* stores knowledge about the execution of actions, which includes their structure, movement patterns, trajectories, etc. The *Episodic Knowledge Base (EKB)* stores knowledge about the occurrence of events, comparison of previous actions, sensory information and executed plans, among others. The *Semantic Knowledge Base (SKB)* includes KNOWROB's concepts and adds further ones presented next. All the KBs are represented hierarchically as ontologies.

Knowledge is not seen as simply a group of concepts but as an association's structure. These associations include information on the relation's nature, enabling more complex reasoning processes. The associations have the domain where they are valid. An association can be defined as a relationship between two representation units, such as concepts.

4.2.1 Action representation by a Personal Service Robot

Action representation is essential in this work. This is why actions are represented in all KBs mentioned before. In general terms, people have an organized structure of knowledge about various actions [Vallacher and Wegner, 1987]. This cognitive representation allows them to remember and identify actions, and their features and relations to others. The relationship among actions in an organized structure captures the person's general knowledge of how to execute an action. An action can be decomposed into several distinct `Sub-actions`; see Figure 4.3.

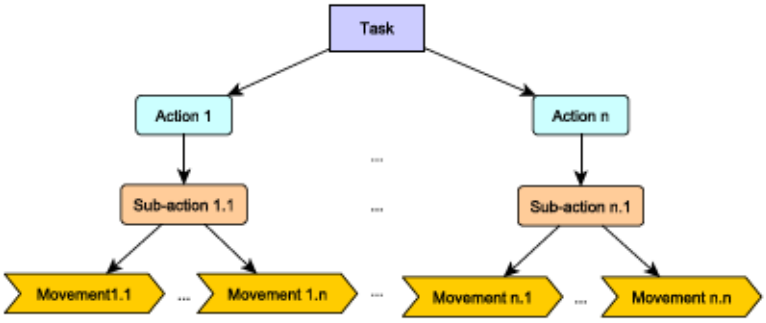


Figure 4.3: Task tree with levels of representation for tasks, actions, sub-actions and movements.

At a basic level, the sub-action consists of discrete movements of one's hands, fingers, eyes, and perhaps lower limbs and the entire body. These movements involve increasing or decreasing the angle's joints [Saladin, 2004], i.e., flexion and extension. The bending of the head, elbows, shoulders, knees, ankles or spine is flexion, while any posterior-going movement is an extension. These movements are performed at the shoulder, hip, elbow, knee and wrist joints. Finally, the

action as a whole can be seen as integrating these specific sub-actions. With more complex involved actions, even more levels become distinguishable.

This lower level in the representation is more concrete, providing depictions closer to the physical substrate of behavior. This level provides a recipe for how an action can be done, whereas higher levels are more comprehensive than this one and are more detailed. It is possible to represent one's action in terms of its components, such as movements and structure of actions and sub-actions. This is why such knowledge is included in the *Semantic Knowledge Base (SKB)*. Other components related to performance, e.g., success and repetitive behavior, are included in the *Episodic Knowledge Base (EKB)*. Both KBs provide flexibility and generalization for defining actions and sub-actions, which is used for planning.

In the case of actions, they can also be combined into different levels of abstraction until they form a task. In this case, ontology represents the level of abstraction as the level in the hierarchy. The actions in the ontology are defined as sequences of graphs. The nodes in these graphs can be objects, while the edges show properties of a touching relationship between a pair of objects to connect them.

Table 4.2: Manual and locomotion verbs.

acquire	chill	defrost	grate	locate	pill	scramble	stay
add	choose	dice	grease	lock	pinch	seal	steam
apply	chop	dilute	grill	lose	poach	search	stew
approach	clear	dip	**grip**	lower	pour	season	**stir**
arrange	close	dissolve	grind	make	press	select	stop
assist	coat	drain	grow	manipulate	produce	**separate**	strain
attach	collect	drop	halve	marinate	**pull**	serve	stuff
bake	combine	dry	heat	measure	puree	shake	**say**
beat	compare	empty	help	melt	**push**	sharpen	take
blanch	connect	fetch	**hit**	mill	put	simmer	thicken
blend	control	fill	hold	mince	**reach**	sleeve	**tilt**
boil	cook	find	incorporate	mix	regulate	slice	touch
bone	count	fit	increase	mount	replace	smoke	trim
break	cover	**flip**	introduce	**move**	**release**	soak	**throw**
bring	crack	fold	join	obtain	remove	spill	transfer
broil	create	form	knead	operate	**retract**	spray	**turn**
burn	crush	fry	layer	order	return	spread	use
carry	curdle	get	leave	peel	rinse	sprinkle	**wait**
carve	cut	give	let	**perceive**	roast	squeeze	wash
clean	collide	glaze	level	perform	roll	stand	waste
change	decorate	go	liquidize	pick	rotate	start	**whisk**

This work focuses on action verbs; for this reason, it considers action words or verbs from different sources related to cooking. For this reason, I compiled and filtered various sources of action names into Table 4.2, which includes only cooking-related actions. This table shows sub-actions in **bold** letters to differentiate them from actions. The first source is the Natural Semantic Metalanguage (NSM), which contains 53 *semantic primes* or words that can be identified in all world languages [Peeters, 2006, p. 13]. Semantic primes are the words with the most fundamental meaning in a language. These primes include only 14 verbs. The second source is the *semantic molecules* from Minimal English, which has 250 simple cross-translatable words of a highly reduced version of this language [Goddard, 2018, p. 8]. Semantic molecules are words strongly linked to the primes for building complex concepts [Goddard, 2018, p. 16]. This adds 12 verbs. A third source is the nonuniversal but useful words of Minimal English, which are related to things that matter to people. They add eight verbs. The rest comes from the manual verbs study by Gijssels and Casasanto [2020]. Their result shows the use of one or two hands while executing actions. The resulting manual and locomotion verbs are shown in Table 4.2. They represent the actions and sub-actions (bold) present in the KBs. As presented in previous chapters, all actions are broken down into these sub-actions. KNOWROB already included 25 actions, and 123 were added to the *Semantic Knowledge Base (SKB)*. From the sub-actions, ten were added, and eleven were already present in the KB. It is important to note that all actions can be broken down into sub-actions, both presented in Table 4.2.

Other representations from actions and sub-actions are episodes. Episodes can be from human demonstrations or robot executions. In the case of human examples, they include positions in time for the hands (end effectors), head and objects inside the VR environment. All these positions are associated with sub-actions and one level above action. These positions form a trajectory, which the robot can test. One example is in Figure 4.4, where the trajectory of topping a pizza is presented. This trajectory shows the right-hand position recorded while the actions were performed.

In general, episodes serve as an initial source of knowledge for the KBs, except for the *Semantic Knowledge Base (SKB)*. In this Figure, the trajectory marks the starting point in the bowl containing the cheese to a spreading position for the cheese on the dough. The robot can use this example when it has not performed this task before.

Actions and sub-action representation in episodes

In a cognitive architecture, it is crucial storing action executions in episodes because it allows the system to learn from past experiences and use that knowledge to inform future decisions and actions. Cognitive architecture can learn from experience and improve by storing past episodes. For example, suppose a robot encounters a new situation similar to a past experience. In that case, it can use its stored episodes to make decisions and take actions that were successful in the past. Episodes can also help make better decisions and plan more effectively by analyzing past experiences and identifying patterns to predict what might happen in the future. This can help the system make better decisions and plan more effectively in complex, dynamic environments.

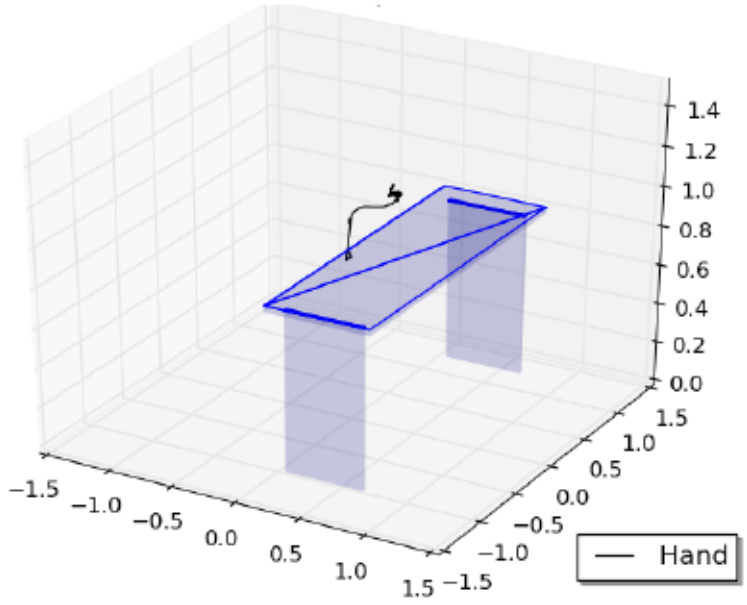


Figure 4.4: Trajectory extracted from a human demonstration for `putting` cheese on pizza dough.

Furthermore, episodes can also help adapt and be a more flexible response to environmental changes. For example, suppose a robot encounters a new situation, unlike anything it has encountered before. In that case, it can use its stored episodes to generate a plan of action based on past experiences.

In this thesis work, episodes store motions related to manipulation actions are described by patterns that include a set of constraints or properties between the tool used and the object in the world. These objects are not referred to specifically, but their generics, e.g., main axis, specific part. This information is grounded in the object used during the task execution. In KNOWROB [Tenorth et al., 2014] and this work, these constraints are represented as classes in an *Ontology Web Language (OWL)* file. OWL represents knowledge based on formal logic by providing a way to represent knowledge in a structured and machine-readable format. This language is used in this thesis work because of its rich expressiveness for complex concepts and relationships. It allows the representation of classes, properties, individuals and axioms, which can be used to capture the semantics of the manipulation action domain used in this work. It also allows automated reasoning and inference over the knowledge represented, which can help in tasks such as consistency checking, classification and query answering. Furthermore, this representation can integrate knowledge from different sources and domains.

The hierarchy followed in this work follows the idea that combined motion patterns build a sub-action, multiple sub-actions make an action and various actions make a task. This means all tasks are divided into three levels of hierarchy to sub-actions, translating to a sequence of movements.

To define if a sub-action was executed successfully, we follow the idea that actions can fail. In this case, minor variations in the choice of parameters can determine the success; for example, if a glass is securely grasped, falls out of the hand due to a too-low grip force, or gets broken by the

robot. Side effects can be inherent to the actions or caused by failures, e.g., collisions with other objects. These failures are detected by the event happening in the environment. For example, if the `Touching` event is absent, the object falls while transported. These events depend on the detection from the perception system, but they are also verified by the *Working Memory (WM)* depending on the action goals.

This work uses a realistic physical simulation, which gets parameterized with the robot's environmental knowledge. A detailed semantic robot model describes its size, kinematics, dynamics, and capabilities of actors and sensors. Plans can be executed within this simulated environment and changes in the world are logged with a God's eye view and translated into logical statements. A simulation is only an abstract model of reality and will not always produce the exact result. However, it will likely be much better than what logical inference on a limited, axiomatized model will yield. The simulations provide a cost-effective, low-risk and flexible way to test the manipulation actions to improve the robot's plans. In this case, the robot performs actions similar to humans; for example, using a similar sequence of actions, trajectories, or arm postures can make it easier to understand the outcomes and produce better plans.

4.2.2 Semantic Knowledge Base

The *Semantic Knowledge Base (SKB)* aims to capture the meaning of concepts and their relationships to other concepts. It does this by representing knowledge in a structured format designed to capture the meaning of concepts and the relationships between them. Its representation uses an ontology, a formal specification of the concepts and relationships in the service robot's domain. The ontology includes a hierarchy of concepts, with more general concepts at the top and more specific concepts at the bottom.

The system can extract entities and relationships from the ontology by identifying entities such as places and objects and associations such as *is a*, *part of*, *located in* and so on. This knowledge is extracted by querying for specific entities or relationships to infer new knowledge based on the existing knowledge in the knowledge base. Furthermore, the SKB can also be updated and refined over time by refining the ontology based on real-world data.

As mentioned, the SKB stores concepts about different areas, such as static encyclopedic, commonsense, task descriptions, environment models, object information, observed actions, etc. All representations are combined with semantic properties, such as classes they belong to and features, e.g., the *cup* belongs to the class *container* and has a *handle*. In the case of actions, they are represented by their associated sub-actions and their movements.

This SKB uses a *dynamic hierarchy*. This means that concepts can be modified and added and their associations to other concepts and properties. Each concept can belong to multiple classes. This KB also has *unlimited granularity*. As the concepts can belong to different classes, it forms a network structure where there can be an infinite number of levels. This also allows the structure to avoid redundancy, i.e., the concept is used for different knowledge structures.

As mentioned before, KNOWROB is used as a base of the SKB. Here, more concepts about specific objects found in the Epic kitchen 100 [Damen et al., 2018] are added to this implementation. This dataset is used because it is the most extensive collection to date in standard kitchens from different cities. Inside this dataset, there are 2032 nouns of 31 classes. These classes are explicit appliances, baked goods and grains, cleaning equipment and material, containers, cookware, crockery, cutlery, dairy and eggs, drinks, fruits and nuts, furniture, materials, meat and substitute, prepared food, rubbish, spices and herbs and sauces, storage, utensils, vegetables and others in the dataset. Some nouns belong to more than one category. Some other nouns refer to the same object; for example, there may be a commonly used brand name for a grocery object. For this reason, I performed data cleaning to add the corresponding objects to the correct object type instead of having a different value, e.g., `v60` is a type of `maker:coffee`. Other nouns are part of existing objects; in that case, they are kept as the interaction with them is important to consider, e.g., `door:microwave` and `oven:microwave`. All these nouns were transformed to the naming convention of KNOWROB, which means capitalization, removing the colon separating the name (`door:microwave` in Epic Kitchen syntax) and changing the word order when necessary (`MicrowaveDoor` in KNOWROB syntax). The Epic Kitchen 100 dataset provided 1750 new nouns to the SKB.

To add nouns to the SKB, each noun of the Epic Kitchen dataset was compared with the ones already present in KNOWROB. This was done with the check process described before in Chapter 6. Each noun is compared to an `OWLIndividual` to verify its existence in the SKB. This comparison is performed automatically by using the SM verification system. This function received the list of nouns and categories. If the noun does not exist, its category is obtained from the dataset and added to the KB. If the noun exists, its category is compared with the existing one and either a new connection is added, or kept as it is. Some categories added were *grains*, *crockery*, *dairy*, *fruits*, *nuts*, *meat*, *substitute*, *rubbish*, *spices*, *herbs*, *utensils*, *vegetables* and *cleaning material*. The categories of the nouns added are presented in Figure 4.5. Blue represents the nouns already present in KNOWROB, and orange represents the new additions. This process serves to include information from other datasets into the KB.

Verbs were also added to the SKB; in the same way as for nouns, verbs were compared with the ones already present inside the KB. The categories of the verbs added are *access*, *block*, *clean*, *distribute*, *leave*, *manipulate*, *merge*, *monitor*, *order*, *retrieve*, *sense*, *split* and *transition*. In total, 856 verbs were added to the KB, from which 723 came from the Epic Kitchen 100 dataset and 133 from Table 4.2 (actions and sub-actions), their type is presented in Figure 4.6. In the same way, as in a previous figure, verbs already shown in KNOWROB are presented in blue and new additions in orange.

Similarly to the verification process mentioned above, a verification is performed to add knowledge to the SKB. First, the robot generates a recording from the execution described in an OWL format. These are referred to as episodes. Then, some questions are asked about the execution, primarily related to the actions' structure and the success level. If the sub-actions were executed successfully, they are added to a temporal structure that is compared to the KB's existing one.

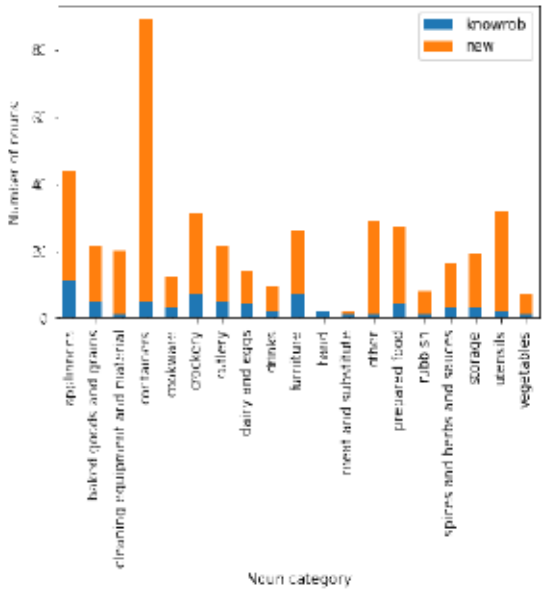


Figure 4.5: Epic Kitchen dataset noun additions to *Semantic Knowledge Base*.

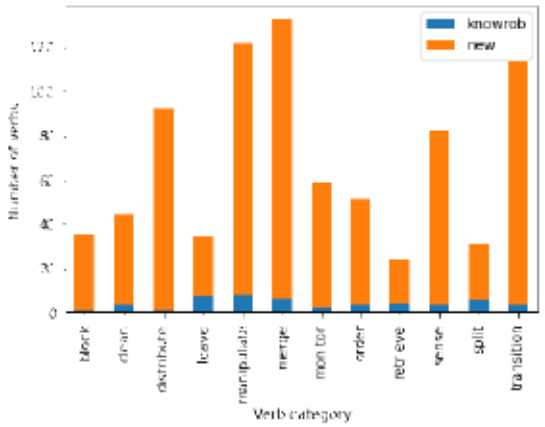


Figure 4.6: Action additions to *Semantic Knowledge Base*.

If the action-sub-action pair does not exist yet; it is added following the existing taxonomy structure. If the pair already exists, a comparative process takes place. If they are different, the robot has to execute the pair from the KB and then both executions are compared to select the one with the best results.

Even though not all the new nouns and verbs are used in this work, this extension serves for future applications. One example is the inclusion of various cleaning materials and equipment. The PSR requires them to be more useful cleaning tasks house environments. However, the focus of this work is cooking actions.

Overall, the SKB provides a way to represent and reason about knowledge in a structured format that captures the meaning of concepts and their relationships. This is useful for the domain of PSRs as they have to reason about complex relationships between entities.

4.2.3 Procedural Knowledge Base

The *Procedural Knowledge Base (PKB)* describes how to perform manipulation actions. It contains a set of procedures and heuristics used to perform a particular action. Each procedure specifies a set of actions and sub-actions that can be taken in response to a particular condition or event.

Procedural Memory (PM) is related to skills. It is important to note that skills depend on the executor's knowledge [Russell and Norvig, 2003, p. 240]. This means that how the system behaves depends on the knowledge it contains about how to execute actions. This is why procedural knowledge is required by a Personal Service Robot (PSR) to perform manipulation actions. As mentioned before, the *Procedural Knowledge Base (PKB)* stores knowledge about skills. Skills in this work include cognitive and psychomotor domains; as the machine does not have emotions yet, the affective domain is excluded [Bloom et al., 1971]. The cognitive domain includes interactions between concepts and other skills. Skills are process-oriented, related to actions and sub-actions commonly described using verbs, see Table 4.2. Figure 4.7 presents some examples of the verbs in the table as follows; the sub-action *release* is represented as `ReleasingGraspOfSomething`, *reach* as `Reaching`, *push* as `PushingAPartOfDevice`, *pull* as `PullingAPartOfDevice`, *retract* as `RetractingAnArm` and *grip* as `GrippingAPartOfDevice`. Sub-actions (orange) have features represented in the figure by lines of a different color for clarity. These features are `previousAction` (dashed and dark green), `nextAction` (continuous and light green), `positionInExecution` (continuous and yellow) to build the sequence in which actions and sub-actions are executed. In the case of the sub-action `Reaching`, it does not have a `previousAction` as it is in the first position in the execution of the action. Similarly, the sub-action `RetractingAnArm` does not have a `nextAction`, has is the last in the execution of the action. They also have features such as `trajectory` (dashed and red), `executionTime` (continuous and turquoise) that are present in all of them. Some of them have extra features such as `objectActedOn` (dashed and dark blue) and `graspType` (turquoise) that are specific for `PushingAPartOfDevice`, `PullingAPartOfDevice` and `GrippingAPartOfDevice`. To complement the knowledge stored in the PKB, *grasp postures* are added in the `graspType` (continuous and black) feature besides trajectories.

When a query is made to the PKB, the system searches for a set action and sub-action pairs that match the query criteria. This involves evaluating the conditions of each sub-action and selecting the one most appropriate for the current situation. Once a set of action and sub-action pairs has been chosen, the system sends them to the WM for the execution of those pairs specified to perform the task or solve the problem. This involves a sequence of steps or actions that must be performed in a specific order.

Besides, this work considers the motions and trajectories to represent the actions and sub-actions [Paulius and Sun, 2019] and details such as the semantic meaning. This meaning includes changes and consequences of sub-actions in the environment as an `effect`, which depends on perception for being detected.

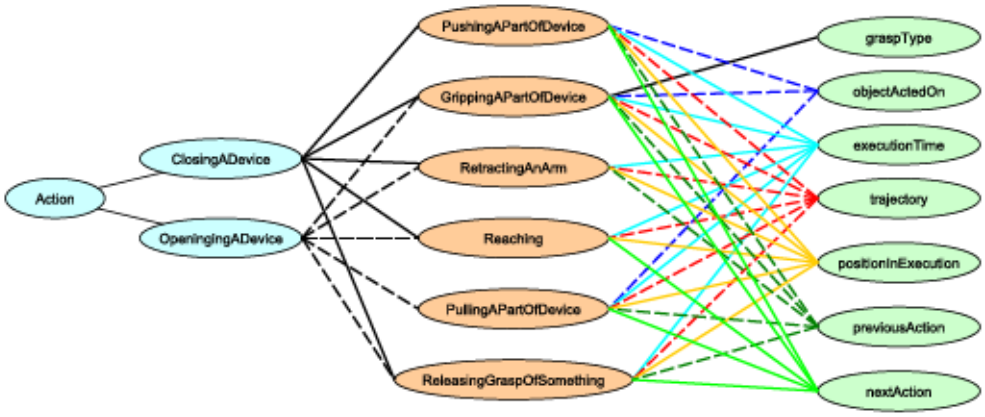


Figure 4.7: Simplified example of actions and sub-actions present in the *Procedural Knowledge Base*.

The associations present in this KB [Ramírez and Valdes, 2012, p. 59] include information on the nature of the relation, which enables more complex reasoning processes, and where it is valid. An association can be defined as a relationship between two representation units, such as concepts and skills.

The PKB is updated and refined over time by incorporating knowledge based on real-world data as a response for executions of the selected actions and sub-action pairs.

Overall, the PKB provides a way to represent and execute knowledge about performing a particular action or solving a specific problem. This is useful in this domain because the tasks and issues are well-defined and can be decomposed into a set of procedures.

4.2.4 Episodic Knowledge Base

The *Episodic Knowledge Base (EKB)* represents concepts of episodes and events, discrete experiences that can be recalled or retrieved from memory. The *Episodic Knowledge Base (EKB)*, as mentioned before, stores knowledge about the experience, in this case, recorded previous action executions. There is a difference compared to the episodes discussed before. In the EKB, information is organized around these episodes rather than around abstract concepts or categories.

When a task is executed, it is encoded into the EKB as a new episode. This involves capturing different aspects of the experience, such as the time, location, objects involved, and other contextual details.

In the case of episodes, they store execution information, which includes low-and-high-level data. They have times, positions, actions and sub-actions from more than the agent but also the environment. The EKB, on the other hand, stores a compilation of the episodes. This compilation includes analyzed information from the execution, such as the duration of execution, success, encountered issues, etc. In Figure 4.8, some of the elements of the EKB can be seen. The actions are represented in blue. The sub-actions have an orange color. There are *global features* that track the agent's performance, which are marked in dark green. For example,

the successLevel verifies the percentage of success of all successRate for each sub-action over time. The successRate is measured by the number of failures against successes and if those failures were solved. These features serve for future improvements. On the other hand, *local features* are related to one-time task execution (light blue) or one episode. For example, *timesExecuted* counts the executions of a specific sub-action inside an action, which can be part of failure solving.

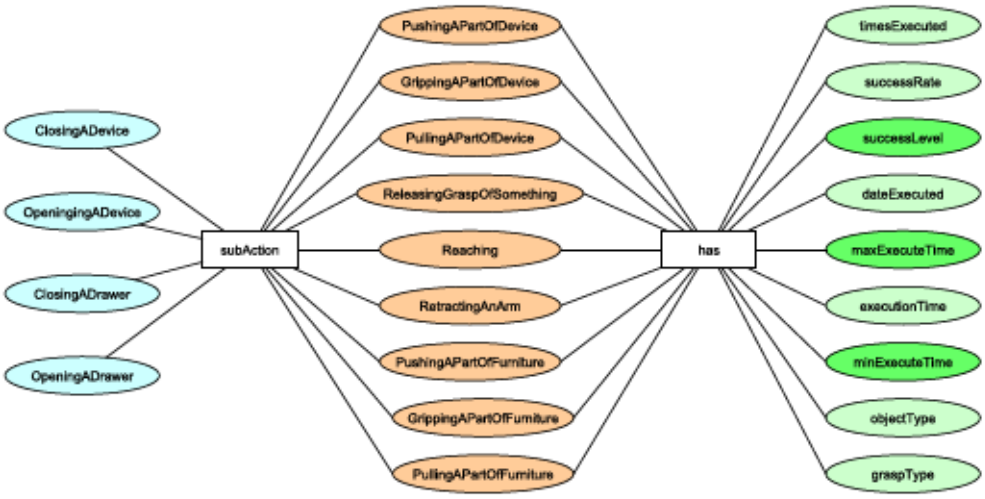


Figure 4.8: Simplified example of global and local features present in the *Episodic Knowledge Base*.

When a query is made to the EKB, the system searches for episodes that match the query criteria. For example, a query might ask for all episodes that occurred at a specific time and place or that involved a particular object. Furthermore, the system can also use associative links to connect episodes with common features or contexts. This helps to retrieve related episodes that might not match the query criteria directly but are still relevant to the task execution needs.

The EKB updates when new similar episodes are added. The system uses this information to refine its understanding of the context and improve its future recommendations. Verification is performed to add new elements to the EKB, similar to the process mentioned above for the *Semantic Knowledge Base (SKB)*. As mentioned above, the robot records its executions and stores them in episodes. Then some queries are asked about the execution, especially related to the level of success, repeatability, time of execution and intermediate steps. There are global and local features attached to actions and sub-actions. *Global features* allow the system to keep track of statistics of performance for further improvement, e.g., if a sub-action can only be executed a few times, and some additional analysis can be performed. *Local features* store information related to the specific execution that can add to other statistics and analyses. This analysis can be performed either by the system or by the programmer. That is how the *Episodic Memory (EM)* can know what happened at a specific time, as the day-time information is also stored. Even if a sub-action was not executed successfully, it is includes its success and the type of error encountered. If an action-sub-action pair does not exist, it is added following the existing taxonomy structure. If the pair already exists, it increases the global number of executed times and adds all the available features of the execution to the particular variables.

Overall, the EKB provides a way to organize and retrieve information based on discrete events or experiences rather than abstract concepts or categories. This is useful because context and personalization are essential for the PSRs' caregiving tasks.

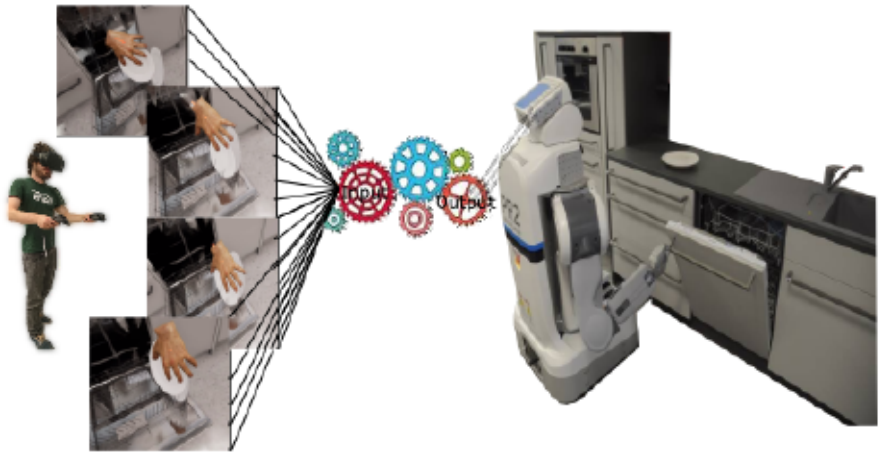
4.3 Summary of this chapter

This chapter presents concepts of knowledge relevant to this work. One of the main concepts present is *Knowledge Representation*. It includes a comprehensive introduction to the types in which knowledge can be represented in *Knowledge Bases (KBs)*, the main application used in this work. The KBs created in this work use ontologies, which are also introduced in this chapter.

This chapter presents the contents of the KBs created in this work. Specifically, the *Procedural Knowledge Base (PKB)* includes actions and sub-actions with their structure, movement patterns, trajectories, etc. The *Episodic Knowledge Base (EKB)* stores the occurrence of events, comparison actions and sub-actions and executed plans, among others. And finally, the *Semantic Knowledge Base (SKB)* builds on top of KNOWROB concepts. With these KBs, the robot's knowledge is increased for future executions.

Chapter 5

Human manipulation action recognition from virtual reality data



As mentioned in previous chapters, a Personal Service Robot (PSR) requires particular capabilities to perform tasks successfully in human environments, especially caregiving tasks. Today, high-performance robot arms are faster, more accurate and more potent than humans. However, human manipulation capabilities still need to be improved beyond the ones of such robots. The main reason is that humans' neural information processing and control mechanisms are much faster than those in robots.

Before PSRs can perform manipulations, they need to detect which action elements are necessary for successful execution, for example, a grasp type required to grab a full cup. In particular, it is interesting for this thesis work to identify the structure of manipulation actions related to cooking activities to introduce them inside an execution plan. Cooking actions are selected as they require various manipulation actions that PSRs must master and be considered suitable workers in human environments [Yang et al., 2015b]. In this sense, a robot should understand the interaction between perceptual, learning, reasoning, planning and control mechanisms well. The robot should select the tool to act on the desired object during manipulation. For example, the robot should manipulate a bottle differently depending on whether it intends to fill a glass or put it away. Inside the robot's control system, concepts should be defined regarding their roles in actions, experience and knowledge. These concepts could be acquired from the observation and performance of activities.

In the case of the observations, the area of imitation learning suggests observing humans. Therefore, this area is called, more specifically, *Programming by Demonstration (PbD)* or, from the robot's viewpoint, *Learning from Demonstration (LfD)*. In this thesis work, demonstrations are used as an initial source of knowledge for unknown actions. They are also significant examples of complex manipulations where the robot can not find a competent solution or can present a future problem in a human environment that the robot needs to be made aware of when acting. For example, the orientation of the pan's handle on the stove might seem insignificant from the robot's point of view. However, if the handle is placed towards other burners that are in use, it can get hot. This can damage the robot's gripper (depending on its material) or burn a human hand if touched directly. Similarly, the knife's orientation is vital to avoid injuries when passing it to a person.

One problem is that such observations come from action executions that, particularly manipulation actions, present a challenge because of the large number of variations in how they are performed [Yang et al., 2015b]. For example, as they have other goals, the desired *grasping type* could differ for a plate *transported* with food on top and a vessel from the dishwasher to its storage locations.

By taking this into account, this chapter presents first an overview of manipulation actions studied in humans and robots in Section 5.1. There, the definitions of actions and sub-actions used in all this thesis work are presented. It also offers state-of-the-art systems to classify actions.

The knowledge needed by PSRs can come from different sources. Since humans are experts in performing everyday tasks with the human body and adapting to changes, they are a reliable source of knowledge. This knowledge can come in different ways, such as language instruction, demonstrations in videos or *Virtual Reality (VR)*, etc. However, this knowledge must be represented in how the robot can understand. On the one hand, semantic knowledge about the environment is required to understand the robot's surroundings. On the other hand, specifics about how to perform actions and naive physics, known as commonsense knowledge, are also required when trying to perform such actions. One question is how to represent this knowledge. One example is to use *situations* like in the robot Golem [Pineda et al., 2013]. Other options can be *tasks* or *actions* used in this work. Another question is how much information should be included. Based on these questions, it is essential to explore how different robots represent the actions they perform. Some of the robots presented in Appendix Table A.2 use a hierarchy to include concepts about tasks and actions in an ontology, which is also used in this thesis work. An ontology organizes concepts from the world into a hierarchy of categories. A *category* is a member or a more complex object that has a relation to other members. For example, if we consider all members of the category `Food` are edible, and then we add `Fruit` as a subclass of `Food` and `Apple` as a subclass of `Fruit`, then we know that every `Apple` is edible. Subclass relations organize categories into a *taxonomy* or taxonomic hierarchy. The ontology serves to represent knowledge, which seems similar to *Semantic Memory (SM)* [Kotseruba and Tsotsos, 2018].

It is essential to give some definitions before entering the rest of the chapter. These definitions are related to perception motion [Bobick, 1997] and physiology [Saladin, 2004]. *Movements* are the most atomic primitives that can be defined as a type of motion. *Motion* is a definite trajectory in space and time. In the case of the kinematics of the human body, motion involves increasing or decreasing the angle of the joints.

An activity analyzes the statistical properties of the movement. Then, *activity* refers to sequences of movements or static states [Bobick, 1997]. It can be seen as a purposeful action or occupation.

Actions are larger-scale events, including environmental interaction and causal relationships [Bobick, 1997]. An action can be defined in terms of movements similar to an activity. However, an action includes semantically rich descriptions of causal relations with those movements. In other words, an action has semantic concepts related to the context of the motion. It requires an interpretation of the context, which can be seen as a set of constraints on possible explanations for the observed movements.

5.1 Manipulation actions

Cognition is believed to be interconnected to action [Rosenbaum et al., 2012], as human and animal object manipulation reflects their intentions. Even more, the way an object is manipulated indicates the understanding of the object's functionality and physical composition, e.g. the `Spoon` can be used for eating and may be picked up by its `Handle` rather than its `Bowl` because of sanitation or etiquette concerns. Therefore, the definition of an *end-state comfort effect* is given, in which a person manipulates an object in an initially awkward posture to get a better final one, in many cases.

Different types of planning are required to perform manipulation actions by considering planning as the steps needed to achieve a task. One of these types is *first-order*, which depends on immediate task demands. Another is *second-order*, which is modified depending on the next task to be performed. Most robots are capable of first-order planning. In the case of humans, planning abilities increase from age 3 to 10, but it is not clear when second-order planning appears. Still, children's planning abilities are less robust than in adulthood.

The correct sequence of actions in everyday life is usually learned or selected through experience. The idea of how to build plans from actions and sub-actions comes from the work by Johansson and Flanagan [2009]. This work mentions that manipulation tasks can be broken down into action phases delimited by the mechanical events representing subgoals. Those mechanical events have specific auditory and visual patterns. Such patterns are related to object properties and the actuator, the hand in this case. If an error occurs, the representation of the object's properties can update the plan for further execution.

But also, the skills required to perform manipulation must be transferred through social interaction rather than book learning [Collins, 1995]. In this thesis work, we use demonstrations and experience.

5.1.1 Types of actions

According to Aggarwal and Xia [2014] and depending on the environment, human actions have different forms ranging from simple to complex. These actions are conceptually categorized into *atomic*, single, and *complex*, containing a sequence and group of other atomic actions and interactions such as person-object and person-to-person. Many human actions are complex; two examples are assembling furniture or food preparation [Ziaeefard and Bergevin, 2015].

This thesis work focuses on atomic and complex serving and food preparation actions. Complex actions will be referred to simply as actions, while atomic actions will be referred to as sub-actions. In one example presented in the introduction, `Pouring` hot water into the cup requires sub-actions, such as `Reaching`, `Grasping`, `Lifting` and `Tilting` the water kettle.This division is made for representation and planning purposes, detailed in Chapter 4.

Another way of classifying an action is inside everyday actions, also known as *usual*, which connotes a *routine* [Collins, 1995]. Everyday actions follow the rules in their order of execution. We almost always know when we have broken those rules, for example, when we start mopping the floor before pouring water into the container to make the mop wet. This type of action is also attractive to this thesis work as they are repetitive in daily human life and are generally called chores.

5.1.2 Actions in real or virtual environments

Actions executed in natural or artificial environments depend on perceived opportunities, not on the environment [Rosenbaum et al., 2012]. This hints that actions recorded in virtual scenarios, such as *Virtual Reality (VR)*, might be a good way of obtaining data. Even further, virtual grasp conditions would map onto the same hand position and grasp type as in the real grasp conditions. This is because mental representations from grasp postures to be performed are available even before movement happens. So far, there is no objection to performing a specific manipulation action in virtual environments. However, using such data as a demonstration brings some challenges.

Even more, the *Virtual Reality (VR)* 3D *motion caption system (Mocap)* is an advantage against natural execution captured in a video. This advantage is the fact that the data collected would not need to deal with common vision issues while being processed [Aggarwal and Xia, 2014], such as the number of camera viewpoints. The *motion caption system (Mocap)* technique has mainly been used to animate computer graphic figures for motion pictures and video games, analyze sequential mechanics of athletes and monitor recovery progress during physical therapy. One disadvantage is that this technique records only specific point locations, e.g., where sensors are present. *human activity recognition (HAR)* and *activity recognition systems (ARS)* have emerged thanks to the advancement of this sensor technology. These sensors measure human movements and interactions while performing *activities of daily living (ADL)*, referred to here

as everyday actions. The *activity recognition systems (ARS)* should extract features based on joint positions and angles to use the Mocap technique.

As mentioned before, the method used to record human examples for this thesis work is the implementation by Haidu and Beetz [2016]. Their work is extended by adding more objects to the virtual environment, including their semantic properties into the knowledge base. This is because, even when the previously created environment includes various objects and properties, it still needs tools and everyday objects to perform cooking activities. Some examples of these tools are a hand mixer, pans and pots. Objects such as fruit and vegetables were also missing from that virtual environment and are added for this thesis work.

5.1.3 Action segmentation, classification and recognition

The control of object manipulation depends on sensorimotor mechanisms that exploit predicted and actual contact events [Flanagan et al., 2006]. To understand how we give meaning to the actions performed by others, there is a proposal called the *direct matching hypothesis* [Flanagan and Johansson, 2003]. It stipulates that action understanding is achieved by a mechanism that maps an observed action onto motor representations of that action.

Based on neuropsychological evidence [Ziaeefard and Bergevin, 2015], humans recognize both the movement (physical) and action (semantic) goals of individuals. Physical goals are related to the kinematics of specific movements, e.g., `Reaching` toward the left. Conversely, semantic goals are related to functional expectations that lead to movement execution, e.g., `Reaching` toward a glass, shown in Figure 5.1. Identification and description of activities and actions are referred to as *activity analysis*.

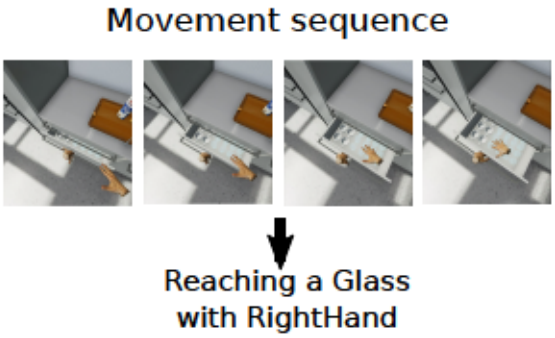


Figure 5.1: Example of `Reaching` sub-action for a Glass with the `RightHand`.

According to Zacks et al. [2009] and Zacks and Swallow [2007], to identify an action, it is required to know its body motion, intention and effects. In this case, event segmentation is viewed as breaking up continuous activities into meaningful events, a side effect of understanding and prediction in the *Event Segmentation Theory*. The human brain and mind can track and perceive changes in environmental features. This perception of change happens using sensory characteristics such as color, sound and movement and conceptual features such as cause-and-effect interactions and actor goals.

Then, various tests were performed in which people had to segment activities from videos [Zacks and Swallow, 2007]. These tests found that everyday event timescales go from just a few seconds to tens of minutes. For example, fine-grained events associated with specific actions on an object, see Section 5.1.1, take around 10–to 15s, and coarse-grained events related to action contexts, e.g., goals and causes, take about 40–to 60s. These times serve to verify the action segmentation performed in this thesis work and are presented in Section 5.2.1.

Traditional HAR methods rely primarily on tracking and motion capture [Ziaeefard and Bergevin, 2015]. Recent approaches use mid-level features such as bag-of-word (set of words) that include features of actions. The work by Ryoo and Aggarwal [2009] uses a similar approach by maintaining their representation of the temporal and spatial structure of the target activities. They represent a high-level human activity in terms of relationships between simpler activities. They see actions' and interactions' temporal relationship as very important to describe them, as they can be defined by a sequence of subevents using *Allen's interval temporal logic*, such as *before*, *meets*, *overlaps*, *starts*, *during* and *finishes*. This interval temporal logic representation [Allen and Ferguson, 1994] relates periods to the actions and events and their effects.

The work by Aggarwal and Xia [2014] mentions that the goal of *human activity recognition (HAR)* is to detect and analyze human activities from sensor information automatically. One source of data is *Red Green Blue Depth (RGB-D)*, which requires depth sensors together with 2D images. This type of HAR development began in the early 1980s. Nowadays, there are sensors capable of recording the execution of activities in virtual environments. There are three main types of sensors: accelerometers, magnetometers and gyroscopes. They measure the motion and direction of the object of the body part they are attached to in the space.

Nowadays, humans can also record demonstrations via teleoperation, e.g., the work by [Seo et al., 2023]. This means that the human directly controls the robot's movements and records them. That way, the robot learns the movements required with their parameters. However, this thesis work uses a different type of recording that does not require specific equipment mounted into the robot.

The approach implemented in this thesis work is also similar to the one presented by Ramírez Amaro [2014] and Diehl et al. [2021], as the activity observation and the intentions are detected one after the other. The difference with her approach is that the demonstrations stored in a hierarchy are transferred directly to the robot without an intermediary. On the other hand, in this thesis work, the *Working Memory (WM)* serves as an intermediary by receiving the command and returning the actions and sub-actions the robot requires to solve the task.

Some examples of these semantic features are the description of objects related to their parts (a cup's handle), shape (cylinder) and materials (ceramic). In this thesis work, semantic features are used for human action recognition, similar to the work of Ryoo and Aggarwal [2009] and Ziaeefard and Bergevin [2015]. The system matches semantical representations with actual observations. In this thesis work, semantic information is used first and then as a classification technique.

Another approach is to use the human pose to distinguish actions, which is also used in this thesis work. Pose-based techniques are more robust to intra-class variety than appearance-based methods used in computer vision. Intra-class variety is related to actions that seem very similar to other ones, for example, `Pull` and `Push`, where the difference is the direction of the movement. However, pose-based methods require selecting the sensors to measure specific attributes [Alpoim et al., 2019, De-La-Hoz-Franco et al., 2018]. The data coming from the sensors has to be processed in different stages, such as preprocessing, segmentation, feature extraction, dimensionality reduction and classification. As HAR systems processing the data offline provide generally better recognition performances, the system presented in this chapter works offline.

A very similar system to the one implemented in this thesis work is the Automated Probabilistic Models of Everyday Activities (AM-EvA), which perceives, interprets and analyses everyday manipulation tasks and activities of daily life [Beetz et al., 2010a]. The difference between this work is that *subevents* such as `PickingUpAnObject`, `Carrying` and `PuttingDownAnObject` related to the action `PuttingSomethingSomewhere` are named sub-actions and include more features.

A more recent and similar approach uses the same type of data and inspects it through openEASE [Kazhoyan et al., 2020a], which is a web interface for answering queries with a graphical visualization. There, the data is analyzed to segment the continuous data stream of observations into human action sequences. In this case, the context of the task is known beforehand. Furthermore, *Fuzzy Markov logic networks (FUZZY-MLN)* are used as a statistical model to infer the discrete grasping poses. In contrast, this thesis implementation can recognize the actions and sub-actions without knowing the context beforehand or prior analysis.

Even when there is success in HAR, some issues are handling intra-class variability and inter-class similarity of actions [De-La-Hoz-Franco et al., 2018]. This means that individuals perform the same action in different ways, including various body part movements, and two separate acts may be too similar in the spatiotemporal details. For this reason, this is still a complex problem for algorithms using various types of data. Also, in most cases, it is impossible to compare the different HAR approaches as there is not a standard dataset that allows the validation of many of these approaches, see Section 5.1.5.

To recognize actions, machine learning techniques can be applied. As mentioned in previous chapters, there are two main types of learning, supervised and unsupervised [Nilsson, 1996]. *Supervised learning*, or learning with a teacher, uses known values associated with an input or labeled examples. Each example pairs unique input signals and a corresponding desired (target) response. One example of this would be that given the trajectory of the hand $(x_1, ..., x_n), (y_1, ..., y_n), (z_1, ..., z_n)$ for the sub-action performed `Reaching` as a label for that trajectory. Then trajectories are used to train the model of the sub-action to identify if other trajectories can be classified as the same sub-action. On the other hand, *unsupervised learning*, or self-organized learning, trains a model with a set of data as inputs, from which the associated values are unknown. The model tries to classify the inputs meaningfully when the training is performed.

This thesis work uses a semi-supervised learning approach for the HAR system. That means we take some data with labels combined with those without them. The resulting model predicts data class memberships based on what it learned during training [Aggarwal and Xia, 2014]. Both types of learning divide the data into training and testing for evaluating the resulting learned model, see Figure 5.2. A complete machine learning process can be seen in Chapter 2, Figure 2.2.

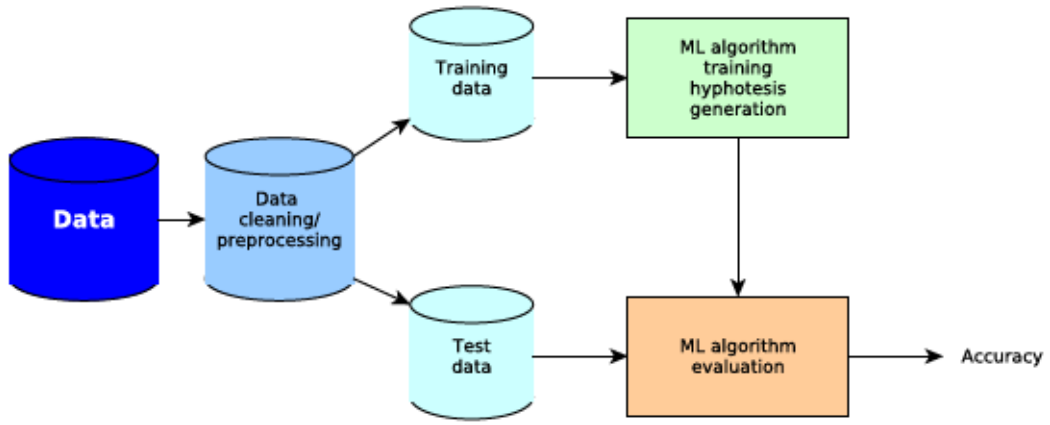


Figure 5.2: Example of data separation for a learning algorithm.

Among the classification approaches commonly used are Data-Driven Approaches (DDA) and Knowledge-Driven Approaches (KDA). DDA are based on machine learning techniques with a pre-existent dataset. KDA build an activity model based on rich prior knowledge from the application domain. In this work, DDA classification is implemented using a *Recurrent Neural Network (RNN)* with *Long Short Term Memory (LSTM)*. This is because the best accuracies were obtained when applying the segmentation technique using this type of classifier.

In this case, to understand how *Recurrent Neural Network (RNN)* works. It is essential to know some background about how ANNs work, see Chapter 2, Section 2.3.4. A RNN is an ANN that keeps track of its input data. RNNs are models suited for processing sequential data [Henaff et al., 2016].

5.1.4 Learning from Demonstrations

The *human activity recognition (HAR)* system is used to provide a representation of action-sub-action pairs inside a *Learning from Demonstration (LfD)* system. LfD requires observations recorded by sensors and acquisition systems to store the information. There are common aspects of LfD among most applications. The first is a *teacher* demonstrating the desired execution behavior. Another is a set of these demonstrations provided to the *learner* (robot), and from them create a *policy* to reproduce those behaviors. This typically happens offline. The difference between online and offline learning is the moment it happens. In offline learning, the data is processed outside and then given to the learner (robot) or inside before or after execution time. In the case of online learning, it happens during task execution and inside the learner. They both

have advantages and disadvantages. One is that offline learning allows for a more extensive data analysis. On the other hand, the result is immediately available with online learning. This imitation technique presented in this work uses external sensors on the demonstrator's body to record the execution and an offline method to analyze the data.

Some LfD implementations, like the one in this thesis work, extract the teacher states/actions from this recorded data and then map the extracted states/actions to the learner. This type of algorithm aims to reproduce the underlying teacher policy, which is unknown, and to generalize over the set of available training examples. The examples are sequences of state-action pairs that are recorded during the demonstration. The policy helps obtain valid solutions and, in that way, handles similar situations that may not have been encountered during demonstrations. In other approaches, the *reward function* is directly learned from the demonstration data [Argall et al., 2009]. The reward function scores the robot's behavior by identifying good or bad actions.

The second phase of this approach requires the action plans generated by the robot to be updated. For doing so, one method is using a planning framework that represents the policy as a sequence of actions that lead from the initial state to the final goal state. Actions can be defined in terms of pre-conditions and post-conditions or teacher annotations. Another approach is based on policy execution failures where the robot recognizes when it cannot complete its task (for example, due to a physical obstruction) and asks the help of a human. Though in this work, the robot identifies a task failure, it does not identify the failure cause or modify its policy.

Similarly to the work by Brooks et al. [2004], this thesis work produces a representation. These representations are created after the demonstration recordings and include tasks and actions with an added notion of goals. This way, when the robot learns a new action, it also gets the goals associated with each sub-action, see Section 5.2.4.

Systems using VR-based demonstrations use different mechanisms to recognize actions and integrate them inside robotic platforms. For example, one system using the same virtual environment as this thesis work uses statistical models use known context and event to identify actions with probabilistic methods to increase their level of success [Kazhoyan et al., 2020a], but requires knowing the executed tasks to identify the actions. In this thesis implementation, this context is not needed. A different approach by Lucci et al. [2022] uses semantic information and state machines to detect the actions. However, the tasks are simple (staking cubes) and the robot cannot recover quickly from an already given post-condition. Another approach using semantic representation is the work by Diehl et al. [2021]. This work has a high success rate with staking actions and has yet to be tested for other cooking-related actions. Another approach by Zhang et al. [2021] uses activity definitions similar to the rules presented in Table 5.2 and decision trees. However, the success rate is lower and recognizes fewer actions than the system presented in this thesis implementation. Another approach is the work by Bajracharya et al. [2020], where a demonstrator can teach a behavior to the robot by setting the parameters for the actions. On the contrary, this thesis implementation takes the parameters from recognized sub-actions given by the recording system.

5.1.5 Datasets

As mentioned earlier, many *human activity recognition (HAR)* approaches use different datasets and are not easily comparable. One classification for datasets is by the types of events or level of activity detail they present [De-La-Hoz-Franco et al., 2018]. In this case, it is the data generated after segmentation and classification from data obtained from the VR framework. It provides events between actors, such as hands, objects, heads, etc. The end dataset includes sequences of cooking-related actions and sub-actions according to the definition presented in Section 5.1.1. These sequences include features such as one or various `Pre-State`, `Goal`, `ObjectActedOn`, `GraspType`, `Hand` used, which depends on the specific action. For example, the action *go* does not have an `ObjectActedOn`.

It is essential to have adequate sensors and a data-gathering system to produce the data included in datasets. One example is incorporating multiple, synchronized information streams from actors and the environment [Aggarwal and Xia, 2014]. With this kind of system, it is possible to track positions and orientations from the head and hands of participants. It is possible to record a video and track all objects with which there is an interaction. In the implementation used in this work and presented by Haidu and Beetz [2016], the interaction video can also be recorded; see Figure 5.3. The advantage is that all locations of objects are known; as it is a VR system, the head and hands are also tracked. However, the fingers are not included, as the controllers that track the hands do not have this feature.

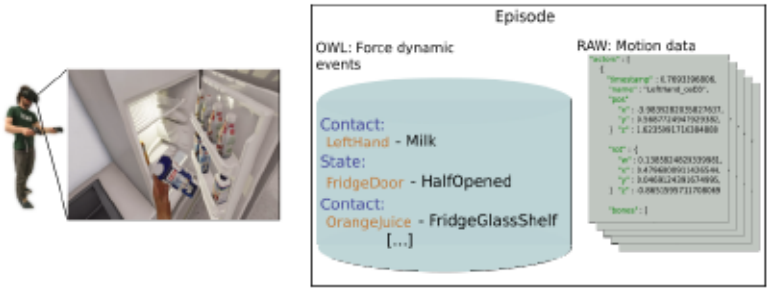


Figure 5.3: From VR to episodes.

Another dataset example is HumanEva, which includes seven calibrated video sequences synchronized with 3D body poses from a motion capture system of 4 subjects performing six everyday actions (e.g., walking, jogging, gesturing, etc.) [Sigal et al., 2010]. The TUM Kitchen dataset demonstrates several subjects setting the table differently [Tenorth et al., 2009]. It also includes motion tracking of the subjects. The work by Ziaeefard and Bergevin [2015] presents the Willow action dataset, which consists of 454 images and nine types of actions such as `phoning`, `playing` an instrument, `reading`, `riding` a bike, `riding` a horse, `running`, `taking` a photo, `using` a computer and `walking`. None of them are related to cooking activities.

Related to cooking, most datasets are for vision and have a third-person view. Some examples are the YouCook dataset, which includes 88 YouTube cooking videos [Das et al., 2013]. Similarly, the Max Planck Institute for Informatics provides the Cooking Activities dataset with some

pose estimation and activity recognition algorithms [Rohrbach et al., 2012]. The largest dataset to date is the Epic Kitchen, which provides first-person (egocentric) viewpoint recordings in 32 native environments, i.e., the wearers' homes, in 4 cities [Damen et al., 2018]. It does not include body positioning information. However, they are not recorded in VR or have depth information. As mentioned earlier, we take some of the objects of the Epic Kitchen dataset to have them in our virtual environment.

5.2 Human Action Recognition from Virtual Reality data

The action words or verbs used in this work are presented in Table 5.1. As mentioned in previous chapters, it includes manual and locomotion verbs related to serving foord and cooking tasks. Manual verbs include object manipulation. Locomotion verbs require full-body movement. The verbs are divided into actions and sub-actions. As mentioned, basic or atomic actions are *sub-actions*, one level above body part *movements*. Movements are defined by the increase or decrease of the angle in the joints [Saladin, 2004]. An *action* is the combination of sub-actions or other actions.

The approach used in this thesis work performs action recognition offline with data collected from the *Virtual Reality (VR)* of right-handed people. The VR system does not have tactile feedback from the objects. However, contact events are predicted and monitored, as in the work by Johansson and Flanagan [2009]. The goal is to produce episodes used by the *Episodic Memory (EM)*; see Chapter 6, Section 6.2.2.

Data is acquired using the VR system mentioned before. From this system, raw data is recorded, which includes time, position, and orientation of objects, hands, head and furniture. To limit the file size, it does not record the position of the hands and head for each time step but only when there is a change in position. But, it also records high-level data, which includes events and objects acted on; see Figure 5.4a. The events include the contact between objects, their state, and if an object was grasped.

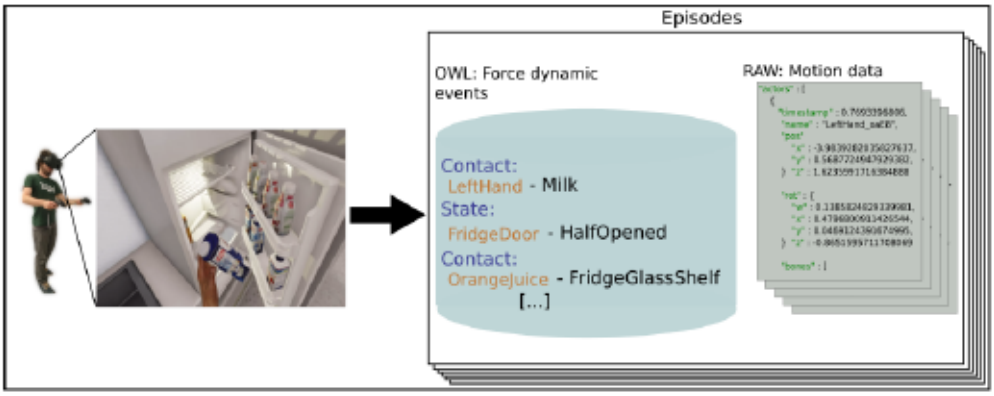
The object type, such as `Sauce`, `Spice`, `Box-Container` and `ContainerArtifact`, among others, defines the grasping type of the `GrippingAnObject` sub-action as `IntermediateGrasp`, `PowerGrasp` or `PrecisionGrasp`. One example is to grasp a teaspoon with precision and an empty cup with an intermediate grasp type.

For the data preprocessing, raw and high-level data are merged to apply rules and machine learning methods; see Figure 5.4b. One of these methods is a set of if-else rules to segment sub-actions; see Section 5.2.1. Another method is a decision tree introduced in Section 5.2.2. Finally, a *Recurrent Neural Network (RNN)* classifier is also used and tested; it is presented in Section 5.2.3 and Figure 5.7.

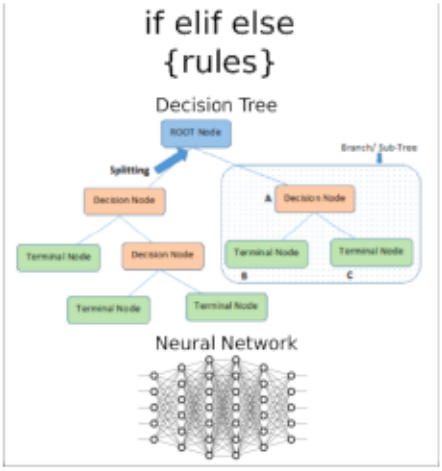
All these methods increase or reduce the confidence of the recognized action and sub-action. Then, this confidence measure from each of them is compared to then select the highest. Finally, a representation in the form of episodes is obtained, as shown in Figure 5.4c.

Table 5.1: Manual and locomotion verbs.

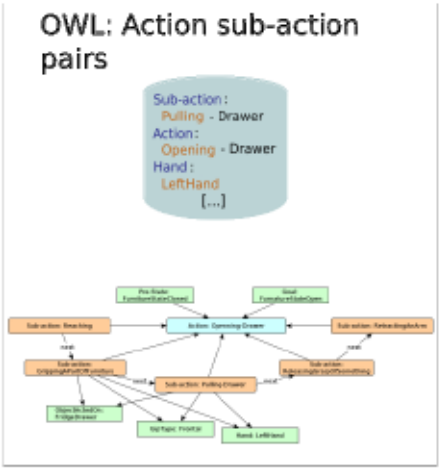
Actions						Sub-actions
acquire	close	fetch	knead	pick	simmer	approach
add	collect	fill	layer	pill	sleeve	carry
apply	combine	find	leave	pinch	slice	flip
arrange	compare	fit	let	poach	smoke	grip
assist	connect	fold	level	pour	soak	hit
attach	cook	form	liquidize	produce	spill	move
bake	count	fry	locate	puree	spray	perceive
beat	cover	get	lock	put	spread	press
blanch	crack	give	lose	regulate	sprinkle	pull
blend	create	glaze	lower	replace	squeeze	push
boil	crush	go	make	remove	start	reach
bone	curdle	grate	manipulate	return	stay	release
break	cut	grease	marinate	rinse	steam	retract
bring	decorate	grill	measure	roast	stew	say
broil	defrost	grind	melt	roll	stop	separate
burn	dice	halve	mill	rotate	take	shake
carve	dilute	heat	mince	scramble	stuff	stir
clean	dip	help	mix	seal	strain	throw
change	dissolve	hold	mount	search	thicken	tilt
chill	drain	incorporate	obtain	season	trim	touch
choose	drop	increase	operate	select	transfer	turn
chop	dry	introduce	order	serve	use	wait
clear	empty	join	peel	sharpen	wash	whisk



(a) Data acquisition.



(b) ML methods.



(c) Representation.

Figure 5.4: HAR process.

The hierarchy and names of action and sub-action pairs need to be known. For that to happen, the semantic knowledge is queried to get them. Specifically, the KNOWROB packages[1] used are `knowrob_actions`, `knowrob_mongo` and `knowrob_objects`. The `knowrob_actions` package includes the description of actions organized in a hierarchy and can consist of sub-actions, pre-conditions, effects and actors. The `knowrob_mongo` package integrates the raw data into the information. The `knowrob_objects` package includes geometrical and visual properties of objects, such as color, dimensions and position.

Prolog Query 6 lists of all sub-actions in the lower hierarchy level with their upper level. It looks for all the actions `Act` with the property sub-action `Sub` in the packages to build a list `ActionList`. This is only required once.

Prolog Query 7 gets all atomic actions and creates a list with them. This is to verify if any is any missing because it has no sub-action in Prolog Query 6. It looks for all the subclasses of the tasks `Task` that belong to the class `knowrob:'Action'`.

After both lists are created, a check is performed to verify the actions and sub-actions repeated and the ones that only appear once. From them, one final list is created with sub-actions to know

[1] github.com/lizyazpin/knowrob

Prolog Query 6 Find all action-sub-action pairs.

```
findall(
        ActionList,(
           owl_class_properties(Act,knowrob:subAction,Sub),
           ActionList=[Act,Sub]),
        ActionList
).
```

Prolog Query 7 Find all actions.

```
findall(
        ActionList,(
           owl_subclass_of(Task,knowrob:Action),
           ActionList=Task),
        ActionList
).
```

the lower level already present inside KNOWROB and used for naming. As mentioned, actions of interest for this thesis work are related to cooking and serving food. The specific actions present in KNOWROB are `boiling` food, `setting` the table, `pouring`, `adding` an ingredient, `cutting`, `opening` and `closing`, `loading` and `unloading` a dishwasher and `turning` on and off a heating device. Each requires a sequence of sub-actions not present before inside KNOWROB are created for this thesis work. The final representation was presented in more detail in Chapter 4.

The segmented activities' features include individual object speeds and accelerations, and the distance between objects. More specifically, there are 15 variables to describe the actors' movement, i.e., the speed and acceleration of each hand and the head, pairwise distances between the tracked points, in this case, the left and right hand and the head, pairwise relative speeds and accelerations. This is explored in a sub-action segmentation implementation presented in Section 5.2.1.

The system used here can detect contact events in an execution timeline. The events related to `grasping` are `GraspingSomething` and `GraspingMulitpleItems`. States of the furniture are identified by their level of completeness. In the case of doors and drawers they are identifies as `FurnitureStateClosed`, `FurnitureStateOpened`, `FurnitureStateHalfClosed` and `FurnitureStateHalfOpened`. There is another identifier for the touching event identified as `TouchingSituation` [Haidu and Beetz, 2016]. They help the identification of manipulation actions while segmenting in this work.

5.2.1 Sub-action segmentation with if-then and if-then-else rules

The data used in this case includes hands, head, furniture and object positions. Also, the time, event and object and furniture names are included. The head orientation is significant, reflecting the attentional focus while acting [Breazeal, 2009]. For that reason, the head position and orientation are used in this segmentation implementation, as well as the orientation of the hands.

Segmentation in this thesis is based on the parameter rules used in event segmentation in Zacks et al. [2009] and Zacks and Swallow [2007]. These rule features are available in Table 5.2. These rules were obtained by analyzing the trajectories and events by using the querying and visualizing tool OpenEASE similar to the work by Kazhoyan et al. [2020a]. Cell values in the case of the *object in hand* can be *T=true* or *F=false*, as at a specific moment, they can be either of them. In the case of *speed*, *acceleration* and *Euclidean distance*, the values can be:

- *R*: the value is in a specific range,
- *I*: the value increases and/or
- *D*: the value decreases.
- Empty means that they are not used

During the segmentation, rule features are included in if-else statements that give a value of confidence. These rules also take into account previous actions to increase their confidence value. That is how the ambiguity between two similar actions decreases, although it is not erased. This can be seen in Algorithm 1 for `Reaching`. In case of the same probability for more than one sub-action, the software selects randomly between sub-actions with the same value. The result is a list of the sub-actions, in this example `Reaching`, with its probability. Other rules are implemented similarly.

This thesis work uses the representation of actions present in KNOWROB. It analyzes the pre- and post-conditions of sub-actions. The `knowrob_actions` motion library includes pattern descriptions such as part of an object and tool usage [Tenorth et al., 2014]. Some descriptions include velocity, the distance between hand and objects and object orientation. Velocity profiles are used to detect a movement to apply further rules then. Motion patterns combined can form a sub-action.

The rules also take into account other features, such as furniture. For example, it was learned empirically in this thesis work that sub-actions such as `Reaching` depend on the furniture type, considering that furniture does not or only partially change location, e.g., drawers and doors can move somewhat. This means that to reach a specific furniture area, for example, a drawer or the counter, the distance between the body and furniture area is different when starting the `Reaching` sub-action. For instance, when `OpeningADrawer`, some space needs to be left for the drawer to take the space.

The unachieved goals or failures are only automatically detected in the case of an event related to furniture states, especially the doors and drawers. These states can be identified as `FurnitureStateHalfClosed`, `FurnitureStateHalfOpened` or `FurnitureStateOpened`. These states are important for the robot as it would require fully opened drawers.

The sub-action names are obtained mainly from KNOWROB, as mentioned. When the name does not exist in the knowledge base, it is created following KNOWROB's naming style. This is done via a verifying knowledge function; see Chapter 6, Section 6.2.4. As mentioned previously, the hierarchy between actions and sub-actions is made by queries to the KNOWROB semantic knowledge base. However, some levels needed to be created, as in the

Table 5.2: Segmentation sub-action rule features.

Subaction	Object in hand	Speed		Acceleration		Euclidean Distance between				
		Hand	Head	Hand	Head	Hand			Head	
						Obj.	Furnit.	Head	Obj.	Furnit.
ApproachingToLocation		R	R		D	D			R	D
CarryingWhileLocomoting	T	R and I	I				D			D
FlippingAnObject	T	R and I	R		I	R			R	
FullBodyMovementToLocation		R	R		R	R	R		R	R
GrippingAnObject	T	R and D		R		D	D		R	R
GrippingAPartOfFurniture	T	R and D		R		D	D		R	R
GrippingAPartOfDevice	T	R and D		R		D	D		R	R
HearingASound			R							
HittingAnObject		R and I	R		I	D			I	R
HoldingWithOneHand	T							R	R	
HoldingWithTwoHand	T							R	R	
LiftAnObject	T	I		R			I	D	D	R
LoweringAnObject	T	D		R			D	I	I	R
MovingAnObject	T	R and I							I	R
Perceiving-Voluntary			R							
PullingAnObject	T	R		R			I	D	D	R
PullingAPartOfDevice	T	R						D	D	R
PullingAPartOfFurniture	T	R					D	D	D	R
PushingAnObject	T	R		R			D	I	I	R
PushingAPartOfDevice	T	R					D	I	I	R
PushingAPartOfFurniture	T	R					R	I	I	R
RetractingAnArm		I	R	I		I	I	D	R	R
Reaching	F	D	R	D		RD	D	I	R	R
ReleasingGraspOfSomething	F	R and I		R		I	I		R	
SayingSomething			R							
SeparateAnObjectFromAnother	T	R and I	R			R			I	R
Stirring	T	R	R	I	I				I	R
TiltingAnArm		R	R					R	R	R
ThrowingAnObject		R and I				I			I	
TurningAPartOfDevice	T	R	R							
TurningAnObject	T	R	R							
WaitForPredefinedTimeInterval			R							
Whisking	T	R		I	I				R	R

Algorithm 1: System to segment sub-action `Reaching` from VR data.

```
Data: event, time, success, object, hand, furniture, pose, orientation
probabilityReach ← 0
listProbabilityReach ← emptyList
listSubActionProbability ← emptyList
while timeSteps available do
    process handPosition, handOrientation, handSpeed, headPosition, handOrientation,
     headSpeed, handAcceleration, handDistanceObject, handDistanceFurniture,
     handDistanceHead, headDistanceObject, headDistanceFurniture;
    if not ObjectInHand then
        [...]                                                    ▷ Other sub-actions
        if handSpeed decreases and headSpeed in range and handAcceleration
         decreases and handDistanceObject in range and handDistanceObject
         decreases and handDistanceFurniture decreases and handDistanceHead
         increases and headDistanceObject in range and headDistancceFurniture in
         range then
            probabilityReach ← 0.5 if previousAction in list(ApproachingToLocation
             or ReleasingGraspOfSomething or RetractingAnArm or PushingAnObject
             or LiftingAnObject or Reaching) then
                probabilityReach ← probabilityReach + 0.1
            end
        end
        listProbabilityReach append (probabilityReach)
    end
end
compare listProbabilityReach with other listProbabilitySubAction
listSubActionProbability append ([Reaching, probabilityReach],...)
return listSubActionProbability
```

case of `OpeningADrawer`. This action requires `Reaching`, `GrippingAPartOfFurniture` and `PullingAPartOfFurniture`, which was not present in KNOWROB. This was required to create demonstrations from actions and sub-actions not included in previously in KNOWROB. By creating a clear distinction between a movement, sub-action and action, the hierarchy is not only understandable to robots, but also to humans using the system.

One issue with this approach is the similarity between the rules, which have mis-segmentation in 30% of the cases. It is for that reason that a complementary approach is required. However, these results, plus a verification process by looking at the videos, allowed the creation of verified labeled data used to train the decision tree and RNN classifiers.

5.2.2 The sub-actions classifier by a decision tree

This decision tree is implemented by using the Classification and Regression Trees (CART) algorithm [Breiman et al., 1984]. It was chosen because it builds binary trees using the feature and threshold that produces each node's most significant information gain. This is accomplished by dynamically defining a discrete attribute that separates the continuous values into a set of intervals. It converts the tree into a set of if-then rules. Then, the accuracy of each rule is evaluated to determine the order in which they should be applied.

Mathematically, the CART algorithm can be described by a given training vector $x_i \in \mathbb{R}^n$, $i = 1, ..., n$ and a label vector $y \in \mathbb{N}$ (sub-actions). The training vector includes the `Object`, `Furniture` and `Event` names involved in the sub-action, including if `ObjectInHand`. It also includes the speeds and accelerations of the head and hand, and the distances between the head and object `distHeadObj`, head and furniture `distHeadFur` and head and hand `distHandHead`.

The decision tree recursively separates the samples in a way that the same number n of labels are in groups G using the equation

$$G(Q, \theta) = \frac{n_{left}}{N_m} H(Q_{left}(\theta)) + \frac{n_{right}}{N_m} H(Q_{right}(\theta)) \tag{5.1}$$

where Q represents the nodes, N_m is the allowed depth of the tree or the number of training samples at the last node, H is the Gini impurity (Equation 5.4) to measure the probability of a class in the total amount of classes.

It lets the data at node m be represented by Q. For each candidate, it splits $\theta = (j, t_m)$ consisting of a feature j and threshold t_m. In this case, a parent node splits into two children nodes. The $Q_{left}(\theta)$ takes the values that are inside the threshold and is defined by

$$Q_{left}(\theta) = (x, y)|x_j <= t_m \tag{5.2}$$

The second child, $Q_{right}(\theta)$, takes the remaining subset of values and is defined by

$$Q_{right}(\theta) = \frac{Q}{Q_{left}(\theta)} \tag{5.3}$$

The CART algorithm uses a *Gini impurity* to quantify how often a randomly chosen element from the set would be incorrectly labeled related to the distribution of labels in the subset. The impurity at m is computed using the function $H(X_m)$ defined by the sum of probabilities p of node m and classes k as follows.

$$H(X_m) = \sum_k p_{mk} * (1 - p_{mk}) \tag{5.4}$$

where X_m is the training data in node m.

It selects the parameters that minimize the impurity by

$$\theta^* = argmin_\theta G(Q, \theta) \tag{5.5}$$

Recurse for subsets $Q_{left}(\theta^*)$ and $Q_{right}(\theta^*)$ until the maximum allowable depth is reached, $N_m < 5$ in this implementation. The absolute maximum depth is $N_m - 1$, where N_m is the number of training samples at the bottom node.

If a target is well classified by taking on values $0, 1, \ldots, K-1$, for a node m, representing a region R_m with N_m observations, the probability of classes and nodes p_{mk} is modified by a proportion of a class k of observations in node m defined by

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{5.6}$$

where I is the indication or characteristic function that maps the elements that belong to the class K and give them the value 1 and 0 to the rest.

As mentioned in Section 2.3, decision trees can *overfit*. This happens when the algorithm fails to fit more data to predict their values. In this implementation, the minimal cost-complexity algorithm is used to avoid over-fitting. This algorithm is parameterized by $\alpha \geq 0$ known as the *complexity parameter*. This parameter is used to define the cost-complexity measure $R_\alpha(T)$ of a given tree T and is described by

$$R_\alpha(T) = R(T) + \alpha|T| \tag{5.7}$$

where $|T|$ is the total number of nodes T and $R(T)$ is the total sample Gini impurity of those nodes. Minimal cost-complexity pruning finds the subtree of T that minimizes $R_\alpha(T)$.

The data was split by using the *K-fold cross-validation* method [Ojala and Garriga, 2010]. K-fold cross-validation splits the available data into k subsets or "folds." The main advantage of using k-fold cross-validation for decision trees provides a more accurate estimate of the model's performance than a single train-test split.

This method was selected because when using a single train-test split, the model's accuracy may depend on which specific observations are included in the training and testing sets. With k-fold cross-validation, each observation is used for testing exactly once and used for training k-1 times. This ensures that the performance estimate is more stable and less dependent on the specific split of the data. In addition, using k-fold cross-validation allows the model to be trained on a larger proportion of the data, which can help improve its accuracy. This is especially important when the size of the dataset is relatively small.

As mentioned, the dataset is split into k equally sized subsets or "folds." One fold is set aside as the test set, and the remaining k-1 folds are used as the training set. The decision tree is then trained over the chosen training set and evaluated on the test set.

This process is reiterated k times, with each fold used once as the test set and the remaining k-1 folds used as the training set. The decision tree's performance is then evaluated as the average of the performance scores across all k-folds.

The k-fold cross-validation process prevents overfitting by training and testing the model on different subsets of the data. This also reduces the variance of the performance estimate by using multiple training and test sets.

For this decision tree model, *k=10*. This is because smaller values of k are less computationally intensive and require fewer data to produce reliable estimates of the model's performance.

The data delivered by the k-fold cross-validation process is randomized. The randomization generates indexes from which one sample is used as a test set (singleton) while others are used for the training set. One simplified example of the rules created by the decision tree for closing and opening a drawer can be seen in Figure 5.5. The colors represent different things. White represents the numerical values such as the distance between the hand and furniture, the velocity of the head and hand, the acceleration of the head and hand, etc. Light blue represents the object type, such as `PartOfDevice` and `PartOfFurniture`. Light green represents events, such as the state of furniture like *FurnitureStateHalfClosed*. In the case of sub-action, they are represented with yellow. The arrows represent the decision nodes required to return a termination node (sub-action). For example, the sub-action `ApproachingToLocation` required the distance between the hand and object (`distHandObj`), distance between the head and object (`distHeadObj`) and the acceleration of the head (`aceleHead`) to be identified. Furthermore, these decision nodes combined with the distance between the hand and furniture (`distHandFurniture`), `PartOfFurniture` and distance between the head and furniture (`distHeadFurniture`) helped to identify the sub-action `GrippingAPartOfFurniture`.

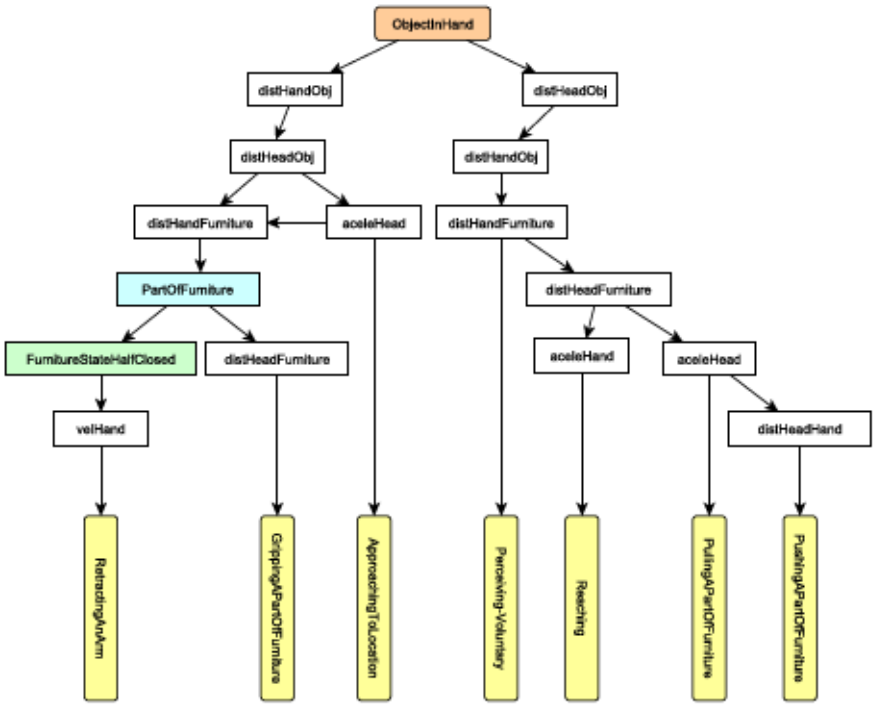


Figure 5.5: Minimized decision tree result for `OpeningADrawer` and `ClosingADrawer` actions for right hand.

The trained model results show how the decision tree performs when testing samples are used as input. As mentioned, these testing samples differ from the ones the tree was trained with, as a good practice. A confusion matrix was created to show these results; see Figure 5.6. A *confusion matrix* shows false positives and negatives during evaluation arranged in a table. It indicates every sub-action that the decision tree predicts. The test samples are run through the decision tree trained model to calculate the confusion matrix. The resulting values have a prediction rate appear, which appears for each input. These values show the correct predictions from 1 to 100 percent.

For the decision tree trained model, most sub-actions are correctly classified with more than 90% accuracy, which means that it is a good model. This result is better than using if-then rules. Only two of them are under 90%; the fists one is when there is no sub-action with a 88%. However, the decision tree misses the *ReleasingGraspOfSomething* sub-action 40% of the time. This is because it only differs in having an open gripper and other features are very similar to other sub-actions.

The trained model is saved for being then used to predict new sub-actions. Each prediction includes an accuracy percentage, shown in the confusion matrix in Figure 5.6. This value is used for comparing with the *Recurrent Neural Network (RNN)* result and deciding which sub-action was most likely to happen in the data coming from the VR demonstrations.

The description of the RNN implementation is presented in the next section.

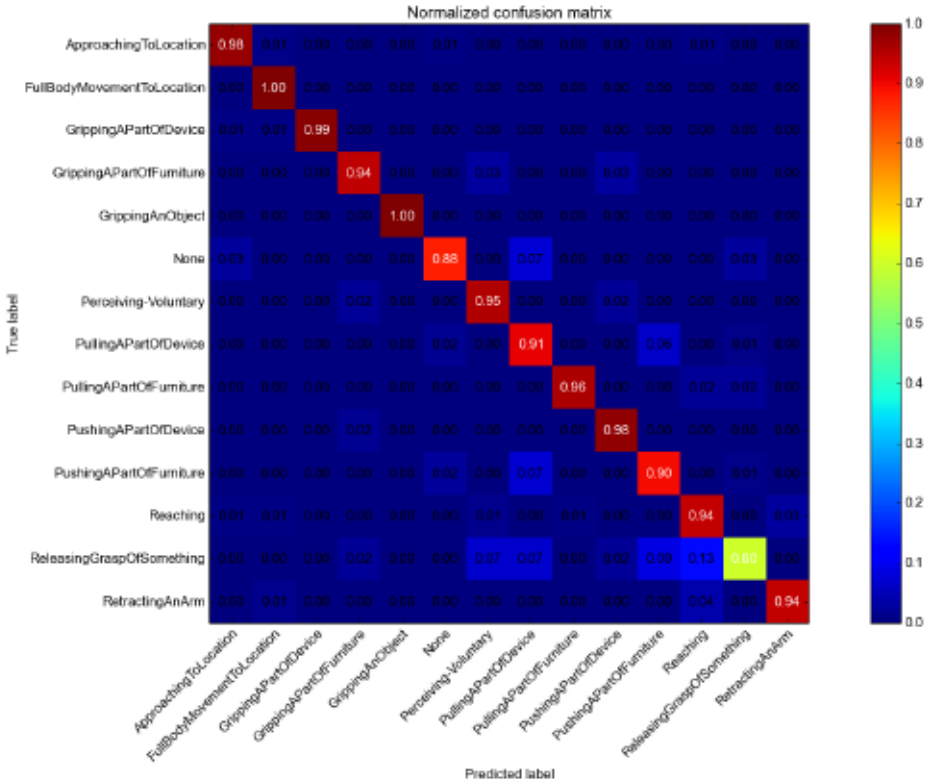


Figure 5.6: Normalized confusion matrix sub-action prediction for `OpeningADrawer` and `ClosingADrawer` actions for both hands.

5.2.3 The sub-actions classifier by a Recurrent Neural Network

A classification approach using *Recurrent Neural Network (RNN)* with *Long Short Term Memory (LSTM)* was implemented because, as mentioned in Section 5.1.3, it can give the best results. The architecture used is presented in Figure 5.7. The RNN is implemented as a regular feed-forward layer with LTM (green) and STM (orange). More information about how ANNs work can be found in Section 2.3.4.

The sub-actions are stored in a dictionary. Each sub-action includes the positions and orientations of the head and hands where the sub-action was found during the rules approach and verification process. The sub-actions are used as targets and their elements as a list of values for the input data x. This input data is also split using the *K-fold cross-validation* [Ojala and Garriga, 2010]. In general terms, K-fold cross-validation works similarly for decision trees and RNNs. It separates the data into k subsets or "folds" and trains the model on k-1 folds while using the remaining fold for testing. However, there are some dissimilarities in the way k-fold cross-validation is applied to decision trees and neural networks. One key difference is that decision trees and neural networks often have different hyperparameters that must be tuned. For example, decision trees have hyperparameters such as the maximum depth of the tree, the smallest number of samples essential to split a node, and the splitting criterion. RNNs, on the other hand, have

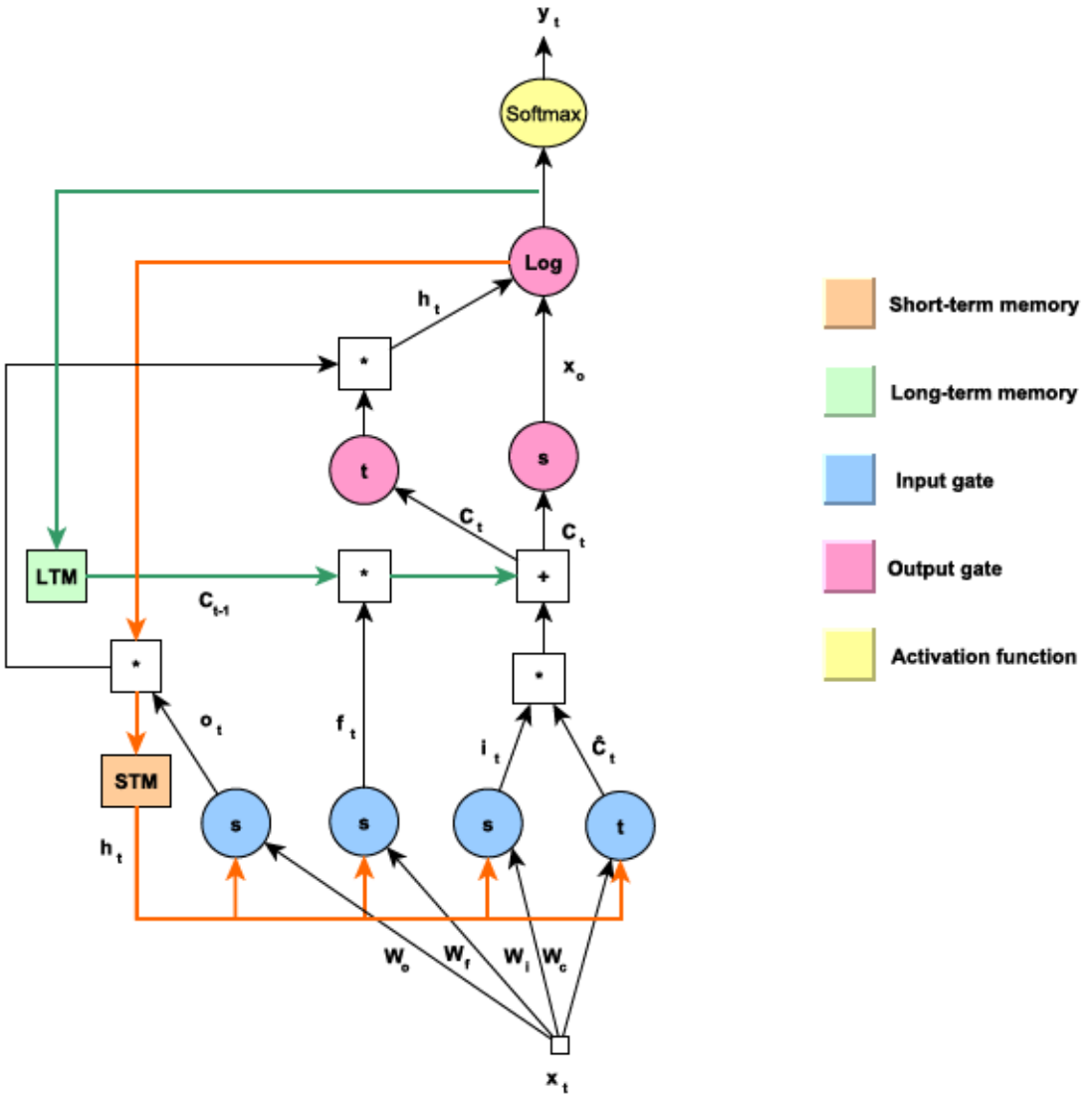


Figure 5.7: RNN with LSTM architecture for sub-action classification.

hyperparameters such as the amount of hidden layers and neurons per layer, the activation function or functions to use and the learning rate.

For this thesis work, *k* in *k-fold cross-validation* is 10. This allows for sufficient training and test sets to provide reliable estimates of model performance while minimizing the computational cost of training the model multiple times. The trade-off between the size of k and the number of observations in each fold was balanced by avoiding a small number of observations in each fold; in this case, 100 observations were selected. This way, *overfitting* was avoided, resulting in a stable and reliable estimate of the model's performance.

The system keeps track of all categories of names and the total number of categories. The input data is defined then by $x \in \mathbb{R}^{m \times D}$ where each row of x is a $D - dimensional$ data point of 33 values and m is the number of training samples. These values include the position and orientation of the hands, head, object and furniture.

To decide which previous information will be kept, the sigmoid layer s called *forget gate layer* uses the values of the previous hidden state h_{t-1} and current input x_t with the weight W and adds the bias b_f, which is independent of the input and helps to increase the classification accuracy.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5.8}$$

In the case of new information, another sigmoid layer called the *input gate layer* creates a vector of the input values.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{5.9}$$

In parallel, a *tangent layer* t creates a vector of candidates $\hat{C}_t$.

$$\hat{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{5.10}$$

Then, both values i_t and $\hat{C}_t$ are combined $\times$ to create the updated state C_t to replace C_{t-1} by

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{5.11}$$

For obtaining the first output x_o, another sigmoid function is used as follows,

$$x_o = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{5.12}$$

The hidden state h_t used by the STM is obtained using a tangent function by

$$h_t = o_t * tanh(C_t) \tag{5.13}$$

The LogSoftmax and negative log-likelihood functions then use these last values. They are used as activation and the cost function of the second layer, respectively. They are typically used together [Bishop, 2006, p. 210]. This LogSoftmax function was selected because it calculates the probabilities distribution of the event over k different events. In other words, it calculates the probabilities of each target class over all possible classes. Its main advantage is that the output probabilities range from 0 to 1, as the sum of all the probabilities should equal one. Compared to the sigmoid function (Equation 2.14) used for binary classification, its probabilities sum is not always equal to 1.

The implementation reads the sub-actions list of values and obtains a prediction and *hidden state* h_t at each step by combining the input and previous hidden state using Equation 2.17. The initial values are zeros. Then the negative log-likelihood cost function is applied to h_t and target. It computes a value that estimates how far the output is from the target. The log-likelihood cost function is defined by

$$J_i = \frac{e^{f_k}}{\sum_j e^{f_j}} - 1 \tag{5.14}$$

where f is a vector containing the class scores for a single example, f_k is then an element from that class k in all j classes.

In this case, the activation function used is LogSoftmax and is described by

$$LogSoftmax(x_o) = \log\left(\frac{e^{x_o}}{\sum_j e^{x_j}}\right) \tag{5.15}$$

The output is defined by $y \in \{0,1\}^{m \times D}$, where each row in y denotes the membership of each point to a class C. Given that $y_{jc} = 1$ if the jth row of x belongs to the class $\hat{C} \in \{1, ..., C\}$ and $y_{jc} = 0$ otherwise. The final prediction is the class to which the sub-action belongs according to the prediction.

To see how the network performs on different categories, a confusion matrix was created, see Figure 5.8. As mentioned, the *confusion matrix* is a table that shows the false positive and negative errors; it indicates for every actual sub-action (rows) which one the network guessed (columns). The test samples are run through the network to calculate the confusion matrix. In case there are not enough examples, the sub-action needs to be iidentified, as well as when there are more examples during training. This explains the tendency for the diagonal cells to have colder colors (direction blue).

To get the best results, two approaches were implemented. In the first one, a random selector takes the sub-action elements for a specific number of iterations. On the other hand, all the training data is taken as a sequence of sub-actions.

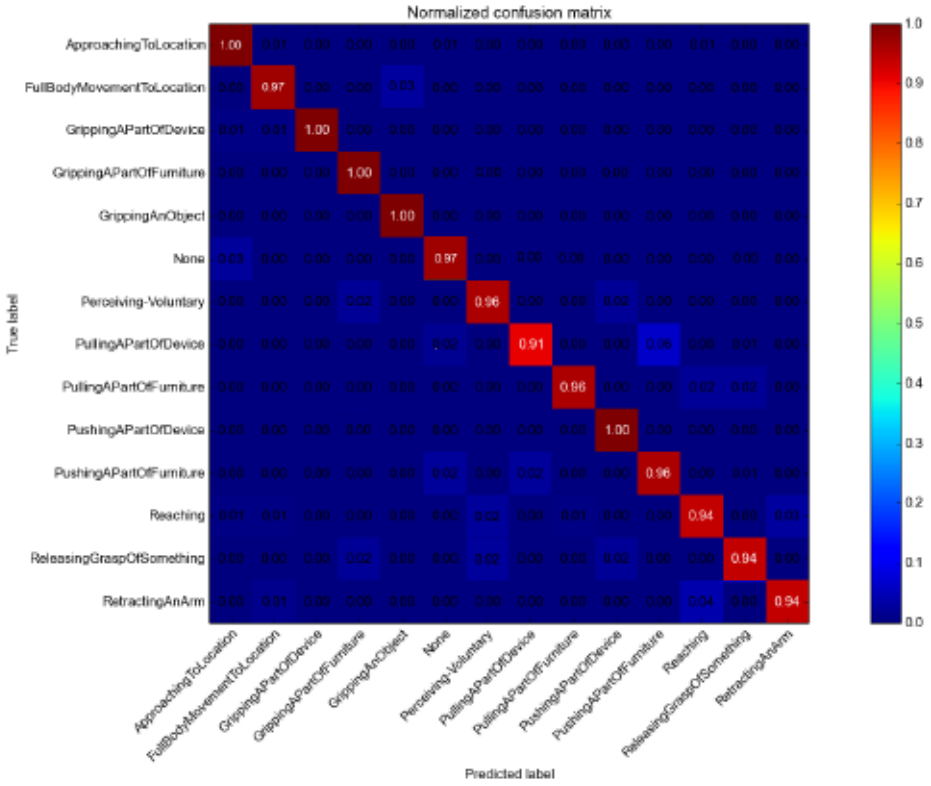
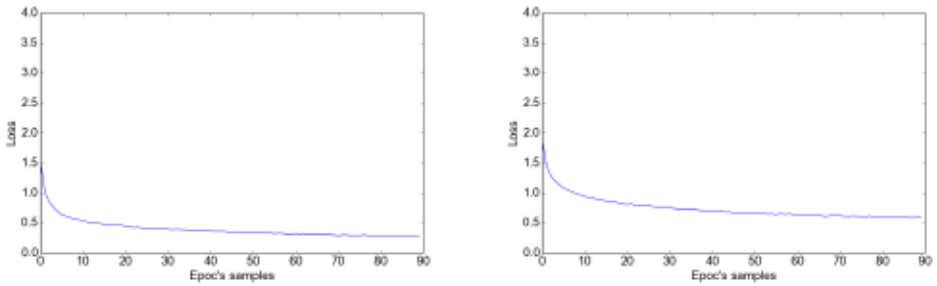


Figure 5.8: Sub-actions evaluation for random inputs for `OpeningADrawer` and `ClosingADrawer` actions.

In both cases, random and sequential, the evaluation takes the test data to verify how many samples were correctly guessed. The results can be seen in Figure 5.8, only for the random input approach. This is because the results in some cases are better for random data, as mentioned in the tutorial presented by Vidal et al. [2017]. This was the case for this application, and also, sequential training takes much longer than random, as it iterates over all training data for the same number of times.

The output and loss are obtained during training to track the cost or loss. When training finishes, the average of the loss is obtained.



(a) Left hand. (b) Right hand.

Figure 5.9: Random input for RNN sub-action classification loss.

Two tests were performed for both implementations, random and sequential input. In the first test, the sub-actions were classified by the hand used (left or right); see Figure 5.9b and Figure 5.9a.

It is interesting to note that the loss of the left hand is performing better. This is related to the fact that there are fewer sub-actions performed by this hand, as the data only includes right-handed samples. In the case of both hands, the number of classes is much more significant, almost double, because of combinations. This gives fewer examples for each class.

In the case of sequential data, a similar test as in the previous section was made, in which sub-actions were classified by hand separately and as both hands pairs. The loss, in this case, is lower than in similar sections. Even when this training approach takes much longer than the previous one, it does not give better cost reduction.

The resulting trained model is also saved from being used to predict sub-actions. Each prediction includes an accuracy percentage used for the comparison with the decision tree result.

In this case, it was not possible to use a large pre-trained model because the type of data used in this work is not found in this type of model.

5.2.4 Action and sub-action prediction

Table 5.3: Comparison of confidence scores between decision trees and RNN.

Sub-action	Confidence score [%]	
	Decision tree	RNN
ApproachingToLocation	98	100
FullBodyMovementToLocation	100	97
GrippingAPartOfDevice	99	100
GrippingAPartOfFurniture	94	100
GrippingAnObject	100	100
None	88	97
Perceiving-Voluntary	95	96
PullingAPartOfDevice	91	91
PullingAPartOfFurniture	96	96
PushingAPartOfDevice	98	100
PushingAPartOfFurniture	90	96
Reaching	94	94
ReleasingGraspOfSomething	88	94
RetractingAnArm	94	94

As can be seen in Figure 5.6 and Figure 5.8, in some cases, the decision tree has better results than the *Recurrent Neural Network (RNN)* and vice-versa. In both cases, the evaluation provides

Algorithm 2: Sub-action selection.

```
Data: listNN(sub-actionNN,scoreNN), listDT(sub-actionDT,scoreDT),
      listRule(sub-actionRule)
for value ∈ length(listNN) do
    sub − action ← sub − actionRule[value]
    if scoreNN[value] == scoreDT[value] then
        sub − action ← sub − actionNN[value]
    end
    else
        if scoreNN[value] > scoreDT[value] then
            sub − action ← sub − actionNN[value]
        end
    end
    else
        if scoreNN[value] < scoreDT[value] then
            sub − action ← sub − actionDT[value]
        end
    end
    listSubAction append (sub-action)
end
compare listSubAction with KB
return listSubActionAndAction
```

a confidence score for every prediction (white text color of confusion matrix) presented in Table 5.3. Then, the final sub-action prediction selects the best score from both methods, as shown in Algorithm 2.

First, the predefined sub-action is produced by the rules explained in Section 5.2.1. Then, the scores from the RNN and decision tree are compared to select the most likely sub-action. All selected sub-actions are stored in a list. Then, the system verifies the existence of sub-actions in the *Semantic Knowledge Base (SKB)*. This KB includes KNOWROB and verbs presented in Table 5.1. If the sub-action belongs to multiple actions, it verifies a sequence of sub-actions until finding the right one.

In the end, a hierarchical task representation is created where a goal is encoded; see Figure 5.10. In this case, the action `Opening-Drawer` (blue) has an ordered sequence of sub-actions (orange) that are identified as `Reaching`, `GrippingAPartOfFurniture`, `Pulling-Drawer`, `ReleasingGraspOfSomething` and `RetractingAnArm`. Those actions have other features (green) such as a `Pre-State`, `Goal`, `ObjectActedOn`, `GraspType` and `Hand` used.

In case the sub-action does not belong to any action, it verifies another dictionary created before in Prolog Query 7.

As mentioned, the interest in this thesis work is manipulation actions. Usually, these specific action types start with the sub-action `Reaching` and end with `RetractingAnArm`. The implementation also considers this if the sub-action does not have a parent action.

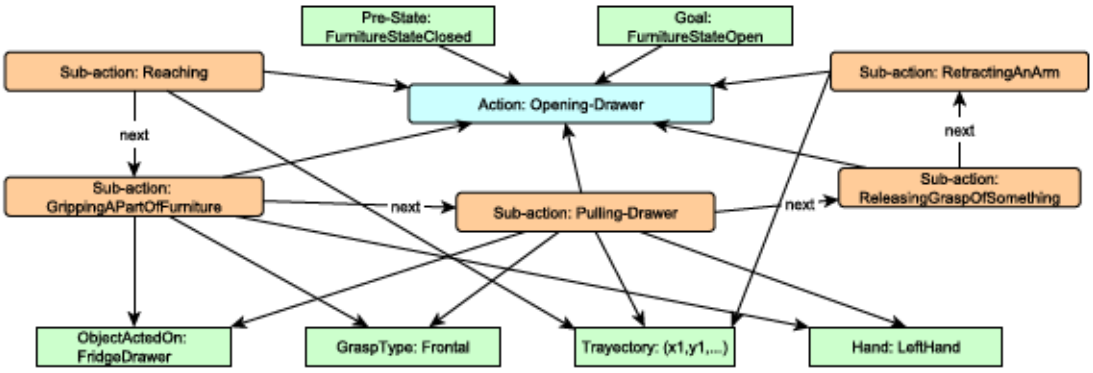


Figure 5.10: HAR representation example.

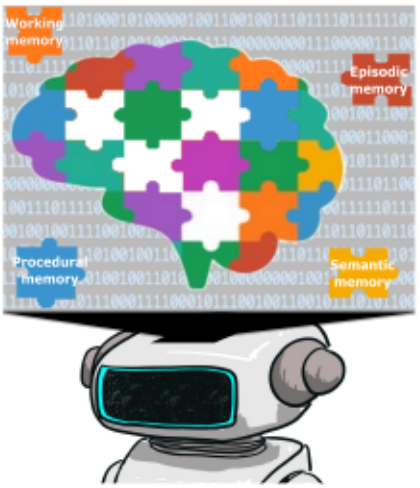
5.3 Summary of this chapter

This chapter presented the approach used to segment and classify sub-actions and actions to fulfill the requirements of a *human activity recognition (HAR)* system from *Virtual Reality (VR)* data. To give an introduction to what a manipulation action and sub-action mean in this thesis work, the state of the art regarding actions and their types was summarized. An introduction to HAR systems was also presented. They include segmentation rules and classification using a decision tree and *Recurrent Neural Network (RNN)* with a *Long Short Term Memory (LSTM)*. Finally, this chapter introduces the prediction approach using segmentation and classification of sub-actions, as well as probabilities of actions depending on their structure.

After this prediction is made, the representation of such action-sub-action pairs is made; it is presented in more detail in Chapter 4 as episodes. If needed, it is possible to query information regarding their structure and features from such representations using KNOWROB and OPENEASE. However, these episodes are used by the memory approach implemented in this thesis work. These episodes complement the experience inside the episodic memory and give information about which sub-actions are required by action to the working memory, both present in Chapter 6 and used by the Personal Service Robot (PSR).

Chapter 6

Knowledge processing memory models for Personal Service Robots



A Personal Service Robot (PSR) requires skills to perform complex manipulations in human environments. To act within more complex situations, PSRs need to recall knowledge from past events regarding interactions between agents (humans or robots) and their environment. This requires memory to store or obtain knowledge. The use of memory models from neuroscience and psychology has been applied to robotic agents for some years. These models are built by combining different memory types. In this work, the types of memories used are *Episodic Memory (EM)*, *Procedural Memory (PM)*, *Working Memory (WM)* and *Semantic Memory (SM)* are used in most literature and defined in this chapter. They belong to the *Long Term Memory (LTM)* and *Short Term Memory (STM)*. However, the terminology used in this chapter comes mostly from psychology and neuroscience. Even though this chapter describes a computational framework, the term memory does not refer to the physical space in the computer but the biological sense by building association maps similar to the human brain into a cognitive robot. One reason to use memory is that it improves learning and the behavior can be optimized using previous experience [Salgado et al., 2012].

This chapter introduces memory concepts with their features in Section 6.1. These concepts are then applied to the construction of a cognitive framework. An explanation of how those concepts are used in this work is presented in Section 6.2. The result is an implementation specialized in

gathering information for the execution of manipulation actions in human environments. This implementation was tested and the results are presented in Section 6.3.

6.1 Concept of memory

Memory is a single term that refers to many human capacities. Its definition can be thought of in terms of answering questions such as "what is the memory's function?." According to Tulving [2007], there are standard definitions of memory. One of them, memory, is a neurocognitive capacity for encoding, storing and retrieving information. In a second definition, memory is seen as a hypothetical store in which content is held and stored, and some properties and processes of that content are retrieved. In a third definition, memory allows great awareness of remembering something.

Some of the memory features can be found in other literature. In work by Zacks et al. [2009], memory has an active, constructive and associative process. On the other hand, Hanheide and Sagerer [2008] see memory as the base for cognitive processing and learning. Even when the concept of memory varies, it is required for cognition and learning as it allows the acquisition, storage, retrieval, use and mix of information, knowledge and experience [Tulving and Szpunar, 2009]. Storage is a way in which memory keeps its content, including its change over time [Laird and Mohan, 2014]. Memory allows the retrieval or access of conceptual categories for specific objects, actions and other kinds of information, knowledge and experience. The storage in biological brains can change with time, as there is a conceptualization process between current and previous events [Tulving, 2000]. That way, changes in memory make possible learning and the development of abilities that allow agents to act.

As the biological brain has a limited capacity for retention, memory can also remove some content from its storage by a forgetting process. Some systems presented in Section 6.1.2 use some models for forgetting. This work does not implement a forgetting model but updates knowledge instead.

Understanding how memory works has eluded scientists; there are only some clues. What is clear is that there are many different kinds of memory. This can be seen in the difference between knowledge from learned facts and recollection of past events; both are stored in different brain areas [Tulving, 2007]. Some types of memory are presented next in Section 6.1.1.

6.1.1 Types of memory

Different areas of neuroscience and psychology agree that a set of memories is required for an agent's different types of knowledge to learn and extend its capabilities [Laird and Mohan, 2014]. Numerous definitions of memory types refer to various human capacities and knowledge types. Tulving [2007] presents 256 as a specific number for memory types. Some of these memory types are embedded, e.g., iconic memory inside sensory memory and semantic memory

in declarative memory. However, the names and structure must be agreed upon [Tulving, 1998]. The literature known for this work agrees on the distinction between two types of memory, the *Long Term Memory (LTM)* and *Short Term Memory (STM)*. STM includes the working memory and provides context to perform tasks or for further processing when receiving information from the outside via sensors [Deutsch et al., 2008]. As it requires active maintenance of information flow, this information is only relevant for a short time [Salgado et al., 2012]. When knowledge in the STM is consolidated, it is transferred to the LTM, which keeps a large amount of information. Furthermore, LTM is divided into explicit and declarative, and implicit and procedural memory. *Explicit memory* associates with conscious memories. *Declarative memory* is related to knowledge about things and facts. It can be further split into *Semantic Memory (SM)* and *Episodic Memory (EM)* [Winkler et al., 2014], the first one stores encyclopedic knowledge and the second experience. *Implicit memory* associates with unconscious or automatically stored. Finally, *procedural memory* is linked to skills. This division depends mainly on the type of information they store and on their principles of operations. The graphical representation of this division can be seen in Figure 6.1.

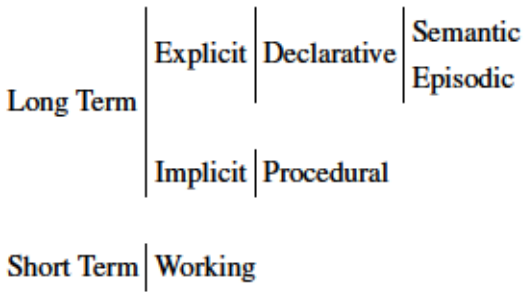


Figure 6.1: Memory type division.

In the case of computer science literature, the term *Long Short Term Memory (LSTM)* is used to join their capacities [Deutsch et al., 2008]. Some people categorize artificial memory types along dimensions other than the ones mentioned above. In the case of declarative memory, it is often considered for accessing information, while procedural memory is seen as a compilation of information [Winkler et al., 2014]. Functionally, memorization is divided into three distinct processes encoding, storage, and retrieval. *Encoding* is related to representing and mapping information in the memory. *Storage* is related to accumulating information into LTM. And *retrieval* queries answers from memory about the data stored. In the case of this work, the system exploits the *Working Memory (WM)* as an interface to the *Semantic Memory (SM)*, *Episodic Memory (EM)* and *Procedural Memory (PM)* by asking for and sending information. These types of memory are introduced next.

Working memory

Working Memory (WM) is used in cognitive psychology to refer to a system with temporary storage and manipulation of information for solving cognitive capabilities such as comprehension, attention, learning and reasoning [Baddeley, 2000]. In the human brain, WM is a memory that represents current events and uses a buffer to compare with previous ones [Zacks et al., 2009, Kotseruba and Tsotsos, 2018]. This memory retains relevant information not just related to events but also to the current task. WM is also believed to have a primary role in planning and preparing voluntary physical actions [Rosenbaum et al., 2012]. For these reasons, it is used in robotic applications and this thesis work implementation.

The WM is capable of language comprehension, dialogue management, problem-solving and acting in the world by interacting with other memories. This includes mappings between objects in the environment and internal symbols and words. This way, this memory links the LTM with the information from the environment.

In some computational systems, WM is used as well as a link between the environment and the system's internal states. One example is the work by Nuxoll and Laird [2012], where the WM encapsulates the agent's current state, including external sensing, the results of internal inferences, selected actions, and active goals. They use triplets of an identified attribute and value called working memory elements (WMEs). Some work also includes this type of memory inside *Artificial Neural Network (ANN)* models. One example is the work by Deklel et al. [2018], which uses associative memory as WM for learning context-free grammar. This associative memory is implemented as a particular *Recurrent Neural Network (RNN)* type.

In other applications similar to this work's implementation, WM is used as a module to store and manipulate information for a short time and divided into units in the long term [Deutsch et al., 2008]. Another model also retains task information [Phillips and Noelle, 2005]. The model stores the required information and discards it after use by keeping only the actions or goals that fulfill the requirements, such as the desire to grasp a specific object. This adaptive memory model uses RL to retain similar details to decide which information is essential.

The use of WM inside a complete integration, such as cognitive architectures, was mentioned in Chapter 2, Section 2.4.3. Different architectures use diverse features and models related to this kind of memory, such as a world state, processing system or activation mechanism, to enter knowledge into the LTM [Kotseruba and Tsotsos, 2018]. On the other hand, WM is used both in this implementation as a processing system to act in the world and as a knowledge selector for the information stored in the LTM.

Procedural memory

PM is the type of memory related to skills and actions. It contains knowledge about how to get things done [Kotseruba and Tsotsos, 2018]. It also includes information regarding habitual motor skills from the body [Tulving, 2001]. PM is capable of obtaining knowledge on how to perform tasks, even when declarative memory structures are damaged [Johnson, 2012].

Implicit memories are often procedural and focused on the step-by-step processes that must be performed to complete a task. These memories are mainly unconscious and occur automatically; you don't need to think about the specific steps you need to follow to complete each task. Repetition becomes automatic over time. It begins with learning skills, and then it is possible to master a task.

In the computational implementation by Laird and Mohan [2014], PM has a set of rules that are compared to the WM to select internal and external actions. The difference to this thesis work implementation is that the structure of actions is compared instead of rules.

This thesis work uses the ideas from the work by Lum et al. [2012] to proceed gradually as stimuli are repeated and skills practiced until knowledge has been acquired; then, skills can be executed rapidly. This can be seen in the implementation presented next, where the execution of actions is crucial to acquiring knowledge by looking at similarities and differences in comparison with repetition.

The use of PM models in different cognitive architectures was introduced before in Section 2.4.3. In those implementations, PM stores long-term facts about how to perform actions [Kotseruba and Tsotsos, 2018], as in this work.

Semantic memory

In the human brain, *Semantic Memory (SM)* is believed to include conceptual knowledge, such as associations and concepts that underlie worldwide meanings, categories, facts and propositions [Tulving, 2001, Patterson et al., 2007]. This type of memory builds awareness about the existence of world objects, events, word meanings, facts, people and other patterns without connection to any particular time or place. SM allows beings to acquire, store and use knowledge of the world [Tulving and Lepage, 2000]. However, it does not require language for its operations.

Theoretical positions about human SM share the view that much of the content of this memory is responsible for perceiving and acting [Patterson et al., 2007]. This happens as the neural representation of how objects look, sound, move, etc., is widely distributed across our neural network.

The use of SM in cognitive architectures was presented in Section 2.4.3. In that case, this memory is used to store concepts about objects and their relationships [Kotseruba and Tsotsos, 2018]. In this thesis work implementation, the SM also stores concepts about actions, physical properties, mathematical operations, etc.

Episodic Memory

In humans, *Episodic Memory (EM)* provides the ability to remember where and when they have been, what they have sensed, and what actions they have taken in various situations [Nuxoll and Laird, 2012, Patterson et al., 2007, Pause et al., 2013]. This memory can also include the individual's internal state, including emotions, perceptions, and thoughts. The EM has

time, self and *autonoetic consciousness* (placing oneself in the past and future to see outcomes and examine thoughts) [Tulving, 2001, Nuxoll and Laird, 2007]. Time can have a sense of subjectivity which enables the distinction between mental representations of an individual in the past, present, and future.

It is interesting to notice that EM can be associated with both, *Long Term Memory (LTM)* in the autobiographical process and *Short Term Memory (STM)* in the temporal duration of events. In the human brain, this memory allows beings to remember personal experiences and events. Because of that, EM enables an individual to mentally travel from the present back to their past to plan for the future [Tulving and Lepage, 2000]. This happens when comparing previous actions and their results to improve future performances [Nuxoll and Laird, 2004]. It relates to the awareness of self-experience as *remembering* and forward as *thinking about*, *imagining* or *planning for* the future. This type of memory stores information about what happened, where and when.

There are three distinguishing aspects of EM [Tulving, 2007] which are awareness of personal past, acquisition and storage of information, and conversion of that into behavior. This memory has several characteristics, which are taken into account in this work [Nuxoll and Laird, 2004]:

- Memories are created without a deliberate decision.
- A retrieved memory is distinguished from current sensing.
- The agent remembers the episode from its perspective, even if not performed by itself.
- The time interval spanned by the memory is not fixed.
- The rememberer includes a time point with the date and hour when the episode occurred.

Individual episodes of experience are essential building blocks for creating a representation of the structure of the world. According to Pause et al. [2013], some factors must be considered to store episodes. One is a novelty, as agents prefer objects they have not seen recently, which can be taken as a measure of temporal order in the memory. As mentioned before, EM can have subjective and accurate time. In this case, we take into account only the objective time to compare events (what happened), place (where it happened), and temporal context (sequence of events) from personal experience.

Pause et al. [2013] propose that how long an episode is stored depends on its emotional association. The proposed rule is that the stronger the emotional activation, the longer the durability of the episode. This durability can also be modulated by factors such as the rehearsal or the number of previous recalls of that episode. It is important to note that extreme emotional activation, such as stress, can disrupt EM function, similar to other types of memory. This hints at how the forgetting process could be implemented in robots. For instance, forgetting could be a good feature for a robot as it would allow it to retrieve information faster when too many episodes where recorded and some needed to be erased. As an alternative to a forgetting implementation, temporal information can be stored as succession or order relative to other events already in memory or reconstructed during recall. A particular event can be stored about or inferred from its occurrence before or after other events. This occurrence is in terms of

preceding or following the event to be specified. The temporal component of an episode can be stored by successively presenting two or more specific events. However, this is not present in the episodes produced by the implemented system but is included in an *Episodic Knowledge Base (EKB)* presented in the later Section 6.2.2.

Retrieval of everyday experiences is fundamental for informing our future decisions. Though the fine-grained neurophysiological mechanisms that support EM retrieval are largely unknown, there are some hints. In that sense, Tulving [2001] mentions that retrieving information from EM (remembering or recollection) is required to establish a particular mental set (retrieval mode). The environment can produce this mental set internally (a thought) or externally. Pause et al. [2013] propose that retrieval depends on the presence of a conditioned stimulus. On the other hand, Wimmer et al. [2020] found that EM has a rapid replay mechanism that can flexibly shift in direction in response to task goals. During episodic memory retrieval, a sequential replay of episode elements depends on success across conditions from distal to proximal elements.

In computational models like the one presented by Deutsch et al. [2008], EM facilitates decision-making. This memory stores experienced situations, and based on the outcome of past situations, the decision-making module can adapt its strategies. This is called *learning by experience*. For example, the work by Winkler et al. [2014] uses EM to store past executions about robot manipulation, including the spatial and temporal context, explicitly. Another example is presented in work by Laird and Mohan [2014], which takes episodes from the WM and stores them in chronological order, also used in this thesis work. EM has also been used in cognitive architectures; this use was presented in Section 2.4.3.

As mentioned, EM stores information about past experiences and is used in robotic platforms to evaluate past executions and then learn from past situations. In this case, a similar approach is used, but the episodes do not only come from robots but also humans. Episodes are used to retrieve information about the experience of others. They are used to improve the SM by combining them with the PM and WM. This thesis work implementation also uses the *monohierarchical relation* [Tulving and Szpunar, 2009], which implies the EM dependency on the SM, see Section 6.2 Figure 6.2. This means that the EM uses the concepts inside the SM to give meaning to the information it includes. One example used in this thesis work is the concept of a sub-action and its difference from an action. As mentioned in previous chapters, a sub-action is above the body movements; in this case, `reaching`, `pushing` and `turning` are sub-actions used by actions, such as `opening` or `closing`.

6.1.2 The use of memory concepts and interconnection models

Memory types are interconnected. Even when their connection is not fully understood, some models are already trying to understand and apply their functionality. Some findings have been implemented regarding the interconnection between human memories and how this is used in computational models. The work by Tulving [2001] proposes a *serial-dependent-independent relationship* between episodic, semantic and perceptual memories. So, they can

obtain knowledge from each other or use only their own. One example of this relationship is the case of SM remembering perceptual information and adding it to the EM. Another example of the interconnection between memories comes during recognition. While it requires EM, it also requires SM and perceptual memory.

People may believe that new facts about the world are learned through experience. This would be modeled by making the directional connection from EM to SM. However, it was found that it is the other way around [Tulving, 2001]. A large amount of learning and storage may occur at the semantic level alone, without the involvement of the EM. This was observed in young children while acquiring knowledge about the world before they could recollect specific information about events in their past. Also, amnesic patients with severely damaged EM can obtain new semantic information in many cases. The interconnection between SM and EM depends on the novelty of incoming information influencing the bottom-up and top-down factors related to the levels of processing. For example, when encountering an unknown object in the instruction, the perception can identify it around known ones, take it from the EM and add it to the SM later.

Some models of memory and its interconnections have been applied to robots mainly to improve their learning capabilities. These models and their interconnections are embedded into a cognitive architecture. Some architectures using various memory models of interest for this thesis work were presented in Section 2.4.3. Another cognitive architecture is given by Alami et al. [2006], which has knowledge about facts from the temporal relations and related known information inside the EM. Its WM provides information about the interpreted situation from the perceived world. The interpretation of this architecture comes from the combination of sensed data with semantic representations.

Another use of EM is presented by Zhu et al. [2017], where the EM supports the narration of plans from high-and-low-level executions for a humanoid robot. The work presented by Tenorth et al. [2010a], used in this thesis work, use memory modules that provide reasoning capabilities for robots to learn from experience. On the other hand, the humanoid robot Intelligent Soft Arm Control (ISAC) [Dodd and Gutierrez, 2005] uses different memory types in parallel, such as STM, LTM and WM systems. ISAC's STM stores information about the environment. Similarly, its LTM includes learned behaviors divided into declarative, procedural, episodic, semantic and perceptual memory. Its WM has task-specific information from the LTM and STM. ISAC uses EM to record sequences from specific events and then learns from them. Its PM holds motion primitives and behaviors required for movement. Finally, ISAC's SM stores data structures about objects. The approach used in this thesis work is very similar to ISAC's; the difference is that the EM does not just record specific events but also includes action and object information. Also, the SM contains more concepts, such as actions and sub-actions.

Other types of memory concepts are also adapted and used in robotics. One example is the work presented by Cruz et al. [2016]. They use associative memory, which supports short-time learning and prediction for future states in a simulated robot performing a cleaning task.

Regarding forgetting, Darwiche and Marquis [2002] define it as a transformation that allows humans to focus or project a theory on a set of variables. Forgetting has applications in planning, diagnosis and belief revision as well. Forgetting can be used to avoid memory saturation; one example is presented by Sigalas et al. [2017]. In this case, the problem regarding storage is solved by merging redundant memories by finding statistical correlations between them. In the case of this thesis work implementation, forgetting is not implemented as such; instead, the statistical correlations of episodes are stored inside an *Episodic Knowledge Base*.

6.2 The use of memory concepts by Personal Service Robots

As mentioned in Section 6.1.1, there is diverse existing research to understand how biological brains work. What is clear and applied is the use of *Semantic Memory (SM)*, *Episodic Memory (EM)*, *Procedural Memory (PM)* and *Working Memory (WM)*, as mentioned in Section 6.1.2. As mentioned before, memory models are interconnected so, they can obtain knowledge from each other or use only their own. One example of this relationship is SM adding perceptual information to the EM to give sense to events. In this thesis work, an interconnected implementation between SM, EM, PM and WM is presented.

In this implementation, an ontology-based on KNOWROB is part of the *Semantic Knowledge Base*. This thesis work uses an *ontology* in *Ontology Web Language (OWL)* and JSON formats to store episodes that the EM can access. The main contribution of this chapter is the introduction of a WM able to receive instructions, acquire the information required to fulfill them, e.g., the action hierarchy, and then send it to a planner. After the execution is completed, the WM uses rules to provide knowledge to the SM and PM, as shown in Figure 6.2. The environment, in this case, can be simulated or physical.

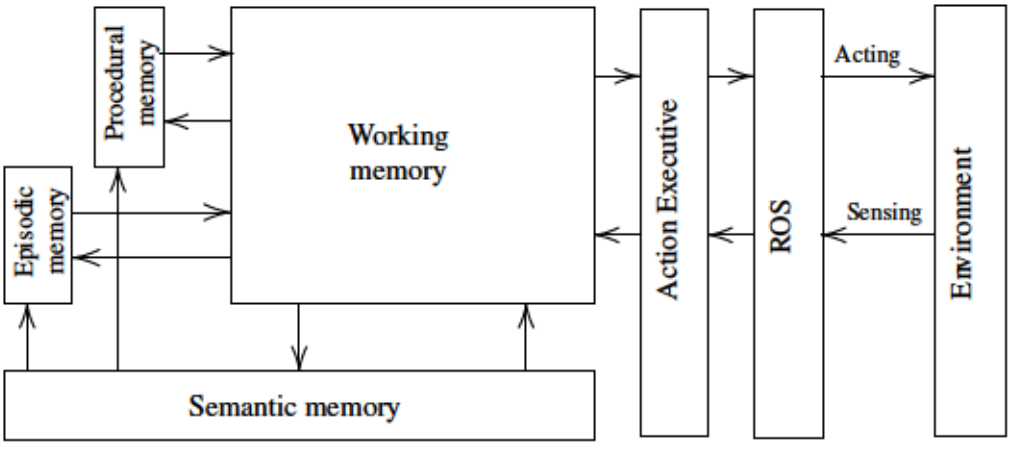


Figure 6.2: Memory architecture for knowledge flow.

This approach uses the EM, PM and SM as part of the *Long Term Memory (LTM)*. Conversely, WM is part of the *Short Term Memory (STM)*. This allows the system to retrieve information via queries from LTM requested by the WM while executing action plans.

To see more about how the different knowledge-processing memory modules work, they are presented next. For each of them, their functionality is introduced.

6.2.1 The Working Memory model

Working Memory (WM) is an *activation mechanism* or filter to decide if knowledge should enter into the other memories. It is also capable of extracting knowledge from *Episodic Memory (EM)*, *Procedural Memory (PM)*, and *Semantic Memory (SM)* to build an action execution structure. For example, extracting spatiotemporal knowledge such as furniture dimensions, cutlery, dishes, etc. This allows the robots to sense space and form while manipulating objects. WM will enable robots to reason about the effects of their actions. WM is used as a processing system to build the environment state using the perception system, in this case, ROBOSHERLOCK.

As mentioned before, *Working Memory (WM)* helps during *planning* in the human brain. Planning combines a set of actions to obtain the desired result. By using this idea, this thesis work implementation uses action features such as their *structure* and *recurrence*. The structure includes the sub-actions in the order of execution. The recurrence counts the number of repetitions of sub-actions. The *plan* is a hierarchy containing action and sub-action pairs obtained from the PM central to successful plan building and execution. Recurrence, on the other hand, is obtained from the EM. For example, when a PSR brings a cold drink to a person, a sequence of sub-actions may repeat more than another. Recurrence is used in case of a specific type of failure during plan execution in which the sub-action can not be performed by any means and another sub-action has to be executed instead. It provides further options for performing sub-actions instead of the one causing the failure. The selection takes the most recurrently executed sub-action first. If the failure continues, it selects the next until no more options exist. Then it gives up.

To build the plan, the process is presented in Figure 6.3, in which a set of actions to be performed `actionSet` is received from the parser to solve a task; see Figure 3.2 in Section 3.2. Parallelograms represent inputs (yellow), outputs (orange), and rectangles processes. Diamonds represent decisions and hexagons `for` loops.

Let's take a look at the example mentioned before of a task to `bring` a cold drink. For that instruction, the actions inside the `actionSet` are `LocatingInIntendedPosition`, where the drinks are, `OpeningADevice` where cold drinks are (most likely a fridge), `PickingAnObject` that is a drink, then follows `ClosingADevice` to avoid waste of energy or future collisions, `LocatingInIntendedPosition` where the robot is to `deliver` the cold drink, and finally, `PuttingDownAnObject` is to give it to the person.

The figure shows how each action in the `actionSet` is searched inside the PKB and EKB in both KBs `ProceduralKB` and `EpisodicKB`, respectively. The `ProceduralKB` contains the action structure, which includes the mostly executed sub-actions for each action; see Section 6.2.3. The `EpisodicKB` contains a compilation of episodes and their action and sub-action pairs; see Section 6.2.2. It looks first inside the `ProceduralKB` to get the action `structure` as the central

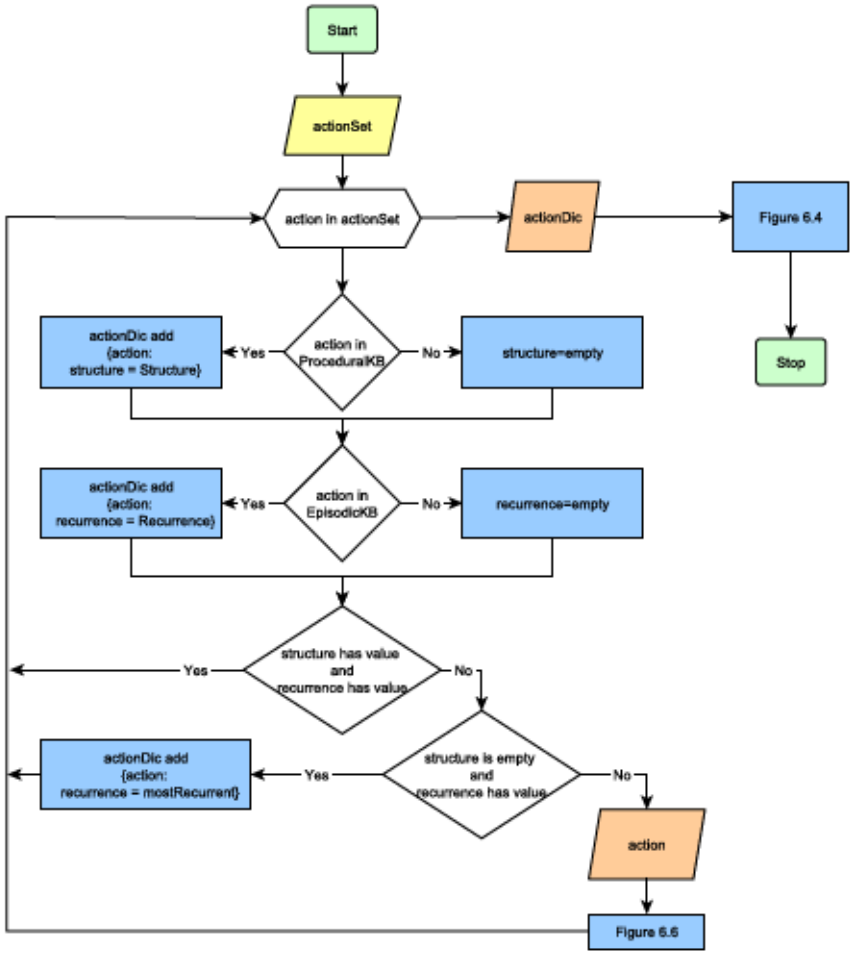


Figure 6.3: Working memory building an execution plan using Procedural and Episodic Knowledge Bases.

part of the plan. Considering the example mentioned before, the structure of an action such as `OpeningADevice` includes the following sub-actions `Perceiving-Voluntary`, `Reaching`, `GrippingAPartOfDevice`, `PullingAPartOfDevice`, `ReleaseGraspOfSomething` and, finally, `RetractingAnArm`. Then, the recurrence of action execution is obtained from the `EpisodicKB` as failure recovery strategies presented later. The `structure` and `recurrence` are then included in an action dictionary `actionDic` used later. This `actionDic` has a plan that consists of a sequence of actions and sub-actions to be executed and measures to recover in case of failure.

The `actionDic` (yellow) is received by the algorithm presented in Figure 6.4, in which each `action` and `subAction` are executed (`execute`) by sending them to the *Action Executive*. Those actions are the output from Figure 6.3. Each `subAction` is represented in the cognitive architecture CRAM; see Figure 3.2 in Section 3.2. The sub-action `type` is verified if it relates to perception; its execution requires further steps, see Figure 6.5. After the plan's structure is created, it can query the remaining information it needs from the EM, PM and SM to send it to the planner on-demand. If the action recurrence or structure is not found, they are set as empty

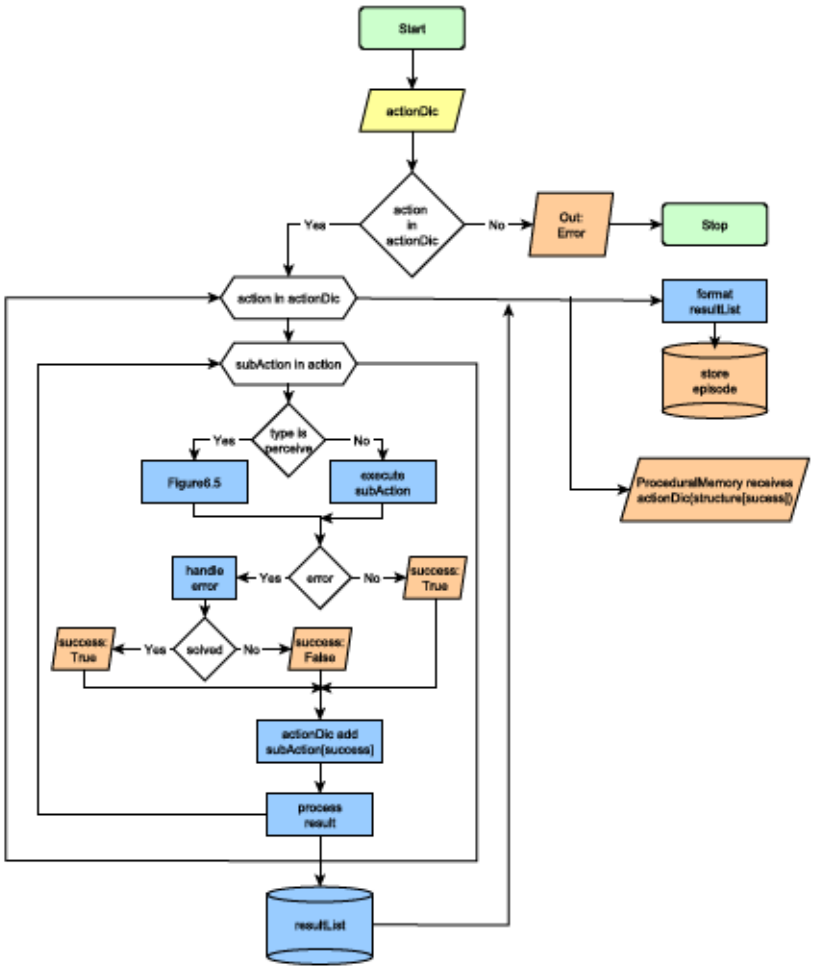


Figure 6.4: Working memory sets actions for execution.

for further processing by the EM presented in Figure 6.6. Otherwise, when the recurrence and structure are found, the process continues to build the execution plan using the `actionDic`.

As mentioned before, there are two ways to handle errors. The first way to handle an error is by using CRAM `handle-failure` strategy, which was not implemented in this thesis work. For example, suppose the PSR can not find the cold drink. In that case, the `perception-object-not-found` error occurs. The recovery strategy uses movement designators to move the head and call the perception system again to find the drink. If the error is not solved, the second error handling strategy is used, which was implemented in this thesis work. In this case, the `recurrence` is used to select a new sub-action to try to solve the error. If this does not solve the error, the system gives up and tries to continue with the following action. In any case, the `success` of the sub-action is stored as part of the execution result in the `resultList`. When the execution of all actions and sub-actions is (both successful and failures), the `resultList` stores a new episode and sends it to the PM to be evaluated and potentially added to the PKB.

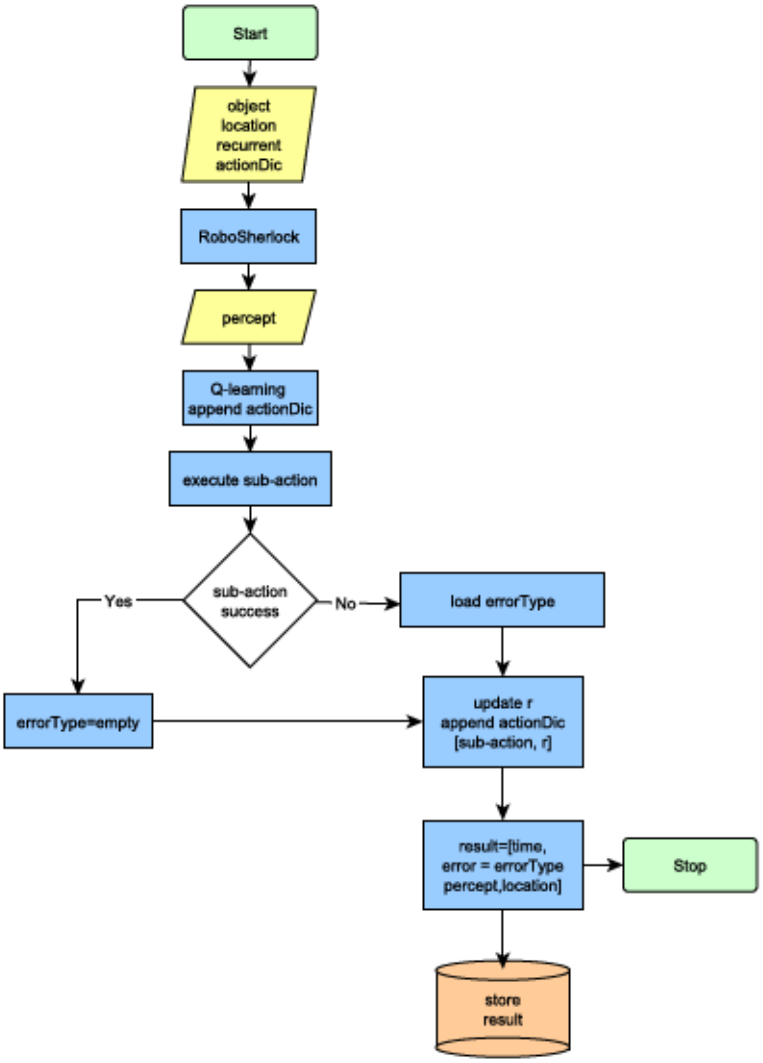


Figure 6.5: Working memory uses perception information.

As mentioned, when the sub-action type is perception, the process followed is presented in Figure 6.5. The algorithm received an `object`, `location`, `recurrent` and `actionDic`. The `actionDic` includes the previous and current sub-actions. The `recurrent` provides for the number of times a sub-action has been executed for each action. It sends an order to the ROBOSHERLOCK system to obtain the perception result as a `percept` that includes any objects seen. Then, it uses Algorithm 3 to select the next sub-action to be executed. For example, if the `object` is close enough, it can choose `reaching`, and if it is still far `approaching`. Then, the sub-action is sent to the action executive to execute it. If there is an error, the reward `r` is negative `r'`; otherwise, positive and the `actionDic` values are updated. Finally, the result of the execution is stored in the temporal storage of the WM for future decision-making. It includes the `time`, `error`, `percept` and `location`. This cognitive framework does not make use of perceptual memory. Instead, WM saves a buffer of perceptual information.

To do so, it uses Algorithm 3, which uses a *Reinforcement Learning (RL)* technique called *Q-learning*; for more details, see Chapter 2, section 2.3.3. This technique was selected as it

Algorithm 3: Working memory uses perception information on Q-learning.

Data: percept with s' and r'
$Q \leftarrow actionDic$
$N_{sa} \leftarrow recurrent$
load s,a,r
if *s != null* **then**
  increment $N_{sa}[s,a]$
  $Q(a,s) \leftarrow Q(a,s) + \alpha(N_s a[s,a])(r + \gamma \max_{a'} Q(a',s') - Q(a,s))$
end
if *last s'* **then**
  $s,a,r \leftarrow null$
end
else
  $s,a,r \leftarrow s', \arg\max_{a'} f(Q(a',s'), N_{sa}[s,a]), r'$
end
return a

learns an optimal policy in an environment with discrete states and actions, like this one [Lewis et al., 2019]. The *policy* converges to an optimal policy quickly and can handle high-dimensional state and action spaces. Even though it does not use knowledge of the underlying transition dynamics of the environment, this is an excellent comparing tool to try new action selections for comparing and updating the PKB. This was the best choice because the environment is discrete, and the state and action spaces are not too large.

As mentioned, Algorithm 3 receives a `percept` with the results from ROBOSHERLOCK. Then `Q` gets the `actionDic` that includes action values and state. N_{sa} gets `recurrent`, which provides for action-sub-action pairs repetitions ordered by action. The values for the previous state `s`, sub-action `a`, and reward `r` are obtained from the `actionDic`. If there are no values, the initial one is equal to null. The current state `s'` is obtained from the `actionDic`. The reward signal `r'` is obtained by Equation 6.1, where the values of γ are chosen randomly between 0 and 1. For further reading on Q-learning, please refer to Chapter 2, Section 2.3.3.

$$r' = \gamma * |r| \tag{6.1}$$

The WM does not just serve as an intermediary; it also stores the acquired information after the action execution finishes. When the execution ends, it keeps track of the information flow to, according to specific rules, verify if the hierarchy is present in the SM, and if not, it can be added. This addition depends on certain factors, as shown in the algorithm presented in Figure 6.10. For example, if there is opposing information between the one already inside SM and the incoming one from WM, an extra round of executions are performed.

In this thesis work, all plans are tested in simulation. After a successful simulation and refinement of the plan, it can be included in the PM. The WM sends some features to the SM to

add knowledge about objects inside the SKB, such as `objectGraspable`, etc. Concepts related to the actions are also allowed to enter information into the SM, presented in Figure 6.10.

6.2.2 The Episodic Memory model

As mentioned before, agents' experience can build representations of the world. In this work, these experiences are stored as episodes. They include the time when events happened, which ones, the place and their sequence. Some episodes come from human demonstrations and others from robots. Demonstrations reduce the necessity of costly trial-and-error by inferring a particular behavior [Puskaric et al., 2017]. In the case of human demonstrations and as mentioned in Chapter 5, action-sub-action pairs are segmented from a *Virtual Reality (VR)* system. The segmented pairs provide information regarding sub-action features, such as velocities, distances, success in limited cases, objects acted on, etc., which are also represented in the *Ontology Web Language (OWL)* format for the high-and-medium-level data and JSON for the raw data. A current implementation that already records episodes from VR [Haidu and Beetz, 2016] is used as a base to record the initial data, which is then processed to obtain action-sub-action pairs. The final representation includes those pairs and the events, including some other sub-action features mentioned before and in more detail in Section 4.2.4. The demonstrations come from a body with different features, so they can not be used directly for imitation. However, the sub-action features can be used during the planning and the actions hierarchy via reasoning using queries.

In the case of robot episodes, a current implementation generates episodes called episodic memories from the robotic executions [Winkler et al., 2014], similar to this thesis work. In this work's implementation, the episodes include the distinction of actions and sub-actions and records features taken from the *Working Memory (WM)* or human examples, which were not included in the other implementations. The robot can improve further when it uses the episode's information for future performance.

Here, the monohierarchical relation mentioned before appears as the *Semantic Memory (SM)* is retrieved to build the episodes by giving concepts required by the *Episodic Memory (EM)*. The EM creates its *Episodic Knowledge Base (EKB)* `EpisodicKB` by comparing previous actions and their results to the most recent ones. This KB keeps an updated version of the best episode results, as seen in Figure 6.6.

As mentioned before, forgetting needs to be implemented in future work. However, the `EpisodicKB` stores a compilation of episodes linked by the action and sub-action pairs. In this KB, the most successful task has higher priority instead of the most current. This serves as a factor to modulate the storage of episodes. Another factor in giving higher priority is the agent executing the task; in this case, the priority is given to the robot.

The retrieval system of EM in animals is still a mystery, but there are some hints, as mentioned before. One main goal of this work is to provide a system with the capability of retrieving episodes of past events to extract information and apply them to obtain insights that will answer questions to perform in current situations. More specifically, it is mainly used to select action

sequencing and trajectory extraction and get other specifications of the actions, such as position and orientation of body parts, velocities, accelerations, distances to objects and level of success. It is essential to make systems capable of learning from their past executions and reaching the point of executing new and similar actions to previously performed ones.

One mechanism to retrieve information from the episodes for further storage and representation is presented in Figure 6.6. The algorithm shows how information about the executed actions is obtained. First, the new `Episodes` are imported, and each `episode` is evaluated. Each action and sub-action passes a series of conditions to verify their features, such as the `type` of action as `handAct`, `cookAct`, `toolAct` or `Act`.

Its result is stored in the EKB `EpisodicKB`. The EM cannot change episodes because they have already happened but can detect some of their features. This process happens every time the robot finishes the execution of a task only for the newly created episodes.

When the WM requires answers from the `EpisodicKB`, it extracts sequences of actions from past experiences similar to Choi et al. [2021] and presented in Algorithm 4. The algorithm computes the relationships between actions. It uses an agglomerative approach of a hierarchical cluster; see Chapter 2 Section 2.3.1 for foundations on clustering.

Algorithm 4: Extraction of actions from episodes.

Data: N actions in episodes
$L \leftarrow emptyList$
$S_0 = \{0, 1, ..., N-1\}$
$Cx \leftarrow$ singleton cluster $\forall x \in S_0$
process dissimilarities between all pairs of clusters D(Cx,Cy)
for $i = 0 \rightarrow N\ \ 2$ **do**
  $(x, y) \leftarrow \arg\min_{(a,b) \in S_i \times S_i} D(Ca, Cb)$
  append (x, y,D(Cx,Cy)) $\rightarrow$ L
  $z \leftarrow \max(Si) + 1$
  $C_z \leftarrow C_x \cup C_y$
  $D(C_z, C_a) \leftarrow F(C_x, C_y, C_a), \forall a \in S_i\{x, y\}$
  $S_{i+1} \leftarrow S_i\{x, y\} \cup \{z\}$
end
return list of N - 1 triplets

Agglomerative hierarchical clustering is a bottom-up approach where each data point initially forms its group, and pairs of clusters are merged based on the similarity measure. This process is repeated until all the episodes belong to a single set. This technique was selected because it is an unsupervised machine learning technique (does not need labels). It can handle a medium-sized and non-linear dataset, which varies in input size (episodes have different lengths), which is difficult for other machine learning techniques. In this case, it identifies potential groups or clusters of similar episodes.

Other machine learning techniques can also handle non-linear relationships, variation in dataset size, and medium-sized datasets. One is *Support Vector Machines (SVM)*, which can handle non-linear relationships and medium-sized datasets [Hearst et al., 1998]. However, if the

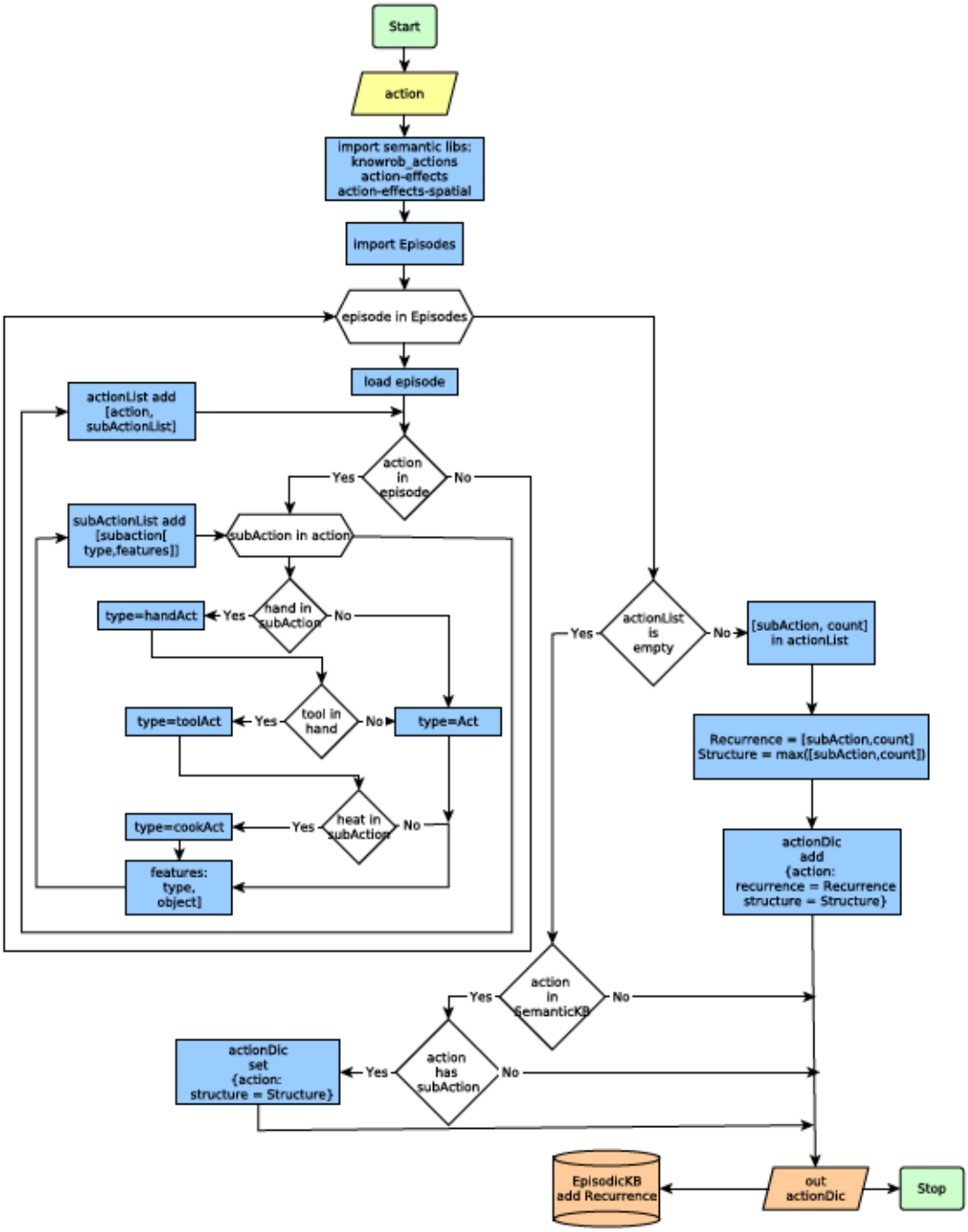


Figure 6.6: Episodic memory asking about specific actions to build recurrent action to the EKB and used by working memory.

variation in input data size is too large, SVM may need help to provide accurate predictions. Another machine learning technique evaluated is *Random Forest (RF)*, as it can handle non-linear relationships and variation in dataset size [Rigatti, 2017]. However, RF is primarily designed for classification and regression problems and not explicitly for detecting similarities between data points. Even though RF can indirectly provide a measure of similarity between data points, it requires using the proximity matrix generated during its construction by measuring the similarity

between pairs of data points based on how often they end up in the same leaf node of a decision tree in the forest. Another technique, such as Gaussian Processes (GPs), can handle non-linear relationships and variation in dataset size and can be helpful for problems with data with high variability [Seeger, 2004]. However, it is a supervised learning technique; it requires labeled data. Another *clustering* machine learning technique is *K-Nearest Neighbors (KNN)*, which can also handle non-linear relationships and medium-sized datasets [Laaksonen and Oja, 1996]. However, it is also a supervised learning method. For all these reasons, the *agglomerative hierarchical clustering* machine learning technique was selected in this thesis work for this specific application.

The *agglomerative hierarchical clustering* approach treats every instance as a `singleton cluster` and merges pairs of the most similar ones until all groups are merged into a single collection. First, the algorithm builds a hierarchical structure of N actions in a set of episodes. It executes $(N-1)$ steps that merge the most similar pair of clusters at each stage. Let a set of cluster indexes at the ith stage be S_i, where $i = 0, 1, ..., N-2$. Then, it initializes S_0 to $\{0, 1, ..., N-1\}$ and C_x to a singleton cluster for all $x \in S_0$. Then, it computes dissimilarities between all pairs of clusters, $D(C_x, C_y)$. For each step i, the algorithm finds the most similar pair of clusters C_x and C_y by $(x, y) = \arg\min_{(a,b)\in S_i S_i} D(Ca, Cb)$. Then, it merges C_x and C_y to C_z by $C_z = C_x \cup C_y$, where $z = max(Si) + 1$. It updates the dissimilarities between the merged cluster C_z and the other clusters as $D(C_z, C_a) = F(C_x, C_y, C_a)\ \forall a \in S_i\{x, y\}$, where F is a dissimilarity update in Equation 6.2. Finally, it updates the set of indexes as $S_{i+1} = S_i\{x, y\} \cup \{z\}$. The output is organized as a *stepwise dendrogram* (diagram of hierarchical relationships), defined as a list of $(N-1)$ triplets

$$(x, y, D(Cx, Cy))$$

such that x and y are the indexes of the most similar pair of clusters at the ith step.

$$F(C_x, C_y, C_a) = \frac{|C_x| * D(C_x, C_a) + |C_y| * D(C_y, C_a)}{|C_x| + |C_y|} \tag{6.2}$$

The first time the WM is executed or if the `structure` is empty in the algorithm presented in Figure 6.3, the one in Figure 6.6 is executed. The algorithm extracts the `type` and other `features` (objects, grasp and hand used) of sub-actions inside actions. The algorithm receives an `action` and verifies if it is included in any `episode`. It also looks for features such as hand used, object involved and type of action. The action types are `Act`, `handAct`, `toolAct` and `cookAct`. One example of an `Act` is `Approaching`, which does not require a hand or tool to be performed. The difference between a `handAct` and `toolAct` is that the second requires an object already in hand to be performed. One example of a `toolAct` is `mixing`, as it requires a tool such as a mixer to be performed. A `cookAct` appears when the heat is used. This type of sub-action allows the execution of intermediate sub-actions when the object is being cooked; for example, while cooking meat, vegetables can be cut.

Prolog Query 8 Find an episode that includes action, object and furniture.

```
1 findall(
2 EpisodeList,(
3 owl_class_properties(Episode,knowrob:Action, Action),
4 owl_class_properties(Episode,knowrob:ObjectActedOn, Object),
5 owl_class_properties(Episode,knowrob:Furniture, Furniture),
6 owl_class_properties(Episode,knowrob:Success, true),
7 EpisodeList=[Episode]),
8 EpisodeList
9 ).
```

After executing the plans using the `features` present in the episodes, it is possible to verify if such knowledge is useful for the robot. In this way, EM helps to enrich the SM with the help of the WM's verification process presented in Section 6.2.1.

In case of failure, a second mechanism is used. This one looks for similarities in the episodes to the current task to be performed. It searches for one successful episode that includes the same `Action`, `Object` and `Furniture` by using the Prolog Query 8. In this case, the time is important when the memory was first created, which means that the most recent episode will be selected. If none exist, it asks the SM which action is similar and applies the same query. This mechanism searches a past episode where the action belongs to the same type or was performed on the same or same type of object.

After selecting one episode, it is sent to the WM to extract more information about how the execution was performed. Then, the system queries the action and sub-action information to execute the task again. This way, the failure might be solved by looking into the previous successful execution and comparing it to the current situation.

In general terms, this *Episodic Memory (EM)* implementation can encode, store, retrieve, and use episodes as other models used in the literature. It mainly relies on the recurrence of actions and sub-actions appearing in the executions.

6.2.3 The Procedural Memory model

As mentioned in previous sections, *Procedural Memory (PM)* stores knowledge related to skills and actions. In this implementation, this knowledge comes from the *Working Memory (WM)* and is stored inside a *Procedural Knowledge Base (PKB)* `ProceduralKB`. PM is updated after every action execution result is received from the WM, as shown in Figure 6.7. This keeps the best behavior for future use when required. The `ProceduralKB` allows fast access to the necessary information by keeping it available during action execution. The behaviors may be associated with the basic skills and habits of the robot.

The first entry to the `ProceduralKB` is obtained by getting the information from the SM, as shown in Figure 6.8. The `ActionList` is obtained by Prolog Query 7; see Section 5.2. Each action is stored in the `ProceduralKB` to look for sub-actions inside the `SemanticKB`. Each

found sub-action is then stored in the ProceduralKB. Sub-actions include properties such as bodyPartsUsed, subEvents, nextMotion. The system looks for the previous (prevAction) and next (nextAction) actions to build the WM's *structure*.

Even if the first entries of the ProceduralKB are not the best, they can be improved with time. The information stored afterward considers stability by considering the execution time and success returned by the WM. This entry is received from the algorithm presented in Figure 6.4 as actionDic(structure[success]) and stored in temp.

Each action actionT and sub-action inside temp is compared to the ones already in the ProceduralKB. Depending on the level of success and if it is the same as the present in the KB, actions and sub-actions are added to the ProceduralKB. Sub-action importance comes from the idea that if a specific sub-action inside an action, like Reaching and Approaching repeats ten times because the robot was too conservative, it would be preferable to store a less conservative one to avoid repeating it too many times.

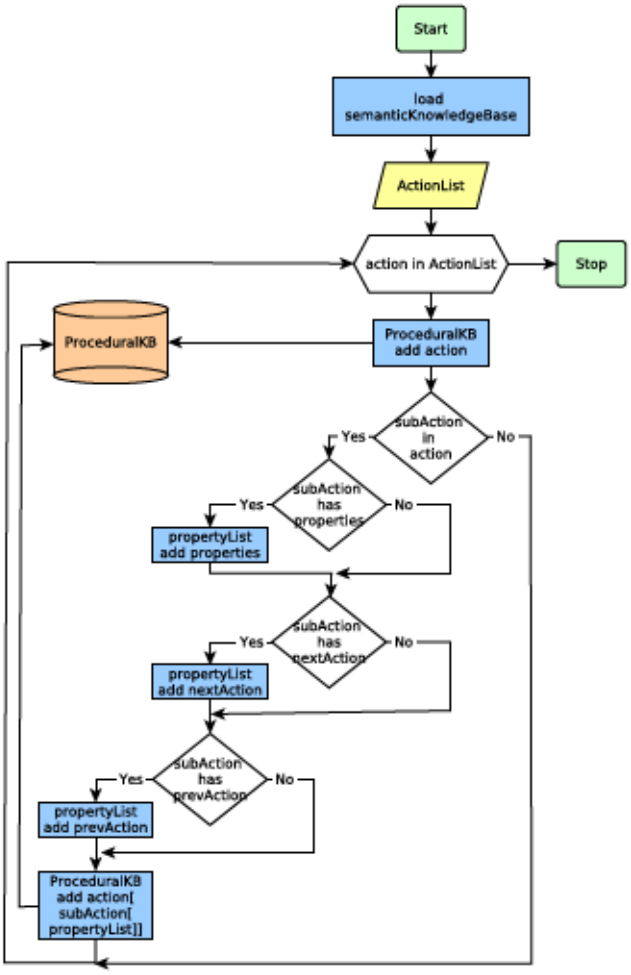


Figure 6.7: Initial Procedural Knowledge Base action structure from Semantic Knowledge Base.

When the WM requires the structure of actions, the PM can build such a structure by querying the SKB and PKB. The SM can also provide information about objects required by the WM. For example, the information can be the parts of the object, its form, etc.

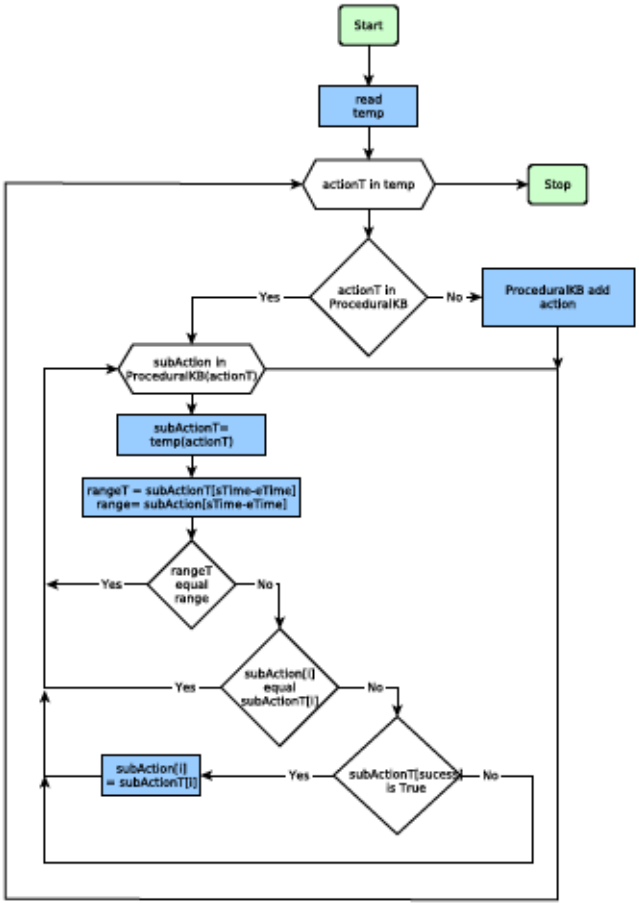


Figure 6.8: Procedural memory adds a temporal action structure to Procedural Knowledge Base.

6.2.4 The Semantic Memory model

As mentioned before, *Semantic Memory (SM)* has general concepts and their relationship to each other. This work relies mainly on Tenorth et al. [2010a] work regarding KNOWROB inside the SKB `SemanticKB`. This thesis work expands the knowledge already present in KNOWROB, as presented in Figure 6.9. There, part of the upper ontology shows temporal and spatial things, actions and sub-actions related to a kitchen scenario. For example, it includes some `HumanScaleObjects` such as `Meat`, `Vegetable` and `CowMilk-Product`, `FoodVessel` and `FurniturePiece`.

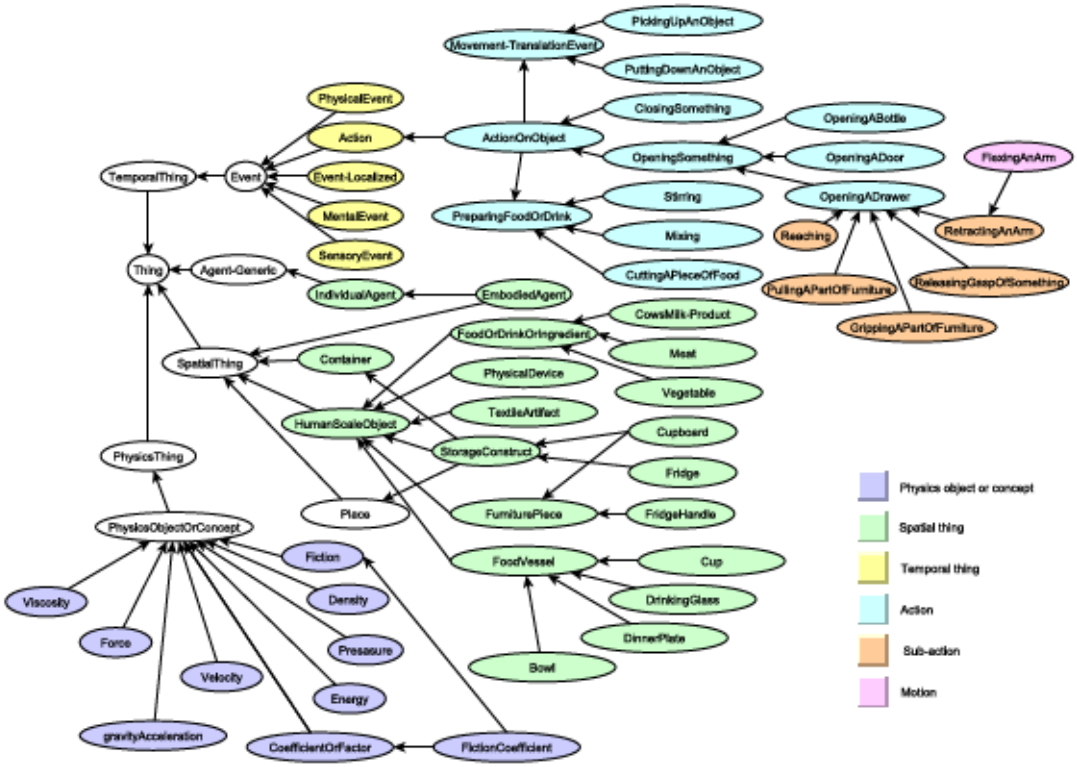


Figure 6.9: Semantic Knowledge Base upper ontology with main classes for physics, temporal and spatial things, actions and sub-actions.

Some knowledge used during the action and sub-action execution from the `SemanticKB` is related to objects. One example is in Algorithm 5. It looks for object features, such as if it is an electrical device `ElectricalDevice` or inside the map `SemanticEnvironmentMap`. For example, if the object is an `ElectricalDevice`, it must have buttons to control its behavior or the robot must be careful while manipulating it. If an object is an `ElectricalDevice` and a `ContainerArtifact` like a fridge, it stores specific types of objects, like milk.

Even when KNOWROB already included many concepts used by robots, it still needs to be completed. In this work implementation, part of the knowledge inside the SKB was added offline by giving the system a set of verbs and nouns. Then, a verification function could verify their existence and changes required; see Chapter 4 Section 4.2.1. However, it is important that

Algorithm 5: Semantic memory retrieves information about object concepts and properties.

Data: SemanticKB
load object
load property
if *object has objectPose* **then**
  append $property \leftarrow objectPose$
end
if *object* $\in$ *SemanticEnvironmentMap* **then**
  append $property \leftarrow inMap$
end
if *object is ElectricalDevice* **then**
  append $property \leftarrow ElectricalDevice$
end
if *object is storage* **then**
  append $property \leftarrow ContainerArtifact$
end
if *object has handle* **then**
  append $property \leftarrow handle$
end
return property

robots are capable of extending their own knowledge by each execution. For this reason, the PM, with the help of the WM, adds concepts regarding actions, events and some of their features to the `SemanticKB`. The representation inside this KB is presented in more detail in Section 4.2.2. New concepts added to the `SemanticKB` consider the last successful execution of actions, as shown in Figure 6.10. Adding an existing concept requires the WM to verify if the knowledge already present is consistent or the novelty of it. In the case of sub-actions, only successful ones are added. In the case of unsuccessful sub-actions, they are present in the EM for error handling and can be used by the WM.

The algorithm uses the action dictionary `actionDic` generated by the process presented in Figure 6.3. Then it verifies if the actions and sub-actions are already members of the `SemanticKB`. If they are not present, they are added. A comparison process is performed to decide which ones to keep if they are present.

Also and as mentioned before in Section 6.2.1, object features related to manipulation are added to the `SemanticKB`, by using the WM perception and execution results in a similar way as the algorithm presented in Figure 6.10.

One example of using semantic knowledge inside the PM and WM can be seen when trying to solve the *pouring action*. In this case, the static friction force can be calculated by

$$F_f = \mu mg \tag{6.3}$$

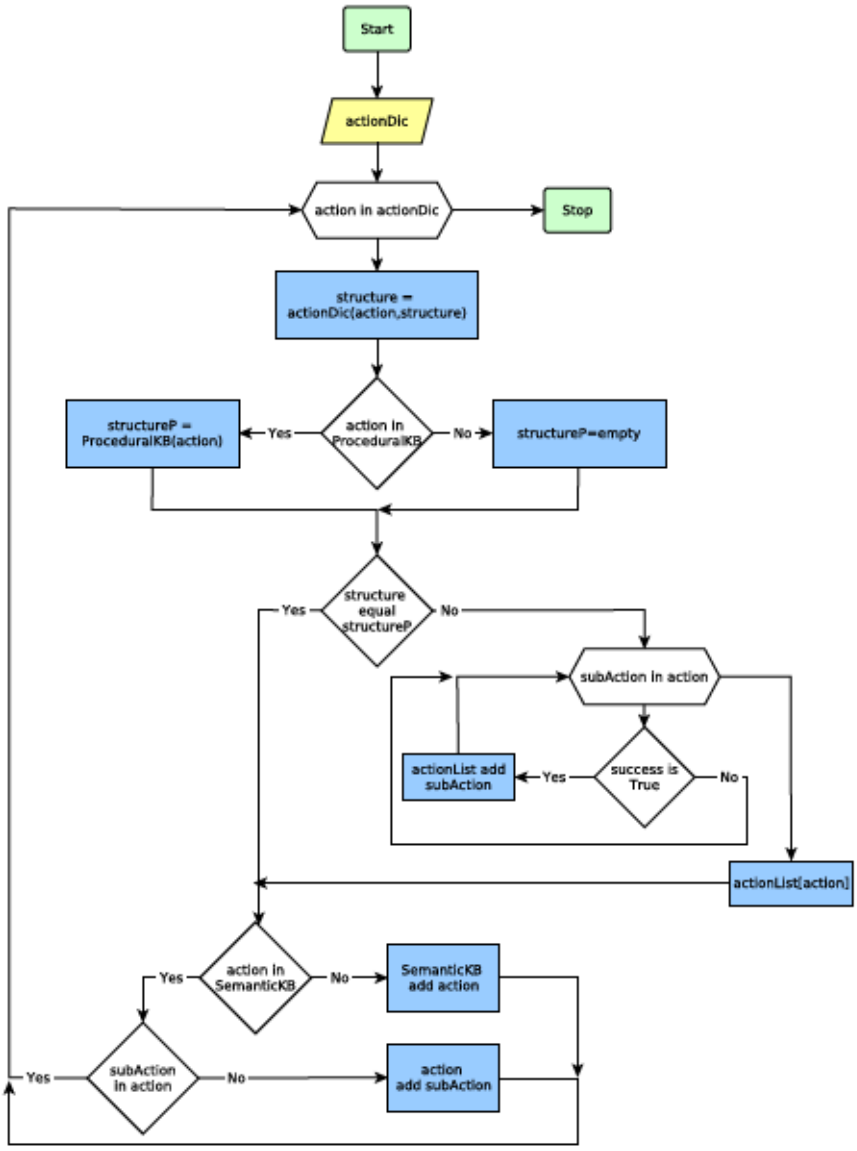


Figure 6.10: Working memory sends the knowledge to the Semantic Knowledgebase.

where μ is the static friction coefficient, m is the mass of the entire container, and g is the acceleration of gravity ($9.8m/s^2$).

In the specific case of pouring a drink from a box, the robot has to know where the liquid will arrive s (Equation 6.4); for that, the fluid theory is applied and added to the *Semantic Knowledge Base (SKB)*, as shown in Figure 6.9. With this addition, the robots can complement their perception capabilities to track the flow of fluid, which was not available inside any other KB to my knowledge.

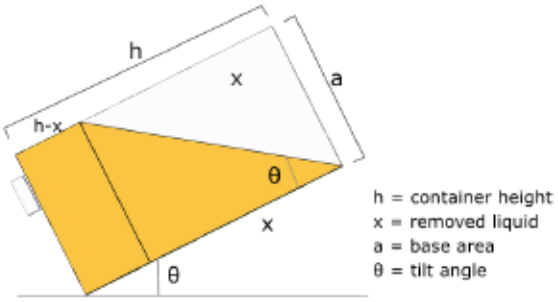


Figure 6.11: Angle for obtaining liquid from a container.

$$s = 2(h * (h - x))^{1/2} \tag{6.4}$$

$$V_f = C_v C_c a(2gh)^{1/2} \tag{6.5}$$

where C_v is the velocity coefficient, C_c is the contraction coefficient of a rounded aperture, and g is the gravity acceleration ($9.8\ m/s^2$).

After calculating the volume flow, it is essential to know the relation of velocity with the angle change. For that, the analysis of forces applied between the container and the liquid can be seen in Figure 6.12. F_v is the viscosity force, N is the normal force and W is the weight, calculated by multiplying the mass by the gravity acceleration. N and F_v are computed by considering the equilibrium (no movement) in Equation 6.6 and Equation 6.7.

$$N = Wcos\theta \tag{6.6}$$

$$F_v = Wsin\theta \tag{6.7}$$

The x component of velocity u is given by

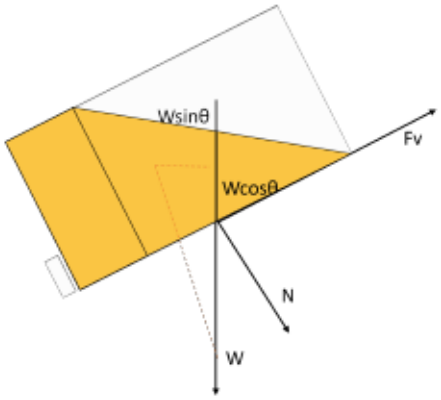


Figure 6.12: Relation of forces of liquid inside the container.

$$u(y) = \frac{V_y}{h} \tag{6.8}$$

where V_y is the speed on the y-axis and h is the height.

Also, the constant of proportionality is the coefficient of viscosity calculated by

$$\tau = \mu \frac{du}{dy} \tag{6.9}$$

where μ is the friction coefficient, du is the distance change in the x-axis, and dy is the distance change in the y-axis. We can then calculate τ by

$$\tau = \mu \frac{V}{h} \tag{6.10}$$

Finally, the total viscous force is equal to the shear stress times the surface area A in contact between the fluid and the bottom surface of the container. Therefore,

$$F_v = \tau A = \frac{\mu V A}{h} = W sin\theta \tag{6.11}$$

Then, the speed V is given by

$$V = \frac{W sin\theta}{\mu A} \tag{6.12}$$

This analysis can also be applied to pouring from a different type of container, such as a bowl, as seen in Figure 6.13.

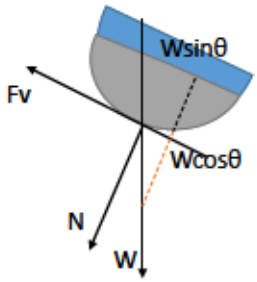


Figure 6.13: Angle for obtaining content from a container.

For this case, Equation 6.5 is not required because the top of the container is wider than the bottom. However, Equation 6.12 can still be applied to different types of containers.

6.3 Testing the memory modules

Some experiments were conducted to test how the memory models perform. All the experiments were run in simulation on the same physical host machine, including three virtual machines on Oracle VirtualBox connected by an internal network; see Figure 6.14. The host computer has a 64 bits Intel Xeon processor, 64 GB in RAM memory and an NVIDIA Quadro M2000 graphics card. One of the virtual machines has two processors and Ubuntu 14.04 with ROS Indigo. The second also has two processors and Ubuntu 16.04 with ROS Kinetic. The third has two processors and Ubuntu 18.04 with ROS Melodic. Each virtual machine runs different packages, compatible with each ROS version. The first virtual machine runs the KBs and robot's models. The second runs CRAM, the knowledge-processing memory models and ROBOSHERLOCK. The third runs Gazebo and MoveIt!. The three of them have to run in parallel to perform the simulations.

All the installing and running instructions, programs, scripts, launch files, object models, etc. are in a package called *learning_from_experience* [1]. Inside this package, there is a package named *robotic_system_simulation*, where the next set of instructions can run an experiment shown in Command Line 9. The values of memory models $< M >$, robotic platform $< R >$, task $< T >$ and number of executions $< N >$ has to be provided. $< M >$ can get the values `SM`, `WMSM`, `WMSMPM`, `WMSMEM` and empty; when the value is empty, it does not execute any memory model. $< R >$ can get the values `pr2`, `romeo`, `fetch`, `hsr` and `tiago`; the value can not be empty. $< T >$ can be set to `pickcb`, `bringd`, `pourd`, `pourcarrys` and `pourmixi`. $< N >$ can get any integer value, which is the number of times the simulations will run. After each task is

[1] github.com/lizyazpin/learning_from_experience

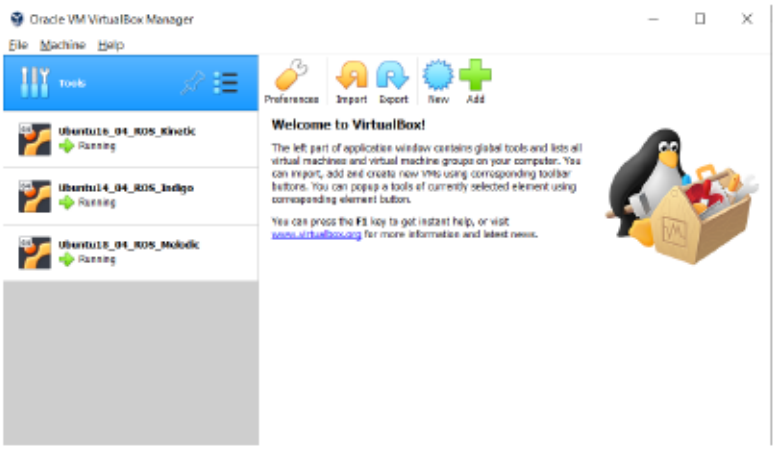


Figure 6.14: Virtualbox with three virtual machines and operating systems running.

Command Line 9 Execute simulation for experiments run.

```
1  roslaunch robotic_system_simulation robotic_system_simulation_memory m
       :=<M>
2  roslaunch robotic_system_simulation robotic_system_simulation.launch r
       :=<R> t:=<T>
3  rosrun robotic_system_simulation run_number_memory.sh <N>
```

finished, all programs are *killed*, episode and KBs stored in hard drive and, if any simulations is missing, the simulation environment restarted.

The simulator used is *Gazebo* [Takaya et al., 2016]. This simulator has a concise physics simulation by utilizing a physics engine based on multi-body collisions and physical properties such as friction or damping. *Gazebo* uses various physics engines for simulating friction. Each object added to the simulation environment includes friction values inside a *Simulation Description Format (SDF)* file. It was selected because it has a precise simulation model for each robotic platform tested in this work. Gazebo offers an interface for spawning, manipulating, and removing objects, making it a flexible simulator. Similarly, MoveIt! [Chitta et al., 2012] was selected for its flexibility as a motion planner. It generates trajectories and monitors the environment for the robotic platforms tested in this work.

In Figure 6.15 the execution flow is shown. There, an instruction comes to the parser, e.g., bring me a drink. Then, the parser breaks it into a set of actions given to the WM. The WM extract the knowledge it requires from the other memory modules and builds the execution plan with action and sub-action pairs. They are sent to the plan executive to execute them. The plan executive returns results, so that the WM can re-plan if required. When the execution finishes, the WM reviews the results and sends specific knowledge to the other memory modules. Then, the robot-labeled episode is generated. The `Episodic DB` indexes this episode, and the EM extracts all critical knowledge. Furthermore, the WM is connected to the SKB to query knowledge directly if needed.

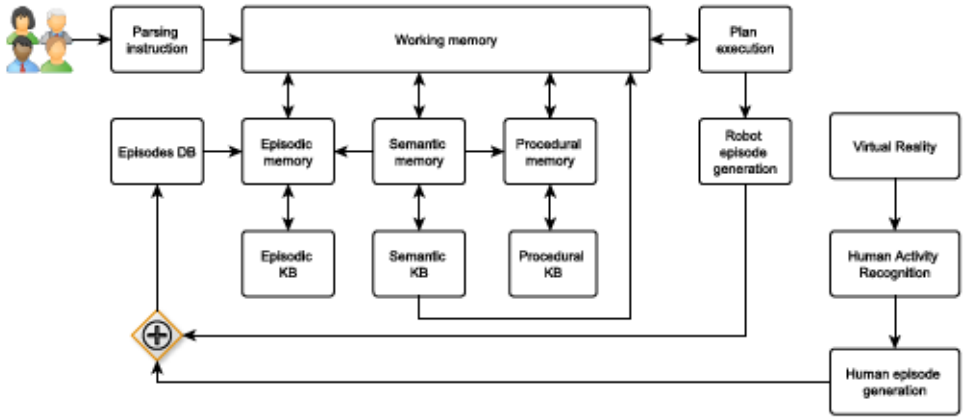
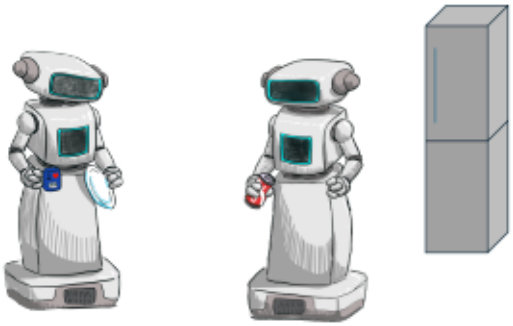


Figure 6.15: Execution workflow.

If a human records a demonstration in the VR environment, the HAR system builds an episode labeled human with the action and sub-action pairs. The `Episodic DB` also indexes this episode. Then, the EM extracts all relevant knowledge when the system is started. This means that the EM is executed at the beginning and end of each system run.

There are five test scenarios; see Figure 6.16. In the first one, the robot Personal Robot 2 (PR2) performs a table `setting` task used as a baseline; Section 6.3.1. In the second test, five robotic platforms `bring` a drink from the fridge to the table; Section 6.3.2. These robots are PR2, Romeo, TiaGo, Fetch and Human Support Robot (HSR). The three following test scenarios are based on the problems introduced in Chapter 1. In the third test, the robots `bring` a bowl of soup to the table; Section 6.3.3. In the fourth test, they `serve` juice into a glass; Section 6.3.4. Finally, in the fifth test, they `pour` and `mix` ingredients in a bowl to create the batter for a cake; Section 6.3.5. In all tests, the objects' initial position is selected randomly on the designated furniture, e.g., on the kitchen island, counter, fridge shelf and pantry.

The first test compares the execution of PR2 with various knowledge-processing memory-models configurations. After comparing the performance of one robot using the knowledge processing memory modules, this thesis implementation was tested on the other four robots developed by industry and presented in Appendix Table A.1, page 179. The experiments were performed 100 times in all the tests because the KBs manage to fill themselves in the middle of experiments and the improvement can be seen in the second half. After each robot execution, the episodes are stored in a specific location to be used in consecutive executions. For each set of simulations, all the episodes are moved to another location and the PKB and EKB are returned to their starting state. The robots use WM, SM, EM and PM. The PKB is empty when each robot starts using it. The EKB only has details from human demonstrations, which are available for the robots. Over time, it fills with robot experience. Each activity has 100 episodes with human demonstrations; see Section 5.2. These activities include actions and sub-actions related to setting up the table for breakfast, lunch and dinner, loading the dishwasher, serving drinks and food, and preparing food, which includes stirring, turning, whisking, cutting and mixing ingredients. The demonstrations



(a) Robot pick and place two objects.

(b) Robot bringing drink to the table.

(c) Robot bringing bowl with soup to the table.

(d) Robot serving juice from container into a glass.

(e) Robot mixing ingredients inside a bowl.

Figure 6.16: Robots performing serving and cooking actions.

are not equal to the tested scenarios present here; they only include the actions and sub-actions performed on various objects.

Both the EKB and PKB are exclusive for each robot. The SKB already has all the concepts mentioned in Figure 6.9. The robots were only tested with all the memory modules for experiments two to five.

The robots used in these tests have different capabilities. PR2 (by Willow Garage) has two-fingered hands (gripper) and Romeo (by SoftBank) has five-fingered hands; both have two arms. Then, the other three robots were tested: TiaGo, Fetch and HSR, which have only one arm with two parallel fingers (gripper). They were developed by PAL Robotics, Fetch Robotics and Toyota, respectively. In this case, noticing the difference between the robots is essential. PR2 has a wheeled mobile base and a gripper. Conversely, Romeo is bipedal and has a five-fingered hand, which is harder to control. In the case of TiaGo, Fetch and HSR, all of them have a wheeled mobile platform and gripper as a hand.

6.3.1 Table setting experiment

Motivation

This test compares the execution of the Personal Robot 2 (PR2) using the cognitive architecture with and without knowledge-processing memory models. It also compares different configurations of the KBs.

Test setting

PR2 `picks` a `cup` and `bowl` from the kitchen counter and `places` them on the table; see Figure 6.17. Either of the objects must be picked and placed to execute the task successfully. The robot executes the task 100 times in the simulation.

Figure 6.17: PR2 brings bowl and cup to the table.

In the scenario, four objects are on the kitchen counter, a cup, a bowl, a plate and a milk box. They are positioned randomly for the first trial and the previously used positions are used for the next simulations. Different knowledge-processing memory model configurations are tested and called cases. In the first case, PR2 performs the task `pick` and `place` with only the *Semantic Knowledge Base (SKB)* but no memory modules or other KBs. In the second case, PR2 performs the task with *Semantic Memory (SM)* and SKB. In the third case, PR2 performs the task with *Working Memory (WM)*, *Semantic Memory (SM)*, *Procedural Memory (PM)*, *Procedural Knowledge Base (PKB)* and *Semantic Knowledge Base (SKB)*. In the fourth case, PR2 performs the task with *Working Memory (WM)*, *Semantic Memory (SM)*, *Episodic Memory (EM)*, *Episodic Knowledge Base (EKB)* and *Semantic Knowledge Base (SKB)*. Finally, in the fifth case, PR2 performs the task with *Working Memory (WM)*, *Semantic Memory (SM)*, *Episodic Memory (EM)*, *Procedural Memory (PM)*, *Episodic Knowledge Base (EKB)*, *Procedural Knowledge Base (PKB)* and *Semantic Knowledge Base (SKB)*. It is important to note that when the robot uses the WM, a reward is attached to successfully executing the task. This means the reward will be higher if both objects are picked and placed.

When the robot uses the memory modules, the PKB is empty and filled while the robot executes tasks. The robot can only access episodes from human demonstrations when EM, EKB, or PM are used.

When the robot uses the three types of memory modules WM, EM and SM. When the experiment starts, the EKB has only human demonstration indexes and general information.

When the robot uses the three types again, *Episodic Memory (EM)* is replaced by *Procedural Memory (PM)*. In this case, human demonstrations are available without the EKB.

The sequence of actions is presented in Appendix Figure B.1 for the cup and the same repeats for the bowl. Also, Table 6.1 shows the action and sub-actions executed during the pick and place task. It includes the execution order of sub-actions completed. Sometimes the cup can be closer to the robot and it will decide to grasp it first and then return to pick up the bowl. No simultaneous hand manipulation is considered, even when the robot has two arms.

Table 6.1: Actions and sub-actions present in the pick and place task.

Action	Sub-action	Variable	Execution order
MovingLocation	Perceiving-Voluntary	Obstacle	1, 17, 25
	FullBodyMovementToLocation	KitchenIsland	2, 18
		Table	10, 26
	ApproachingToLocation	KitchenIsland	3, 19
		Table	11, 27
PickingUpAnObject	Perceiving-Voluntary	RedCup or Bowl	4 or 20
	Reaching		5, 21
	GrippingAnObject	RedCup	6 or 22
		Bowl	22 or 6
	LiftingAnObject		8, 23
	RetractingAnArm		9, 24
PuttingDownAnObject	Perceiving-Voluntary	Obstacle	12, 28
	Reaching		13, 29
	LoweringAnObject	RedCup	14 or 30
		Bowl	30 or 14
	ReleasingGraspOfSomething		15, 31
	RetractingAnArm		16 , 32

Some of the sub-actions repeat multiple times inside different actions. This happens depending on the action needs. Also, some actions can execute more than one time. This can happen for various reasons, e.g., failures and task demands. In this example, it happens because two objects are manipulated.

Results

The performance of the cognitive architecture CRAM with different configurations of knowledge-processing memory modules and without is presented in Table 6.2. The values in this table

considers the 100 executions. Each configuration is presented as a case, as explained before. The total and particular success rate are given. Also, the execution time with its minimum, mean and maximum values are presented.

Table 6.2: Comparison between the use of memory models and no use.

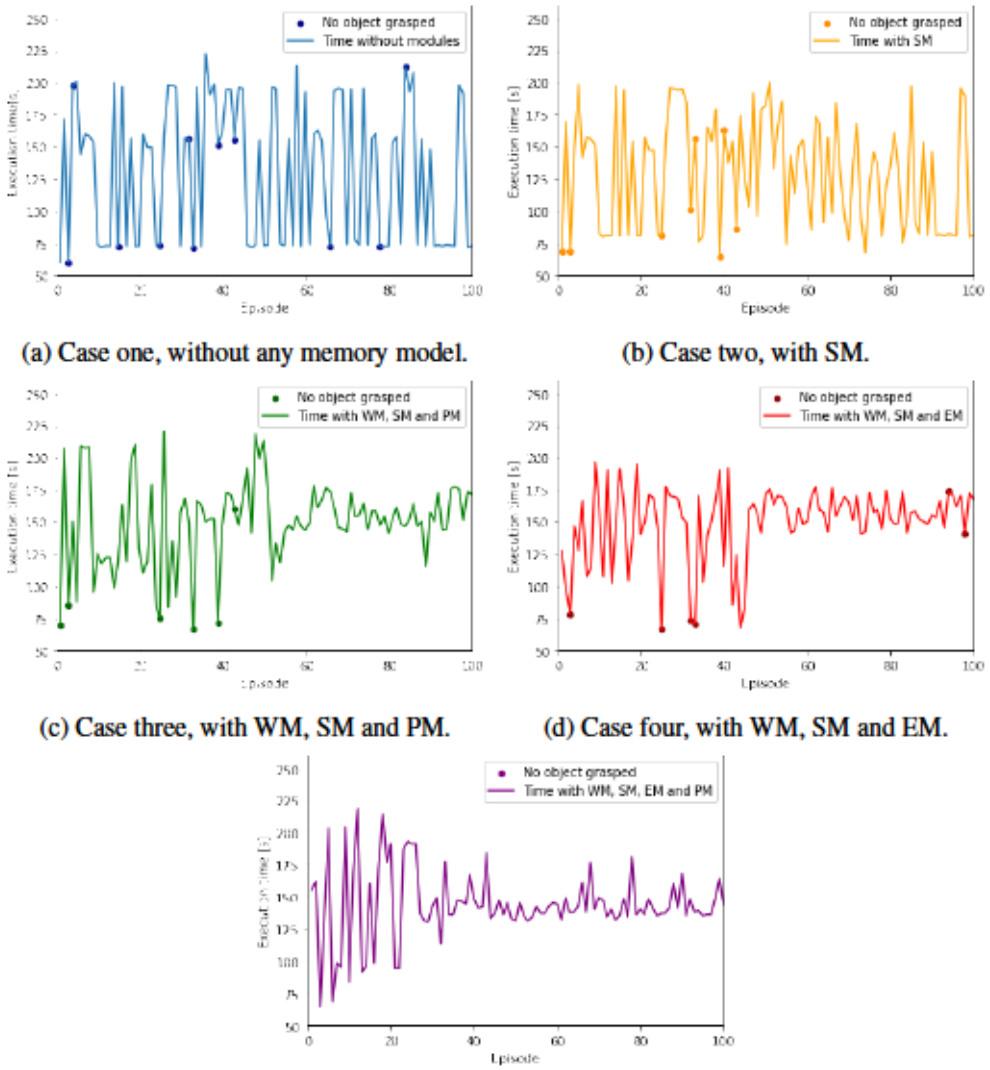
Case	Memory modules	Success rate			Execution time[s]		
		Total	Per object		Min	Mean	Max
			Cup	Bowl			
1	Without	89%	88%	90%	59.62	132.32	222.43
2	With SM	91%	89%	92%	64.50	128.92	200.00
3	With WM, SM, PM	92%	89%	94%	66.50	149.21	220.90
4	With WM, SM, EM	93%	92%	94%	67.60	149.23	196.50
5	With WM, SM, EM, PM	98%	95%	100%	64.00	143.86	218.50

The first case shows the performance results without using knowledge processing memory models, but the SKB. The second case presents the robot's performance improvement in simulation while using the SM and SKB. In the third case, the robot uses three types of memory modules, WM, PM and SM. The results show that the success rate reduces compared with case two but is still better than case one. In the fourth case, when the robot uses again the three types of memory modules, but EM instead of PM, the success rate increases and the execution time reduces. In the fifth case, the robot uses the four memory modules, WM, SM, EM and PM. This case has the highest success rate.

Figure 6.18 shows how the execution time varies in each performance of the simulation. The different graphs show the execution time per simulation. Between case one and case two, there is not much visible difference, There is more visible change when three or four memory models are integrated. Then, the execution time has a smaller amplitude, tending to less variation in execution time.

Looking more into the typical high-level failures, Figure 6.19 shows the total amount for the five cases mentioned in Table 6.2. The first case has no memory model (blue), but with the SKB. The second case has SM and SKB (orange). Another point is using SM, WM and PM only; it includes episodes with human demonstrations but not the EKB (green). Next is the case of using SM, WM and EM only; it uses episodes with human demonstrations and the EKB but no PKB (red). Last, the complete system uses SM, WM, PM and EM; it uses episodes with human demonstrations, EKB and PKB (purple). The results are consistent with previous ones, where the failures decreased when more knowledge-processing memory modules were added.

High-level failures happen when the system could not find a solution for solving the sub-action or action. The system can still try to recover and complete the task. The high-level failures are divided into MoveIt! related, plan related and designator. A designator describes objects and motions; see Chapter 3. MoveIt! related failures are `GOAL-IN-COLLISION`, `MOVEIT-FAILURE`, `NO-IK-SOLUTION` and `PLANNING-FAILED`. Plan related failures are related



(a) Case one, without any memory model.

(b) Case two, with SM.

(c) Case three, with WM, SM and PM.

(d) Case four, with WM, SM and EM.

(e) Case five, with WM, SM, EM and PM.

Figure 6.18: Executing time when executing the pick and place task with and without the use of knowledge processing memory modules.

to location, manipulation and perception. For location, it is called `LOCATION-NOT-REACHED`. In the case of manipulation they are called `MANIPULATION-FAILED`, `MANIPULATION-FAILURE`, `MANIPULATION-POSE-OCCUPIED` and `MANIPULATION-POSE-UNREACHABLE`. And for perception it is called `OBJECT-NOT-FOUND`. Finally, there is one *designator type* failure identified as `DESIGNATOR-ERROR`.

Discussion

In the result, different knowledge-processing memory model configurations were simulated and called cases. The first case shows the performance results without using knowledge processing memory models, but the *Semantic Knowledge Base (SKB)*. The second case presents the robot's performance improvement in simulation while using the SM and *Semantic Knowledge Base*

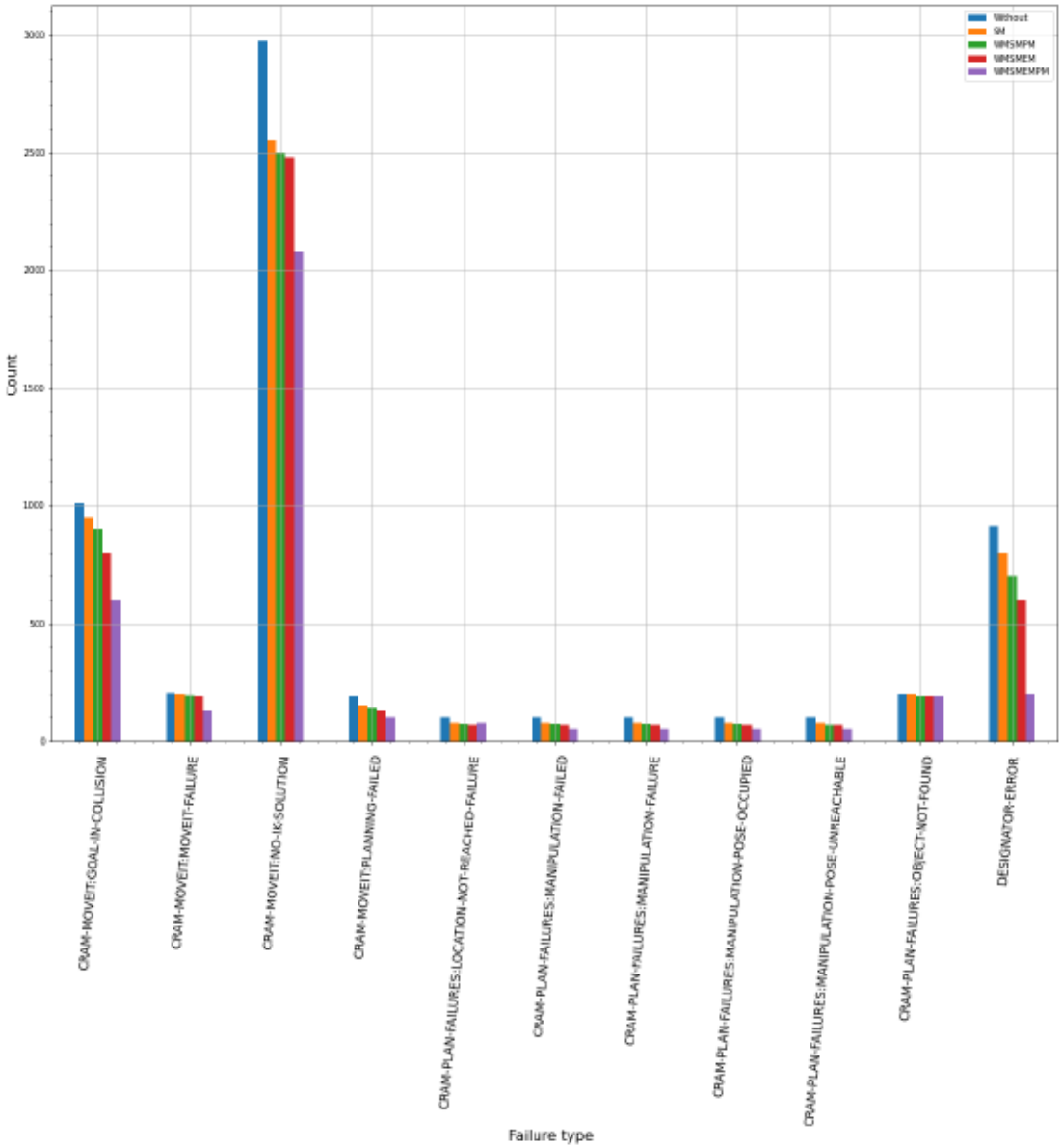


Figure 6.19: Failures presented when executing the pick and place task with and without the use of knowledge processing memory modules.

(SKB). In the third case, the robot uses three types of memory modules, *Working Memory (WM)*, *Semantic Memory (SM)* and *Procedural Memory (PM)*. In the fourth case, the robot uses three types of memory modules, *Working Memory (WM)*, *Semantic Memory (SM)* and *Episodic Memory (EM)*. And in the fifth case, the robot uses three types of memory modules, *Working Memory (WM)*, *Semantic Memory (SM)*, *Episodic Memory (EM)* and *Procedural Memory (PM)*.

The results show that there is already an increase in success rate between cases one and two. There is also a decrease in the execution time. This means that by adding the SM, the PR2 can already improve. This is because the SM is updated after the execution by adding information about the actions and sub-actions executed. By comparing Figure 6.18a and Figure 6.18b, we can see that there is not much visible difference between case one and case two. There is more

visible change when three or four memory models are integrated. Then, the execution time has a smaller amplitude, tending to a less variation and success rate; see Table 6.2.

Then, in case three the success rate keeps and the execution time increases compared to the previous cases (one and two). One explanation is that, as the PKB is empty, it needs to fill its content to improve. This can be seen in Figure 6.18c, where the execution time reduces its variation in amplitude.

In the fourth case, the robot is set to use again the three types of memory modules, but EM instead of PM, the success rate increases and the maximum execution time reduces. However, the mean execution time is still higher than in case one and two. This case is similar to case three, both fill the content of their KB to improve. This can be seen in Figure 6.18d, where the execution time reduces its variation in amplitude. In general, the success rate is better than in the previous cases.

In the fifth case, the robot uses the four memory modules, WM, SM, SKB, EM, EKB, PM and PKB. This case has the highest success rate. Even though the mean execution time is not the lowest, it still reduced compared to case four and three. Again, it has to fill the EKB and PKB, but reaches the decrease of variation in execution time faster than in previous cases; see Figure 6.18e.

In general terms, the proposed knowledge-processing memory modules can reduce failures and reduce execution time. It is crucial to notice that a high number of successes come from queries to the *Knowledge Bases (KBs)* answer the questions about those actions. It is for this reason that the maximum execution time is higher in the case of the use of these memory modules. Initially, the execution depends on the SKB. Over time, the PKB and EKB include more knowledge than before after every execution. Then, the EM and PM access knowledge more quickly.

6.3.2 Bring a drink to the table experiment

Motivation

This test compares the execution of a `bring` a drink from the fridge to the table task between five robotic platforms. It looks into the performance and usability of the knowledge-processing memory models when the task needs to be completely specified. It also looks into how robots with different capabilities solve the same task.

Test setting

One advantage of the plans produced in this work is that they can be reused to solve other tasks. The task tree with actions is presented in Appendix Figure B.3. For this case, the actions `MovingLocation`, `PickingUpAnObject` and `PuttingDownAnObject` are the same as the ones shown before; see Appendix Figure B.1. The difference is the locations used for this action `MovingLocation`, e.g., `Fridge` and `Table`. Also, `PickingUpAnObject` and `PuttingDownAnObject` have the parameter `Drink` for this task, which is a type of object, not a

specific one. The remaining actions and sub-actions, `OpeningADevice` and `ClosingADevice`, required to solve this task are presented in Appendix Figure B.4.

The robots used in this test are first PR2 and Romeo, which have two arms. Later, the other three robots were tested: TiaGo, Fetch and HSR, which have only one arm with two parallel fingers (gripper). PR2, TiaGo, Fetch and HSR have a wheeled mobile base and gripper. On the other hand, Romeo is bipedal and has a five-fingered hand. All of them execute the task 100 times in simulation.

The *Procedural Knowledge Base (PKB)* and *Episodic Knowledge Base (EKB)* start being empty and filled while the robot executes tasks. The robot can only access episodes from human demonstrations when the EM or EKB are in use.

Results

As a result, this difference in the number of manipulators has separated the comparison. To compare the action execution between PSRs, Table 6.3 presents it for two-arm robots and Table 6.4 for one-arm robots. This comparison includes the number of sub-actions performed and the percentage of failures and successes. They all perform the same task, `picking up` a drink from the fridge and `bringing` it to the kitchen table. The actions involved are presented in the task tree in Appendix Figure B.3.

Table 6.3: Comparison between two-arm robots task execution.

Robot name	Mean number of sub-actions	Percentage	
		Failure	Success
PR2	30	5%	95%
Romeo	35	14%	86%

As seen in Table 6.3, Romeo tends to fail more than PR2. However, the number of sub-actions, which takes the mean, is very close.

Table 6.4: Comparison between one-arm robots task execution.

Robot name	Mean number of sub-actions	Percentage	
		Failure	Success
Fetch	40	9%	91%
HSR	43	11%	89%
TiaGo	42	10%	90%

It is important to note that for one-harm robots, a constraint is added where they have to repeat `PuttingDownAnObject` and `PickingUpAnObject` must repeat to close the `Fridge` and not waste energy. Still, as seen in Table 6.4, one-arm robots produce similar results as two-arms. They take longer than two-arm robots and require more sub-actions, as mentioned. Still, all of them can perform this task and `bring` the drink to the table with high success.

Figure 6.20: Robots bring drink to the table results.

Discussion

Table 6.3 shows that Romeo tends to fail more than PR2, but the number of sub-actions is very close. This means that both robots tend to solve the task similarly. These results suggest that the cognitive architecture is transferable to different robotic platforms and the behavior is similar.

Furthermore, as seen in Table 6.4, one-arm robots produce similar results as two-arms. They take longer than two-arm robots and require more sub-actions, as mentioned.

All robots can perform this task and successfully `bring` the drink to the table; see Figure 6.20. This means that the cognitive architecture is transferable to robots with different capabilities. In this case, wheeled versus bipedal and two arms versus one arm were compared.

Romeo and HSR have a success rate of less than 90%. It was seen in the episodes that they tended to have more difficulty `opening` the fridge than other robots.

6.3.3 Serve and bring soup test

Motivation

This test increases the complexity of the tasks to verify the robots' behavior and if, by using the cognitive architecture, they are capable of solving it. It includes the use of tools (`SoupLadle`) to solve the task. The constraints to consider are more than in previous tests, e.g., object orientation, volume, material, mass and shape.

Test setting

In this test, two tasks are involved. The first is to `serve` soup from a `SoupPot` to a `Bowl` using a `SoupLadle`. The second task is to `bring` the full `Bowl` to the `Table`; see Figure 6.16c. Even

though the second task of this test is similar to the one presented in Section 6.3.2, it is essential to notice that the constraints are different, e.g. object orientation and grip force.

For the first task, the robots have to `serve` the soup. The task tree with the actions involved is presented in Appendix Figure B.5. For simplicity, the `SoupLadle` and `Bowl` are next to the `SoupPot` and the pot does not have a `Lid`. The `SoupPot` is on top of the `Stove`. Depending on their volume, the robot must decide how many times to fill the `SoupLadle` to fill the `Bowl`. The robots have access to the physical characteristics of the objects via 3D models and semantic information, such as the weight and material they are made of.

As mentioned, actions can be reused to solve different tasks. In this case, other actions, such as `PuttingSomethingInto`, `CollectingSomethingFrom` and `PouringSomethingInto`, are required; see Appendix Figure B.6. They are relevant for this task specifically but also of interest to this work because they include the effects of the actions `PuttingSomethingInto` and `PouringSomethingInto` between two solid objects, e.g., `SoupLadle` with the `SoupPot` and `SoupLadle` with the `Bowl`. These effects include collision, appearance change and mass change. On the other hand, it is crucial to note the effects of such actions on the `Soup`, as it will have a separation, volume, and mass change, which will be transferred to the `Bowl`. For this execution, `MovingLocation` receives the location `Stove` in its call. In the case of the actions `PickingUpAnObject` and `PuttingDownAnObject`, they receive the object `SoupLadle` to collect the soup from the pot. In this test, the `SoupLadle` is considered to have a mass of 0.30 kg, a friction coefficient of 0.5 and a capacity of 0.17 kg. The robots use Equation 6.3 to decide the force they need to avoid dropping the `SoupLadle`. They have to consider the change of mass after adding the soup. None of the robots has sensors to know the amount of soup they managed to get, so two full `SoupLadles` are considered.

After serving the soup, the robots must `carry` the `Bowl` while keeping it upright and maintaining a maximum restricted acceleration. Task three, in this case, is very similar to Appendix Figure B.1 with the constraints mentioned above. It uses `MovingLocation` to the table and `PuttingDownAnObject` to complete the task. In this test, the `Bowl` is considered to have a mass of 1.76 kg and a friction coefficient of 0.25. The robots use Equation 6.3 to decide the force they need to avoid dropping the `Bowl`. They have to consider how much soup they added in the previous task.

The robots used in this test are first Personal Robot 2 (PR2), Romeo, TiaGo, Fetch and Human Support Robot (HSR), as in the Section 6.3.2.

Results

Table 6.5 presents the mean amount of sub-actions required to solve the `serve` soup task. It also shows the percentage of success for each robot. A sub-action execution is considered successful even if particles went outside the bowl if they are under a threshold of 50 out of 170 particles. In this case, it is considered successful because the robot could transfer the soup from one

container to another. However, the WM reduces the obtained reward as a percentage depending on how much soup was spilled when executing the sub-action `PouringSomethingInto`.

Table 6.5: Comparison between robots serving soup task execution.

Robot name	Mean number of sub-actions	Percentage	
		Failure	Success
PR2	40	8%	92%
Romeo	40	9%	91%
Fetch	45	15%	85%
HSR	48	21%	79%
TiaGo	49	21%	79%

As in the previous test, robots with two arms tend to require fewer sub-actions than the ones with one arm.

Table 6.6 presents the mean amount of sub-actions required to solve the `bring` soup to the table task. It also gives the percentage of failure and success for each robot.

Table 6.6: Comparison between robots bringing soup task execution.

Robot name	Mean number of sub-actions	Percentage	
		Failure	Success
PR2	13	0%	100%
Romeo	13	0%	100%
Fetch	13	2%	98%
HSR	13	3%	97%
TiaGo	13	3%	97%

In general, one-arm robots failed more times as they depended on the grip force they could apply to the bowl.

Discussion

Figure 6.21 shows the results for the five robots. Two-arm robots, namely PR2 and Romeo, used both arms to handle the situation. On the other hand, one-arm robots failed more times as they depended on the grip force they could apply to the bowl. Furthermore, two-arm robots can `hold` the bowl while `pouring` the soup into it. Even though it was considered that all of them could carry the full bowl.

This result was expected, even though manipulating two arms simultaneously can be challenging, so two-arm robots required fewer sub-actions to solve the task.

The objective was achieved. All robots can solve these tasks successfully.

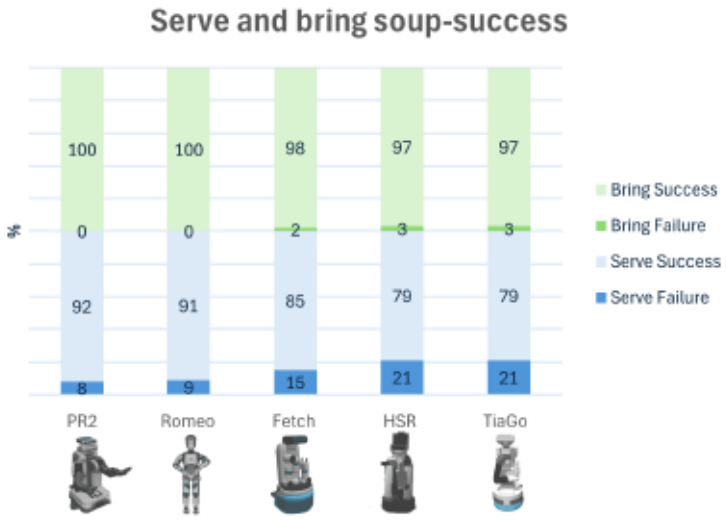


Figure 6.21: Robots bring drink to the table results.

6.3.4 Serve a drink test

Motivation

This test looks into the change in robots' behavior when two objects could be manipulated if possible. It looks into the importance of how manipulating objects can impact the success of solving the task. It also looks into how knowledge can become significant in solving a task where the robot's capabilities are not enough.

Test setting

In the second scenario, the Personal Service Robots (PSRs) served juice from a box container into a glass, see Figure 6.16d. These PSRs are Personal Robot 2 (PR2), Romeo, TiaGo, Fetch and Human Support Robot (HSR).

The `OrangeJuiceBox` is already opened in this experiment. The `OrangeJuiceBox` and `Glass` are already on the kitchen island, and the robot is beside them. The task tree with actions and sub-actions is presented in Appendix Figure B.7.

Two-arm robots can `take` the glass to keep it stable while `pouring` orange juice into it. Furthermore, this task has other constraints presented next.

There are some measures of the friction coefficients for human fingers and rubber gloves on different objects [Gee et al., 2005, Lewis et al., 2007, Abdlkarim et al., 2021], including glass, paper and plastic, among others. These measures were included in the SKB for the robot to decide which force to use in each case. For this test, the friction coefficient used is 1.30. With this value, the robot can calculate the *friction force* to decide the grip force required to prevent the juice container from sliding. The *friction force* F_f is calculated in Equation 6.3, where

μ is the static friction coefficient (1.30), m is the mass of the entire container, and g is the acceleration of gravity ($9.8m/s^2$).

The mass of the juice container considers the density of orange juice as $1.03g/cm^3$ at a temperature of 20 degrees Celsius and considering that $ml = cm^3$, and a mass of 30g for the orange juice container, the mass of a whole orange juice container of 1L is around 1 kg and $F_f = 71.84N$ is the friction force. This value aligns with the human finger forces measured in work by Gee et al. [2005].

The robots must also consider the box's orientation and angular speed while `pouring`. As none of them included sensors to measure the change in weight of the box, they used the juice viscosity to calculate the change of liquid. Since the volume of liquid should be the same before and after tilting the container, the robot calculates it in both conditions to get the angle from that. The assumption is that the volume of liquid inside the container (1 L) and desired outside one that goes to the glass is known to the robot (150 mL). Then, the robot needs to know the angle θ and the time it has to keep the container on (obtained by the volume flow V_f in Equation 6.5); see Figure 6.11. Besides the angle, the robot has to know where the liquid will arrive s (Equation 6.4); for that, the fluid theory is applied and was added to the *Semantic Knowledge Base (SKB)*, as shown in Figure 6.9. Without this addition, the robots would need perception capabilities to track the flow of fluids, which is not yet available to my knowledge.

The volume flow is calculated using Equation 6.5 with a velocity coefficient of 0.97, contraction coefficient of a rounded aperture of 0.97, and gravity acceleration of 9.8 m/s^2.

After calculating the volume flow, it is essential to know the relation of velocity with the angle change. For that, the analysis of forces applied between the container and the liquid can be seen in Figure 6.12. F_v is the viscosity force, N is the normal force and W is the weight, calculated by multiplying the mass by the gravity acceleration. N and F_v are computed by considering the equilibrium (no movement) in Equation 6.6 and Equation 6.7. The flow speed of the orange juice is calculated by Equation 6.12. Here, the friction coefficient is considered to be 0.35 [Telis-Romero et al., 1999] and the mass is 1.25 kg.

Results

After completing the task, the results are presented in Table 6.7. In this case, the execution is considered successful even if some particles go outside the glass if they are under a threshold of 100 out of 10,000 particles corresponding to 1L of orange juice. This is considered so the robot can transfer the liquid from one container to another. However, the WM reduces the reward as a percentage depending on how much juice was spilt when executing the sub-action `PouringSomethingInto`.

In this case, two-arm robots perform more sub-actions because they also pick up the glass.

Table 6.7: Comparison between robots serving juice task execution.

Robot name	Mean number of sub-actions	Percentage	
		Failure	Success
PR2	27	5%	95%
Romeo	28	9%	91%
Fetch	24	14%	86%
HSR	23	12%	88%
TiaGo	22	13%	87%

Figure 6.22: Robots bring drink to the table results.

Discussion

Figure 6.22 shows the results from the five robots. Similarly, as in the previous experiment, two-arm robots take a long time as they use their second arm to stabilize a second object. However, this increases their chances of success. For one-arm robots, ensuring that the juice force will not tilt the glass is more challenging. The robots can calculate the theoretical force and speed of the fluid coming from a box container. However, it is convenient to add sensors for them to measure more accurately the change in volume and weight of the juice container.

It is important to note that the robots could consider their capabilities.

6.3.5 Pour and mix ingredients test

Motivation

This test looks into the cooking tasks of adding (pouring) and mixing ingredients. Similarly to the previous test, it looks into the importance of how the manipulation of objects can impact the

task's success and how knowledge can become significant in solving a task where more than the robot's capabilities is required.

Test setting

In the final test, the PSRs mixed ingredients in a bowl to create the batter for a cake; see Figure 6.16e. These robots are Personal Robot 2 (PR2), Romeo, TiaGo, Fetch and Human Support Robot (HSR).

Similarly to the previous experiment, the ingredients to be added are already on the kitchen island, next to the bowl and whisk. The robot is also next to the kitchen island to avoid navigation. Most actions and sub-actions required for this task were already introduced in previous experiments. They had to be done repeatedly for each ingredient, such as `PickingUpAnObject`, `PuttingDownAnObject`, `PuttingSomethingOnto`, `PouringSomethingInto`. Their task tree is presented in Figure B.6. The additional actions in this experiment are `MixingSolids` and `MixingSolidsAndFluids`. Their trees are shown in Appendix Figure B.9. Both actions have the same sub-actions. However, they must follow different end effector speed constraints.

The objects present are `Butter`, `Sugar`, two medium-sized `Eggs`, `MilkBox` and `Flour`, added in this order. The `Butter`, `Sugar`, `Eggs` and `Flour` are already measured, each in a `Bowl`, and the `MilkBox` is opened. The entire task tree of actions is shown in Appendix Figure B.8.

As in the previous test, the robot has to pour different ingredients inside a specific bowl. For this experiment, the friction coefficients on a glass surface considered are shown in Table 6.8 [Subramanian and Viswanathan, 2007, Claassens, 1959, Astolfi-Filho et al., 2012, Diaz Flauzino et al., 2010, Altuntas and Sekeroglu, 2008, Nwakuba et al., 2019].

Table 6.8: Physical properties of various products.

Product	Mass [kg]	Friction coefficient
Butter	0.125	0.54
Sugar	0.125	0.02
Eggs	0.115	0.20
Flour	0.125	0.35
Milk	0.130	0.16
1L milk box	1.04	0.40
Glass bowl	1.76	0.25

In this case, Equation 6.3 can only calculate the force the gripper needs to `pick` the bowls with the ingredients and milk (Table 6.9), where the milk box has the highest value. However, it does not help to know the ingredients sliding speed while pouring them.

The milk is similar to the juice so we can use Equation 6.4 and Equation 6.5 to calculate the position and volume flow for obtaining the 0.130 g of milk.

Table 6.9: Friction force of various objects.

Object	Friction force [N]
Bowl with butter	4.62
Bowl with Sugar	4.62
Bowl with eggs	4.59
Bowl with flour	4.62
1L milk box	16.56

For the case of the ingredients inside the bowls, the robot has to calculate the speed and angle to break the static force for them to come outside the container. It can consider the forces present in Figure 6.13. As mentioned in the previous experiment, F_v is the viscosity force, N is the normal force and W is the weight, calculated by multiplying the mass by the gravity acceleration.

N and F_v are calculated by using Equation 6.6 and Equation 6.7.

As in the previous experiment, the robot can calculate the speed change in relation to the angle change with Equation 6.12.

Results

Table 6.10 shows each robot's average number of sub-actions and percentage of success during the `pouring` ingredients task. The task is considered successful even if particles go outside the bowl under the threshold of 70 out of 300 particles.

Table 6.10: Comparison between robots pouring ingredients task execution.

Robot name	Mean number of sub-actions	Percentage	
		Failure	Success
PR2	87	8%	92%
Romeo	87	10%	90%
Fetch	90	19%	81%
HSR	92	17%	83%
TiaGo	91	18%	82%

Table 6.11 shows each robot's average number of sub-actions and percentage of success during the `mixing` ingredients task. The task is considered successful even if particles go outside the bowl under the threshold of 70 out of 300 particles.

Because of the complexity of the task, the WM does not reduce the reward in this case. Still, in general terms, the success is lower than in previous tests.

Table 6.11: Comparison between robots mixing ingredients task execution.

Robot name	Mean number of sub-actions	Percentage	
		Failure	Success
PR2	80	6%	94%
Romeo	83	10%	90%
Fetch	90	16%	84%
HSR	92	14%	86%
TiaGo	91	15%	85%

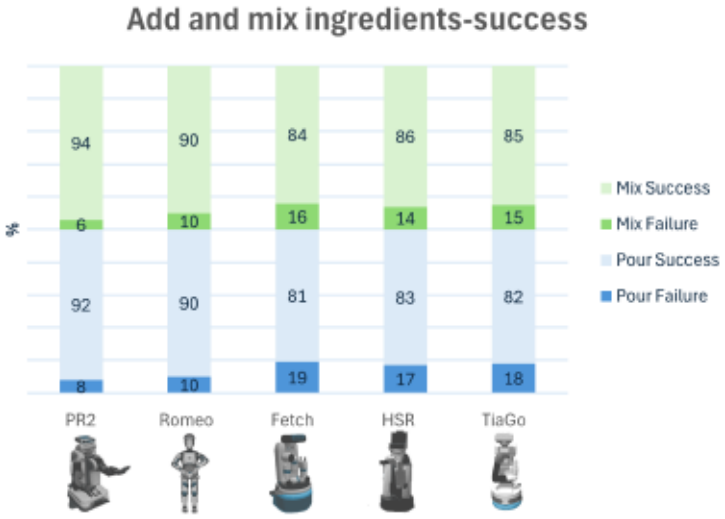


Figure 6.23: Robots add and mix ingredients for a batter results.

Discussion

Figure 6.23 shows the results from the five robots `pouring` ingredients from different bowls into a bigger one and `mixing` them. Similar to previous tests, one-arm robots have less success than two-arm ones. Even though one-arm robots mix ingredients slower, they still must keep the `MixingBowl` stable and avoid spilling the batter. However, they can perform the task by adapting the plan to their capabilities. This shows that the extension of the cognitive architecture does provide more tools for robots to execute most of the time successfully. This is because all results are above 80% success for the five robotic platforms in both tasks.

Similarly, when `pouring` a liquid into a glass experiment, two-arm robots can hold the `MixingBowl` to keep it stable and increase their whisking speed. One option that one-arm robots do not have. This has a consequence that depending on the stability of the bowl, a one-handed robot uses less speed than a two-handed robot, which can keep the bowl stable with the other hand.

6.4 Summary of this chapter

This chapter presents concepts from human memory, including the types used in this thesis work. Here, also some applications are presented in computational models and robots.

This chapter also presents the way memory concepts are implemented in robots. The *Working Memory (WM)* performs the main link between the different memory types and the environment. Its different processes are presented here, including its algorithms. For the *Long Term Memory (LTM)*, *Procedural Memory (PM)*, *Semantic Memory (SM)* and *Episodic Memory (EM)* models are implemented. The PM implementation is presented where the robot's behaviors are stored in a *Procedural Knowledge Base (PKB)*. The implementation of the EM is introduced where the robot's experience regarding actions is stored inside an *Episodic Knowledge Base (EKB)*. The SM implementation is presented where the methods to retrieve and store knowledge related to concepts are present. The *Semantic Knowledge Base (SKB)* uses mainly the KNOWROB implementation. Furthermore, new knowledge is added to increase the robot's abilities.

This implementation allows a Personal Service Robot (PSR) to perform complex manipulation tasks in house environments and increase its knowledge for future executions. This work can perform *Learning from Demonstration (LfD)*, acquire knowledge, and simultaneously increase its own. Still, future work needs to include an implementation for forgetting. However, the KBs are updated after every action execution.

This implementation was tested on five Personal Service Robots (PSRs) with different capabilities. These robots are Personal Robot 2 (PR2), Romeo, TiaGo, Fetch and Human Support Robot (HSR). PR2 (by Willow Garage) has two parallel fingers hand (gripper) and Romeo (by SoftBank) has five-fingered hands; both have two arms. TiaGo, Fetch and HSR have only one arm with two parallel fingers (gripper). They were all developed by PAL Robotics, Fetch Robotics and Toyota, respectively.

They were tested on five scenarios: table `setting`, `bringing` a drink from the fridge to the table, `serving` and `bringing` a bowl of soup to the table, `serving` juice into a glass and `mixing` ingredients in a bowl to create the batter for a cake. They completed all the test tasks successfully in more than 80% of the cases.

Chapter 7

Conclusions

A cognitive architecture is a framework that describes the basic cognitive processes and structures involved in intelligent behavior, such as perception, attention, memory, learning and reasoning. By implementing cognitive architectures in robots, we aim to create more intelligent and adaptive machines that can perform tasks successfully in complex and dynamic environments.

The cognitive architecture allows robots to perceive and interpret sensory information more effectively by modeling the different stages of perception, such as feature extraction, pattern recognition, and object recognition. It also allows them to focus on relevant information and filter out irrelevant information by modeling attentional processes and mechanisms, such as salience detection, selective attention and cognitive control. Furthermore, it allows robots to store and retrieve information more efficiently by modeling different types of memory. In this case, *Working Memory (WM)*, *Procedural Memory (PM)*, *Episodic Memory (EM)* and *Semantic Memory (SM)*. The robots can then learn from experience and adapt to new situations by modeling different types of learning, such as reinforcement learning, supervised learning, and unsupervised learning. That way, they can reason about the world and make decisions based on uncertain and incomplete information by modeling different types of reasoning, such as deduction, abduction, and induction.

The cognitive architecture allows robots to perform tasks successfully in complex and dynamic environments, such as navigating in unknown terrains, manipulating objects in cluttered spaces and interacting with humans in social settings. However, it is important to note that creating effective cognitive architectures for robots is still an active area of research and there are many

challenges and limitations to be addressed, such as scalability, adaptability, and robustness to noise and uncertainty.

In this thesis project, cognitive knowledge processing capabilities using memory models were presented. They are used by Personal Service Robots (PSRs) to decide if and how manipulation actions should be adapted by integrating and expanding knowledge to their execution. These robots can be used as tools to perform household tasks that are required or desired. They can also serve as assistants in caregiving tasks.

This work presents knowledge-processing models based on four types of memory in the human brain; see Figure 7.1. These models extend the cognitive architecture CRAM for PSR to perform manipulation actions in the context of cooking. These knowledge-processing memory models use neuroscience and psychology concepts, allowing PSRs to perform complex manipulation tasks in a human-like way. This way, PSRs can acquire, store and process knowledge from actions. This allows them to improve their performance. With this extension, the cognitive architecture CRAM can now handle more types of actions and failures. Robots using it can recover faster than in the past; see Table 6.2.

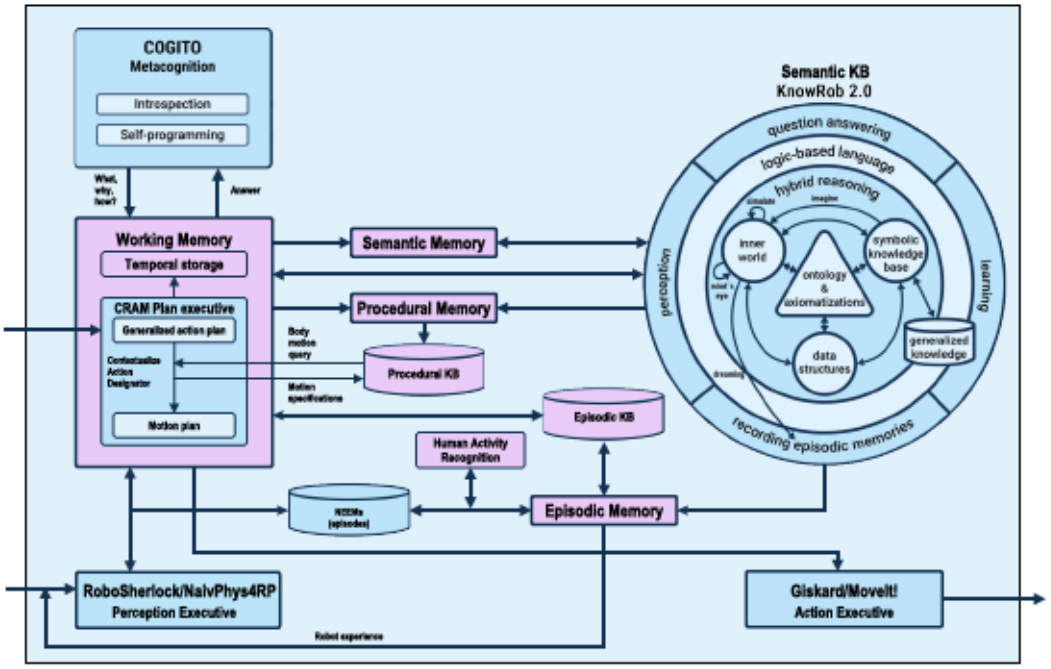


Figure 7.1: Extended CRAM with Memory-based knowledge processing modules.

The implementation made for this thesis work is presented next.

7.1 Human Action Recognition

This work provides a HAR implementation using two different machine learning techniques to classify actions and sub-actions. The first is a decision tree, which separates possible actions into classes depending on certain conditions, such as the speed of the hand and/or head and distance from the object. The results of this implementation have a mean accuracy of 92%, being the lowest 60% and 100% the highest, in the recognition of sub-actions in the sequence performed by humans (see Figure 5.6 of Section 5.2.2). The second one is a RNN which takes samples of the trajectories and classifies them accordingly. The results of this method have a mean accuracy of 96.93%, being the lowest 91% and 100% the highest. The results of the combination of these methods have an accuracy higher than 97% in all sub-actions and actions performed by a human; see Figure 5.8 of Section 5.2.3. The implementation can differentiate similar action trajectories, such as reaching for an object and retracting a robotic arm. Even though, it still has limitations in identifying correctly 100% of the times the `ReleasingGraspOfSomething` sub-action and when there is no action. The system has an accuracy of over 90% in all sub-action identification.

When a sub-action is added to an action, it depends on the previous association inside the KB or a manual addition. A different classifier can be added in the future to avoid this issue, but it might modify the existing knowledge. For this reason, a selector and evaluator will also be required to decide automatically if that change is needed.

So far, the system can only handle known actions. In the case of unknown action, more examples from human executions can be integrated into the system, but they require to be pre-processed beforehand by the HAR system mentioned before in the previous section and in more detail in Chapter 5.

7.2 Knowledge processing modules

As mentioned before, robots need advanced cognitive capabilities to predict where, how and why to manipulate objects by using and expanding knowledge for future executions. This thesis work shows how five different robots achieve the manipulation of prior knowledge and produce new. In this sense, they generated a well-defined plan structure by using prior knowledge and storing current knowledge for future executions. They did not only call the respective sub-actions at specific times but determined dynamically when and how many times to execute them. This included the preparation for future executions, such as the locations where robots stand or place objects in visible locations from the robot's perspective. For example, not to place objects at locations where they are occluded from others which are needed later. More specifically, they created high-level plans which not only called the right components in the proper order but also found parameters to prevent errors.

As mentioned earlier, PSRs use memory modules that make them capable of looking for similarities in the structure of actions. Then, they build and execute plans. These plans can be

simulated and evaluated before being executed in the real environment. In this thesis work, they were only simulated. This is because some of the robots tested in this work were not available physically. However, the transfer to real robots can be done in a short time. In the case of other types of failures, the WM uses stored knowledge from previous executions to compare both cases and then decide how to change the plan, if necessary.

This extension allows the robot to perform tasks and manage knowledge as required, i.e. `acquire`, `store` and `find`. With this work, PSRs, with their cognitive architecture, can accomplish vaguely specified tasks such as `bringing` a drink by identifying which drinks are available and then deciding how to `grasp` a bottle, open it and `transport` it. To accomplish that, robots obtain the missing information from different sources, such as internal (inferring) and external (perception systems) methods. Memory is not seen as a repository of sensed data but as an active process that transforms factual knowledge into linked structures.

There are four different types of memory involved in this work. Knowledge is stored inside the correct type of memory to then be managed in a human-like way.

This model uses the *Working Memory (WM)* interface to compare current and previous action executions. It functions as a memory manager based on a model from cognitive science. WM serves as an interlocutor between the environment and other memory modules. This memory model retrieval can be accomplished in an accurate and fast manner. In case of failure, it uses prospection to simulate possible actions that can solve the issue. The WM is largely automatic while managing goals, plans and task execution.

Episodic Memory (EM) has a retrieval system for specific experiences and an analysis system of previous executions. *Procedural Memory (PM)* retrieves knowledge of previously executed actions similar to the learned skills. *Semantic Memory (SM)* retrieves concepts about objects, materials, places and actions. Each memory module has an evaluation method to retrieve the relevant knowledge for the current context.

It is important to note that the *Episodic Knowledge Base (EKB)* and *Procedural Knowledge Base (PKB)* are exclusive to each robot. Even when all robots can use human demonstrations, each has its own experience stored in the EKB. Similarly, their physical capabilities define the best way to execute an action. These new execution details are stored in the PKB. When a robot executes a task and fails, it can use human demonstrations to modify its execution to try again with different parameters. This allows the robot to try executions in additional ways until the best one is found.

7.2.1 Working memory

Working Memory (WM) processes factual knowledge into linked structures. This memory model compares current and previous action executions to provide knowledge about how to act. WM chooses the best course of action by computing the probabilities of success based on previous episodes present in the *Episodic Memory (EM)* and *Procedural Memory (PM)* modules.

The execution plan includes actions and sub-actions selected based on relevance. This is done by checking pre- and post-conditions from each of them. The action and sub-action selection also depends on a score given to the pair depending on their contribution to achieving the current goal. The previous success rate gives the score by *Reinforcement Learning (RL)* implementation to improve the robot's behavior in later executions.

WM decides if knowledge should enter the other memories. It keeps track of all the flow of information from the environment and internal KBs. This allows robots to perform actions depending on the world state in a limited time interval. The knowledge includes information about dimensions, shapes and materials. This allows the robots to have a sense of space and form when interacting with objects.

As mentioned before, WM has a link with the LTMs and the current state of the robot. This link helps to process the incoming information and then transform it into knowledge to be sent to the corresponding memory. WM stores the information during execution time, and after processing, it is erased.

7.2.2 Episodic memory and knowledge base

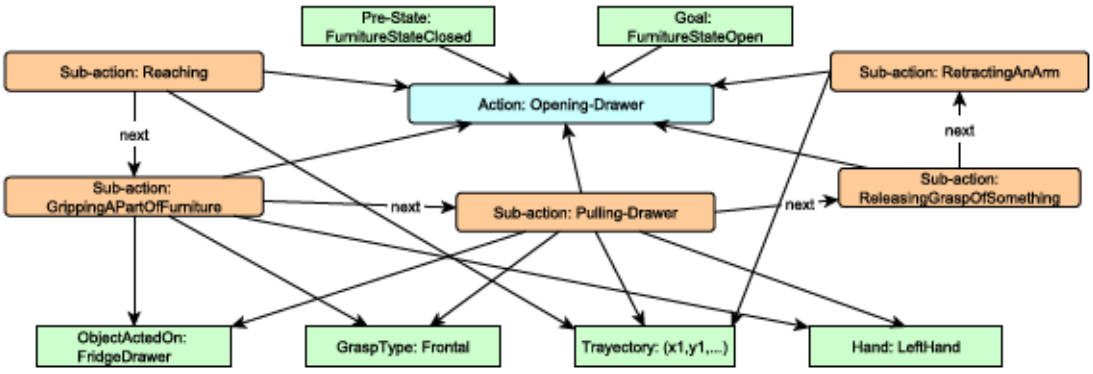


Figure 7.2: Episode representation example.

The *Episodic Memory (EM)* model analyzes produced episodes and stores their defined insights. It is executed after a new episode is created after the task is finished.

Episodes include details about performed actions and events during a task. An example of the representation inside an episode is presented in Figure 7.2. There is represented the action `Opening-Drawer` with an ordered sequence of sub-actions comprising `Reaching`, `GrippingAPartOfFurniture`, `Pulling-Drawer`, `ReleasingGraspOfSomething` and, finally, `RetractingAnArm`.

Episodes store sub-action and action pairs in a low- and high-level hierarchy. Part of this knowledge is used to extend the created *Episodic Knowledge Base (EKB)* with previous experiences. One example of the knowledge included in the EKB is presented in Figure 7.3. Their actions (blue) and sub-actions (orange) are represented with their *global features* that track the performance of the agent (dark green), and their *local features* track specific performance execution

information. For example, the successLevel verifies the percentage of all successRate for each sub-action executed over time. It is a global measure of success. The successRate is measured by the number of failures against successes and if they were solved.

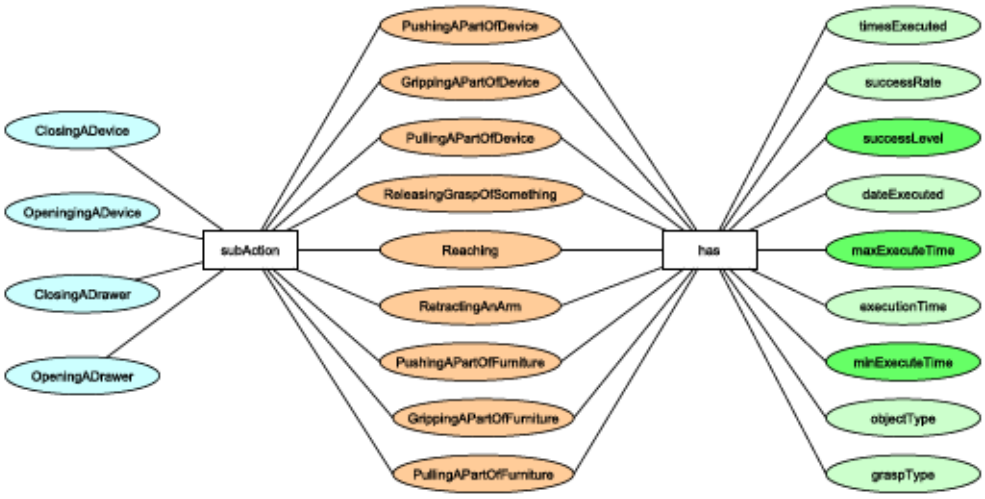


Figure 7.3: Episodic Knowledge Base global and local features.

In the future, a complete system can be added to the EM model, so it has other methods to solve failures or search for episodes faster. For example, an implementation that applies similarity in the search and select actions that are more likely to be executed next when certain conditions are met. Also, a forget implementation can be added to the system, so episodes are not permanently stored, and memory can be freed.

7.2.3 Procedural memory and knowledge base

The *Procedural Memory (PM)* model verifies which actions already have a sequence in its *Procedural Knowledge Base (PKB)* and stores new ones. An example of the knowledge inside the PKB is presented in Figure 7.4. Some verbs are represented, e.g. *reach* as Reaching, *release* as ReleasingGraspOfSomething, *push* as PushingAPartOfDevice, *pull* as PullingAPartOfDevice, *retract* as RetractingAnArm and *grip* as GrippingAPartOfDevice. Also, some features are represented here, such as previousAction (dashed and dark green), nextAction (continuous and light green), positionInExecution (continuous and yellow) to build the sequence in which actions and sub-actions are executed. Some of them have extra features such as objectActedOn (dashed and dark blue) and graspType (turquoise), which are also represented. To complement the knowledge stored in the PKB, *grasp postures* are added in the graspType (continuous and black) feature besides trajectories.

PM is capable of verifying inconsistencies in the sequences. However, it gives preference to new sequences. For this reason, it still requires a complete verification process to decide which sequence to take.

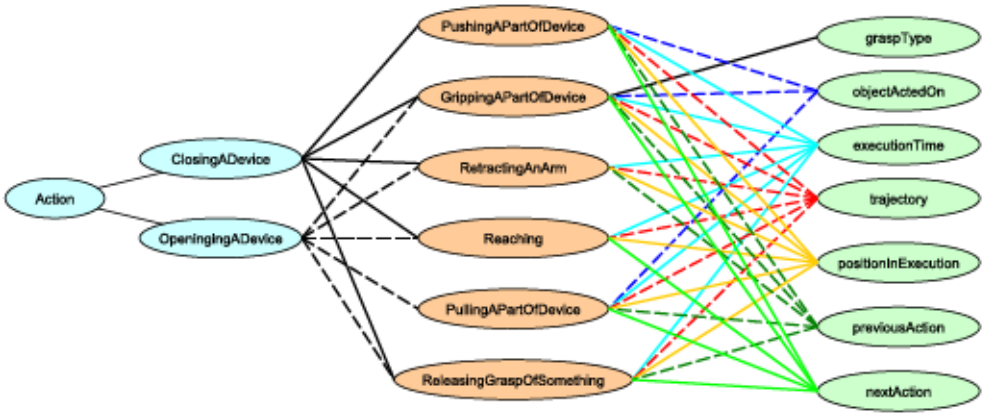


Figure 7.4: Simplified example of actions and sub-actions present in the *Procedural Knowledge Base*.

After the performance of actions, the PKB is updated. This includes trajectories, used body parts, the effect of action related to the execution goals.

7.2.4 Semantic memory and knowledge base

The *Semantic Memory (SM)* model is in charge of verifying the existence of knowledge inside its *Semantic Knowledge Base (SKB)*. It gives preference to knowledge already present inside the *Knowledge Base (KB)* in case of contradiction.

The SKB includes the knowledge present in KNOWROB, the manual and locomotion cooking verbs compilation (Table 4.2), and the Epic Kitchen dataset before there is a robot execution. When the robot executes a task, new knowledge coming from the WM is added. This integration of new knowledge coming from the WM happens after the task execution is over. One example of how the representation of knowledge looks inside the SKB is presented in Figure 7.5. They are part of the upper ontology that shows temporal and spatial things, actions, and sub-actions related to a kitchen scenario. For example, it includes some `HumanScaleObjects` such as `Meat`, `Vegetable` and `CowMilk-Product`.

Even when the SKB includes a vast amount of knowledge, it still needs more related to the chemical properties of food, for example. Another critical addition would be sound related to kitchen actions to the SKB, so the robots can use it to detect better when the is a failure or action happening. A few examples are *boiling*, *falling* of objects, etc.

During this research, it was noted that the physical properties of food are not vast. These properties include viscosity and friction coefficients, among others. This kind of information is of great value for the robotics community, as it would help robots to increase their capability of better decision-making. This thesis project compiled available information from objects in the Epic Kitchen dataset, as no other resources were found. Also, knowledge about fluid mechanics was added to the SKB to give more information about the interaction between objects in an environment to the robot.

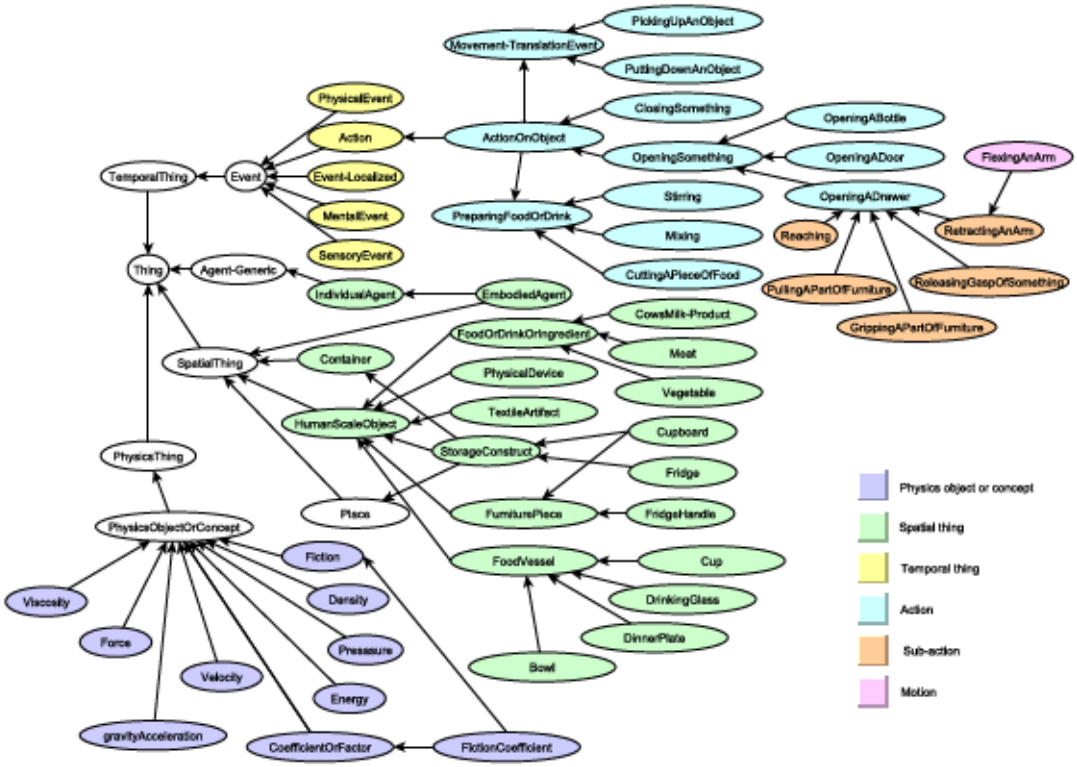


Figure 7.5: Semantic Knowledge Base upper ontology with main classes for temporal and spatial things, actions and sub-actions.

7.3 Summary

In this work, five scenarios were tested on five robotic platforms in simulation. These robots are PR2 and Romeo with two arms, and TIAGo, Fetch and HSR with one arm. They all had to decide how, when and which type of prior knowledge to use.

Various robots use the cognitive architecture. They are then capable of extracting observations' representation of the demonstrated tasks. This property makes the cognitive architecture superior to classical approaches, where the task is learned for a specific scenario or a particular robot.

Also, using the knowledge presented in this work allows robots consider their capabilities and limitations while performing a task. This way, the robots act more reliably and recover from failures faster. Another example of knowledge use happens while using an object's properties, such as shape, weight, material, viscosity and so forth, by the perception system to rank grasp candidates. Furthermore, the robot can perform actions where the plan is unknown beforehand if it is similar to actions in the *Knowledge Base (KB)*. In this case, robots can only perform unknown actions if they have knowledge about them.

The first example concerns picking up a bowl and cup from the kitchen counter and placing them on the table; see Figure 7.8.

Figure 7.6: Robot pick and place bowl from counter and bring them to the table.

This task is used as a baseline for comparing the difference in robots' execution when using the knowledge-processing memory models integrated in the CRAMcognitive architecture and when they are not used. The results showed an increase of success of around 15% and lower execution time when using these models.

The second example concerns bringing a drink from the fridge to the table; see Figure 7.7. First, the robots were capable of `opening` and `closing` the fridge, even when three of them have only one arm. In the case of two-arm robots, such as PR2 and Romeo, they made use of both arms to handle the situation. Conversely, one-arm robots failed more times and took longer to perform the task as they had to return to close the door or use other body part for doing so.

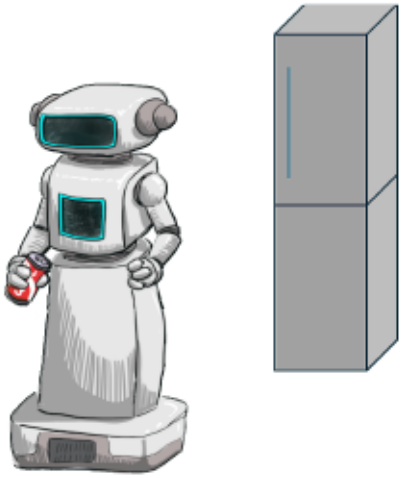


Figure 7.7: Robot bringing drink to the table.

The third example concerns bringing a bowl of soup to the table; see Figure 7.8. First, the robots were capable of `serving` the soup. Then, the robots could keep the bowl upright and maintain a specific maximum acceleration while `carrying` the bowl. In the case of two-arm robots, such as PR2 and Romeo, they made use of both arms to handle the situation. Conversely, one-arm robots failed more times as they depended on the grip force the could apply to the bowl.

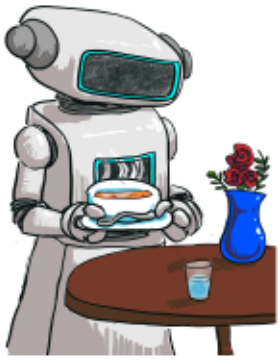


Figure 7.8: Robot bringing bowl with soup to the table.

In a forth example, the PSRs served juice from a box container into a glass; see Figure 7.9. The robots considered the box's orientation and angular speed while `pouring`. As none of them included sensors to measure the change in weight of the box, they used the juice viscosity to calculate the change of liquid related to time.

Figure 7.9: Robot serving juice from a box container into a glass.

In the last example, the PSRs mixed ingredients in a bowl to create a batter, see Figure 7.10. Depending on the stability of the bowl, a one-handed robot uses less speed than two-handed robot, which can keep the bowl stable with the other hand.

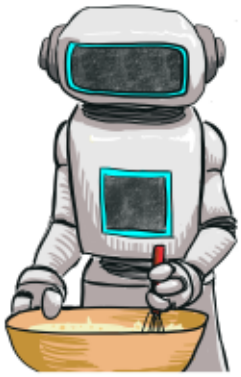


Figure 7.10: Robot mixing ingredients inside a bowl.

The robots used the ROBOSHERLOCK framework to perceive and interpret sensory information more effectively by modeling the different stages of perception, such as feature extraction,

pattern recognition and object recognition. Even though more knowledge can be added to improve the robot's capabilities, all the knowledge added to the different KBs is sufficient for performing manipulation actions in human environments. Furthermore, the memory models and KBs integrated into the cognitive architecture CRAM increased the capabilities of PSRs tested in this thesis work to perform manipulation actions successfully. They could focus on relevant information and filter out irrelevant information by modeling attentional processes and mechanisms. Furthermore, they were capable of storing and retrieving information more efficiently by using effectively the memory models implemented in this thesis work.The robots could learn from experience using the different types of learning implemented in this thesis work, such as reinforcement learning, supervised learning and unsupervised learning. That way, they could reason about the world and make decisions based on uncertain and incomplete information. In general terms, the cognitive architecture CRAM is able to improve successful task executions across the five robotic platforms by 11% compared with an implementation without specialized memory modules.

This work shows that the use of cognitive modules can improve the performance of cognitive architectures. This work shows also that the use of cognitive modules facilitates the transfer of knowledge from humans to robots and, could transfer knwoledge from one robot to another, even if different capabilities are present.

Appendix A

Competitions for Personal Service Robots

In order to provide a test scenario for Personal Service Robots (PSRs) prototypes, e.g., in household-like environments, in 2006, the Robot World Cup Initiative (RoboCup), formed by international scientists [Kitano et al., 1997], known for its robotic football leagues, created the RoboCup@Home category [Wisspeintner et al., 2009]. This league aims to test autonomous mobile robots as an assistive technology in relevant tasks in a home-like scenario [Ferrein et al., 2013]. Its tests include *Human-Robot Interaction (HRI)*, navigation and mapping in dynamic environments, computer vision and object recognition under natural light conditions, object manipulation, adaptive behaviors, behavior integration, use of devices in their environment, standardization and system integration. In 2017, three new sub-leagues were introduced: the *Open Platform League* (OPL), the *Social Standard Platform League* (SSPL) and the *Domestic Standard Platform League* (DSPL). Commercially available robots can participate in two of these sub-leagues. In the case of the SSPL category, only the robot Pepper can participate. On the other hand, in the DSPL category, both robots TIAGo and Human Support Robot (HSR) can participate. All these robots are in Table A.1. Maybe companies saw the possibility of improving their robot's capabilities by testing and developing them in realistic scenarios provided by robotic competitions. The yearly RoboCup@Home competition happens in different countries each time. There are also local RoboCup@Home competitions, such as the European, German and Iran Open. As expected, some of the robots presented in Figure 2.1 have competed in either of these competitions in different years and are marked by a green circle.

After some years of the RoboCup@Home, roboticists saw the importance of having more international robotics competitions. RoboCup concepts were expanded through two-to-three years projects funded by the European Commission, a consortium composed of six partners under its FP7 and Horizon2020 programs. This commission formed the European Robotics League (ERL) to include indoor competitions related to domestic and industrial robots [Lima et al., 2017]. That is how the Robot Competitions Kick Innovation (RoCKIn) in Cognitive Systems and Robotics, which also includes a RoCKIn@Home camp started [Iocchi et al., 2017]. This tournament's test scenarios focus on PSRs for elderly or impaired care regarding health

and personal development. After the first funding ended in 2015, the ERL launched other robotic tournaments with the same tasks as the RoCKIn tournament. The first one was held in 2018 and is known as the European Robotics League-Service Robots (ERL-SR). Another tournament created by this institution is the European Robotics League-Consumer Service Robots (ERL-CSR), which aims to prepare participants for the European Robotics League-Smart City Challenges. During the ERL-CSR, the robots should execute tasks in a smart home and a shopping mall like smart cities. Even more and in the same year, the ERL-CSR concept organized a major tournament during the *International Conference on Intelligent Robots (IROS)* in Madrid with the same challenges. In the same year, the World Robot Summit (WRS) launched the World Robot Challenge with a service robot category [Okada et al., 2018], aiming for the collaboration of humans and robots in everyday environments, either virtual or physical. The standard robotic platform in this competition is the Human Support Robot (HSR) (see Table A.1). Even more, Toyota organizes a competition with the HSR as a standard platform in order to improve its development.

A.1 Personal Service Robots commercially available

This section compares Personal Service Robots (PSRs) prototypes available to purchase. The service robots without an arm as manipulation devices are mainly used for cleaning, reminders or "mobile tablets," as they are sometimes referred to by users in the study performed by Bedaf et al. [2015]. They open people's familiarity with robots as home devices. However, robots that can manipulate objects are still costly and their capabilities are not yet sufficient to attract buyers outside research institutions. For this reason, in this section, PSRs' capabilities to date are presented, when available. Table A.1 provides information regarding robot command and processing types as local, cloud, mixed and remote. Local refers to having all the computing power inside the robotic platform. On the other hand, the cloud refers to an external computation where knowledge bases or extra processing are available to the robot on a server in the network. In the case of mixed, it is a combination of cloud and local only. Finally, remote is related to a human controlling the robot with an external device, even when the robot could include some autonomous behaviors. In the case of middleware, it takes the names presented in section 2.4 as a reference. As memory is an integral part of this work, a column with memory concepts is added; their names are explained in detail in section 6.1.1. The robot's order is by year of release and, even when some robots are produced by the same company, they are separated by the robot's name. It is also important to mention that robots marked with an indicator ● before their name have already been used or tested in real houses rather than only in lab environments.

Table A.1: PSRs developed by industry from 2000 to present order by released year.

Robot's name, developer and year released	Base type	Manipulator(s) specification	Command and processing type	Middleware	Memory concept used
ASIMO by Honda in 2000 [Sakagami et al.]	Bipedal	Arm: 7 DoF x 2 Hand: 4 fingers and 1 thumb	Mixed	N/A	N/A
HRP by Kawada Industries in 2002 [Kaneko et al.]	Bipedal	Arm: 7 DoF x 2 Hand: 4 fingers and 1 thumb	Mixed	N/A	N/A
by Fujitsu Laboratories Ltd. in 2005 [Stahl et al.]	Wheeled	Arm: 4 DoF x 2 Hand: gripper	Cloud	N/A	N/A
T-HR3 by Toyota in 2005 [Stahl et al.]	Bipedal	Arm: 7 DoF x 2 Hand: 4 fingers and 1 thumb	Cloud	N/A	N/A
Smartpal by YASKAWA Electric in 2007 [Qixin et al.]	Wheeled	Arm: 7 DoF x 2 Hand: 3 fingers or gripper	Cloud	CORBA	N/A
PR2 by Willow Garage in 2009	Wheeled	Arm: 7 DoF x 2 Hand: gripper	Mixed	ROS	Working Episodic
REEM by Pal Robotics in 2010 [Benavidez et al.]	Bipedal or wheeled	Arm: 7 DoF x 2 Hand: 4 fingers and 1 thumb	Local	ROS	N/A
●HSR by Toyota in 2012 [Hashimoto et al.]	Wheeled	Arm: 7 DoF Hand: gripper	Mixed	ROS	Episodic
Fetch by Fetch Robotics in 2014 [Wise et al.]	Wheeled	Arm: 7 DoF Hand: gripper	Mixed	ROS	N/A
●Pepper by SoftBank Robotics in 2014 [Pandey]	Wheeled	Arm: 5 DoF x 2 Hand: 4 fingers and 1 thumb	Local	ROS	N/A
Romeo by SoftBank Robotics 2014 [Benavidez et al.]	Bipedal	Arm: 7 DoF x 2 Hand: 4 fingers and 1 thumb	Local	N/A	N/A
Sanbot by Qihan Technology Co. Ltd. in 2016 [Ltd.]	Wheeled	Arm: 5 DoF Hand: gripper	Cloud	N/A	N/A
SpotMini by Boston Dynamics in 2016 [Phillips et al.]	Quadruped	Arm: 5 DoF Hand: gripper	Remote	N/A	N/A

Table A.1 – *Continued from previous page*

Robot's name, developer and year released	Base type	Manipulator(s) specification	Control system	Middleware	Memory concept used
TIAGo (Configurable) by Pal Robotics in 2016 [Pages et al.]	Wheeled	Arm: 7 DoF Hand: gripper	Mixed	ROS	N/A
AEOLUS by AEOLUS Robotics in 2019	Wheeled	Arm: 7 DoF x 2 Hand: gripper	Local	N/A	N/A
EveR3 by HALODI Robotics in 2019	Wheeled	Arm: 5 DoF x 2 Hand: 4 fingers and 1 thumb	Remote	N/A	N/A
XR-1 by INNFOS in 2019	Wheeled	Arm: 5 DoF x 2 Hand: 4 fingers and 1 thumb	Cloud	N/A	N/A

A.1.1 Functionalities

Personal Service Robots (PSRs) depend on their hardware to perform actions without a doubt. However, to increase their capabilities they could use a model from neuroscience, which is tested in Chapter 6. A second variable is price, the price increase as they have more complex hardware; see Table A.1 column *Manipulator(s) specification*. Some of those robots include a complex hand that gives them more flexibility to a certain degree but makes them less accessible price-wise to the general public. Moreover, some of these robots lack autonomous behaviors and are remote-controlled or used as avatars, such as SpotMini, T-HR3, HRP. Some others, such as PR2, REEM, HSR and TIAGo, are acquired mainly by research institutions and not by other customers. In the specific case of Pepper, it has been deployed in thousands of homes and public places [Pandey, 2018].

In the case of PR2, it is an open-source robotic platform used by some research institutions to test their algorithms. One example is PR2 preparing pancakes [Walther-Franks et al., 2015], pizza and popcorn by using a semantic knowledge base called KNOWROB [Tenorth and Beetz, 2013] and CRAM [Beetz et al., 2010b] for planning, both are also used in this work. PR2 has also been used to learn users' preferences [Abdo et al., 2016]. In this case, the robot learns missing information about where to store objects in a house environment using a collaborative filtering paradigm and personalized recommendations. PR2 has been also used for implementing reinforcement learning and then obtaining new skills from more basic ones [Alami et al., 2006] by associating perception and execution to produce representations from its movement. It includes a feature similar to a distributed episodic (events) and working (planning and voluntary movement) memory set to represent its capabilities, skills and resulting interpretation. This last application benefits this work and will be presented in more detail in Chapter 6. In general, this

robotic platform is versatile for many applications, but it is costly and needs to be ready for a real house environment. Still, this robot is used in this work to test if the knowledge provided is enough to perform complex manipulation actions introduced in Chapter 5.

Advanced Step in Innovative MObility (ASIMO) is capable of similar movement as the human body [Sakagami et al., 2002]. It has been used as a receptionist and presented in trade shows and videos. This service robot can pour drinks, an essential ability for a PSR. However, how it performs such actions is not openly available.

Some companies have agreements with universities to continue the development of their products. Two examples of this are the Humanoid Robotics Platform (HRP) with the National Institute of Advanced Industrial Science and Technology (AIST) and the HSR with the University of Tokyo (JSK). As mentioned in Section A, some robots are also used as standard platforms by various universities in competitions.

However, only a few robots such as TIAGo and HSR have been tested in real houses so far. Even when Pepper has been deployed in public places, its arms are not meant to manipulate objects, but instead to produce gestures and light manipulation tasks [Johnson et al., 2017].

HSR was specially built, as its name says, to support people in need in their daily life [Yamamoto et al., 2019]. The research community, including various institutions participating in the RoboCup@Home, aims to develop the robot's capabilities, such as *operating* on furniture, e.g., `opening/closing` drawers, `using` microwaves, `fetching` and `carrying` objects and `tidying` up. A more specific example is while HSR `grasps` a bottle. For that to happen, it requires a sequence of commands, including to `move` the whole body to a neutral position, `move` the hand to the front of the bottle and specifying force to grasp.

This shows how hard it is to build and program a robot to fulfill human expectations and why they are not yet in houses as science fiction stories predicted. One idea presented in this work is the use of memory models. The use of concepts such as working [Alami et al., 2006] or episodic [Winkler et al., 2017] memories, presented in more detail in Chapter 6, have been used in some architectures and tested in PR2 and HSR. However, to our knowledge, it has yet to be tested on many other platforms from industry. In the case of this work, we use TIAGo, Pepper, Fetch, PR2 and HSR to test our proposed framework in simulation. This selection follows the idea that as many robots use a parallel jaw gripper [Billard and Kragic, 2019], the proposed framework can be better tested with these robotic platforms.

A.2 Robotic prototypes from research institutions

The previous section A.1 presented Personal Service Robots (PSRs) developed by industry. This section introduces PSR prototypes developed by research institutions in Table A.2. As in the previous section, this table provides robot command and processing types as local, cloud, mixed and remote. In the case of middleware, names are taken from section 2.4 and memory concept types from section 6.1.1. They are also ordered by release year, although robots with a different

name but the same robotic platform are grouped in the same cell. To find the members of this table, an exhaustive search was made in the literature for robots specifically used for solving household chores. Also, some were taken from the RoboCup@Home scores list [Matamoros and Luz, 2018], where they solved the competition's challenges (see section A), earning the 1st, 2nd or 3rd place only. Again, the robots marked with an indicator ● before their names are the ones already tested in real environments.

Table A.2: PSRs developed by academia from 2000 to present order by release year.

Robot's name, developer and year released	**Base type**	**Manipulator(s) specification**	**Command and processing type**	**Middleware**	**Memory concept used**
Care-O-bot (Configurable) by Fraunhofer IPA in 2002 [Martinez-Martin and del Pobil]	Wheeled	Arm: 7 DoF Hand: gripper	Local	ROS/Orocos	N/A
HomeMate by Sungkyunkwan University International in 2003 [Zhao et al.]	Wheeled	Arm: 5 DoF Hand: gripper	Local	N/A	N/A
MARY by Tohoku University in 2004 [Taipalus and Kosuge]	Wheeled	Arm: 5 DoF Hand: gripper	Remote	N/A	N/A
Mahru by Korea Institute of Science and Technology in 2005 [Cha et al.]	Bipedal	Arm: 7 DoF x 2 Hand: 3 fingers and 1 thumb	Cloud	N/A	N/A
CAESAR by RWTH Aachen University in 2006 [Ferrein et al.]	Wheeled	Arm: 5 DoF Hand: gripper	Local	N/A	N/A
●HERB by University of Washington in 2006 [Srinivasa et al.]	Wheeled	Arm: 7 DoF x 2 Hand: 3 fingers	Local	N/A	N/A
IRobo by The International Islamic University Chittagong in 2007 [Siddiky et al.]	Wheeled	Arm: 4 DoF x 2 Hand: gripper	Local	N/A	N/A
Security warrior by The National Chung Cheng University in 2007 [Luo et al.]	Wheeled	Arm: 4 DoF x 2 Hand: gripper	Local	N/A	N/A
Lisa by University of Koblenz-Landau in 2008 [Seib et al.]	Wheeled	Arm: 6 DoF Hand: gripper	Local	ROS	N/A
Rollin' Justin by The German Aerospace Center 2008 [Bäuml et al.]	Wheeled	Arm: 7 DoF x 2 Hand: 3 fingers and 1 thumb	Local	ROS	Episodic

Table A.2 – *Continued from previous page*

Robot's name, developer and year released	Base type	Manipulator(s) specification	Control system	Middleware	Memory concept used
SAM by University of Rennes in 2008 [Remazeilles et al.]	Wheeled	Arm: 8 DoF Hand: gripper	Mixed	N/A	N/A
B21 Bender by Technical University of Munich in 2009 [Stulp et al.]	Wheeled	Arm: 6 DoF Hand: gripper	Local	N/A	Episodic
Biron by Bielefeld University in 2009 [Meyer zu Borgsen et al.]	Wheeled	Arm: 6 DoF Hand: gripper	Local	ROS	N/A
Cosero and Dynamaid by University of Bonn in 2009 [Stückler et al.]	Wheeled	Arm: 7 DoF x 2 Hand: gripper	Local	ROS	N/A
Justina by The National Autonomous University of Mexico in 2009 [Savage et al.]	Wheeled	Arm: 7 DoF x 2 Hand: gripper	Local	ROS	N/A
KeJia by University of Science and Technology of China in 2009 [Chen et al.]	Wheeled	Arm: 7 DoF x 2 Hand: gripper	Local	ROS	N/A
Rosie by Technical University of Munich in 2009 [Beetz et al.]	Wheeled	Arm: 7 DoF x 2 Hand: 3 fingers and 1 thumb	Mixed	ROS	Episodic
TWENDY-ONE by Waseda University in 2009 [Iwata and Sugano]	Wheeled	Arm: 7 DoF x 2 Hand: 4 fingers	Local	N/A	N/A
Golem by The National Autonomous University of Mexico in 2010 [Pineda et al.]	Wheeled	Arm: 4 DoF x 2 Hand: gripper	Local	ROS	N/A
AMIGO by Eindhoven University of Technology in 2011 [Elfring et al.]	Wheeled	Arm: 7 DoF x 2 Hand: gripper	Local	ROS	N/A
●HOBBIT by Technical University of Vienna in 2011 [Martinez-Martin and del Pobil]	Wheeled	Arm: 4 DoF Hand: gripper	Local	N/A	N/A
HoLLiE by Forschungszentrum Informatik in 2012 [Vasquez Tieck et al.]	Wheeled	Arm: 6 DoF x 2 Hand: 4 fingers and 1 thumb	Local	N/A	N/A

Table A.2 – *Continued from previous page*

Robot's name, developer and year released	Base type	Manipulator(s) specification	Control system	Middleware	Memory concept used
Floka by Bielefeld University in 2016 [Meyer zu Borgsen et al.]	Wheeled	Arm: 7 DoF x 2 Hand: 4 fingers and 1 thumb	Local	ROS	N/A
Armar by Karlsruhe Institute of Technology in 2017 [Zhu et al.]	Wheeled	Arm: 4 DoF x 2 Hand: 4 fingers and 1 thumb	Local	N/A	N/A
Walking-assistant by The National Chiao Tung University in 2017 [Song et al.]	Wheeled	Arm: 6 DoF x 2 Hand: gripper	Remote	N/A	N/A

A.2.1 Functionalities

Personal Service Robots (PSRs) can perform specific manipulation tasks with a certain degree of autonomy, e.g., fetch and place of geometric objects. All these improvements happen even when the difficulty to build such systems is high, as mentioned before. However, these robots are not yet capable of performing these tasks without being previously pre-programmed or in a generalizable way. Most robots are able to grasp specific types of objects, using specified grasp types [Feix et al., 2016] and put them in certain locations. More complex manipulation is only seen in systems like Cosero, Rollin' Justin and OK-KeJia, all included in table A.2. It is understandable that PSRs require adequate body capabilities, meaning actuators and sensors. However, they also require control systems able to acquire knowledge about how to execute actions such as manipulating electrical devices, cleaning or cooking when commanded. If those requirements are not fulfilled, the gap between human expectations from robots and what they can actually do is not closed, humans won't see the utility of having such expensive devices at home.

PSRs that use frameworks also used in this work are AMIGO, Care-O-Bot and Rosie in the RoboEarth project [Di Marco et al., 2013]. These robots make use of a semantic knowledge base KNOWROB [Tenorth and Beetz, 2013] and CRAM [Beetz et al., 2010b] to create robot execution plans, the same way as PR2 from section A.1. In this case, semantic information about the robot's capabilities and body were included inside KNOWROB. Specific designators represent how to perform actions were created as they are robot dependent. In the case of AMIGO, the task tested was to `get` a drink from another room. The first plan includes actions such as open the door and `navigate` in order to pass a door. Then the robot has to `find` and `transport` a drink back. For the `find` plan, the task is decomposed into the sequence `navigate`, `pick-up`, `navigate`, `operate` the door button, `navigate` and `drop-off` the drink. Care-O-Bot

differs from Amigo in that it has only one arm and a tray. In order to be able to solve the same task, the robot has to place and hold the drink on its tray while manipulating the door button.

Representation is not enough, a PSR also needs to remember certain things. For this reason, the use of different types of memory is of interest for this work, similar to the one presented by Alami et al. [2006] using the PR2 robotic platform, more detail is presented in Chapter 6. That memory can be in the form of the robot's own experience to learn how to better perform an action. One example is B21 Bender, which was used to test the concept of Action-Related Place (ARPlace) that uses probabilistic representation for manipulation [Stulp et al., 2012]. The representation of the task, in this case, includes a prediction of the best behavior to apply through experience-based learning. B21 Bender is tested while cleaning a table 50 times in approximately 6 hours. During this time it locates, grasps, and lifts a cup from the table and moves it to the kitchen oven. In this case, only two grasp types are sufficient to grasp 14 everyday kitchen objects.

COgnitive SErvice RObot (Cosero) [Stückler et al., 2012] is a good example of a low-cost robotic platform able to grasp rigid and symmetric objects typically encountered in house scenarios with two grasp types, side and top. This is possible by its high-level behavior generated by semantic parsing of natural language and a hierarchical representation using a state-machine. For example, to perform a task such as open a door, the robot drives in front, detects and approaches to the door handle to finally grasp it. Then, the drive moves backward while the gripper moves to a position to the robot's side where the angle used to open the door is sufficiently large to approach the open device. The robot moves its body backward until the gripper reaches its target position. This sequence is preprogrammed and includes some failure detection mechanisms to avoid them.

On the other hand, the more complex robotic platform Rollin' Justin [Bäuml et al., 2011] is able to prepare coffee in a pad machine in an office setup. For doing so, the robot grasps coffee pads and inserts them into a coffee machine, which involved opening and closing the pad drawer. This robot is capable of precise fine manipulation with the fingertips, which requires precision in localizing objects. Therefore, Rollin' Justin combines vision and force sensing, its use of perception is accurate by using finger sensors. During the manipulation of the coffee pad, the robot moves while monitoring the contact to the supporting surface before the fingers are closed. To insert the coffee pad into the coffee machine, the robot requires to open and close the pad drawer.

In the case of KeJia [Chen et al., 2017], it participated in the RoboCup@Home since 2009 and won the world championship in 2014. During the competition, the robot presented the capability of manipulating an oven by opening and closing the door and pushing buttons. This demonstration was while making popcorn. KeJia's manipulation module regarding grasping, placing, and touching operations is presented by Shuai and Chen [2019]. KeJia's behaviors are divided into moving the end effector to a specific position and closing or opening its gripper. For actual body transferring, KeJia's navigation is divided into mapping (detects obstacles), locating, and navigation (control robot's movement to a goal location).

The only two robots from this section already tested in real homes with people are HERB and HOBBIT. Home Exploring Robotic Butler (HERB) object manipulation ability depends on a Generalized Approach to Tracking Movable Objects (GATMO) [Srinivasa et al., 2009]. This approach includes a multi-level classification hypothesis hierarchy to keep track of where people and other movable objects are in the environment as *Metaobjects*. It depends on maintaining semantic labels, even regarding changes within the environment. For object manipulation, HERB is capable of `grasping` and `manipulating` free-form objects in high clutter. HERB has been tested while first `navigating` around a kitchen, then `searching` for mugs and finally `bringing` them back to the kitchen sink. HERB is also able to `close` a refrigerator by using task constraints and simple contact analysis, e.g. it first hooks the fridge handle and then pushes it. HERB uses a different sequence than Cosero that is generated by its framework, which gives it more flexibility. In the case of HOBBIT [Martinez-Martin and del Pobil, 2018], it is able of `searching` and `bringing` objects, `transporting` small items, and `giving` reminders, as well as emergency detection and handle by notifying the required emergency department. For that to be possible, the concept of Mutual Care was introduced such that the robot learns user's habits and preferences. The use of HOBBIT resulted in high independence of the elderly in care facilities. In addition, a three-week field trial in real private homes was performed in three European countries. These trials revealed that users highly appreciated the functions of picking up objects from the floor, transporting objects, emergency recognition, fitness program and giving reminders. Concerning its usability, despite the way the robot could perform tasks, robot's errors while performing tasks led users to frustration. For this reason, emotional attachment weakened over the duration of the trials as the user's expectations could not be fulfilled. Nevertheless, users believed that a market-ready robot version would be essential for supporting people who are more fragile and more socially isolated.

PSRs with a gripper, such as AMIGO, Care-O-Bot, B21 Bender, Cosero, KeJia, HOBBIT, for more refer to Table A.2, are able to perform complex manipulation because of their frameworks. Still, they have more limitations in comparison with robots that have fingers as Rollin' Justin, HERB, HOBBIT and Rosie that are able to more precise manipulation and operation of devices. Seeing how they perform their plans allows comprehending their action hierarchy for planning, which seems mixed. One part of this work is to look into the naming of atomic actions above movements to then be added to the ontology. Then to use them for action plan generation. By looking at the naming and sequence of actions, the representation of action-subaction pairs can be improved. Some action names are used constantly, but in other cases, more specific ones are used instead. For this reason, is needed to improve the ontology, which can be used by different PSRs, see Chapter 4.

Appendix B

Action and sub-action plans for solving specific tasks

The path with the *next* label will follow the plan in an ideal situation. However, failures can appear and in that case, either an action or a sub-action inside that action repeat. Also, actions can repeat multiple times depending on the needs of the task execution. The same happens with the sub-actions; they can repeat, mainly in case of failure.

B.1 Pick and place a bowl and a cup task

The pick and place task requires of the actions showed in Figure B.1.

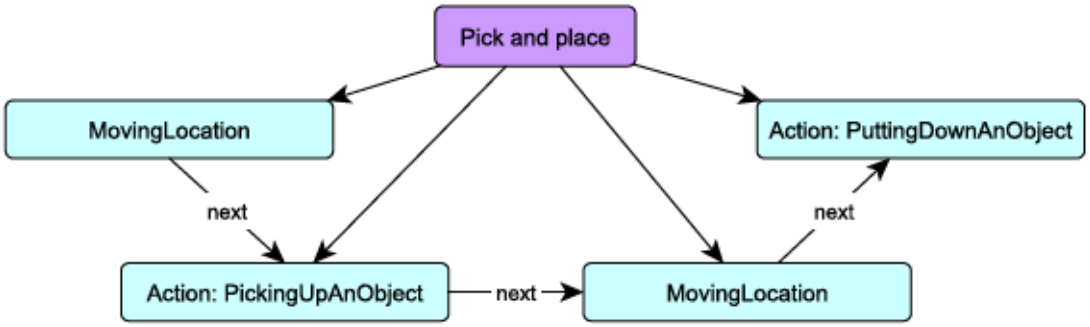
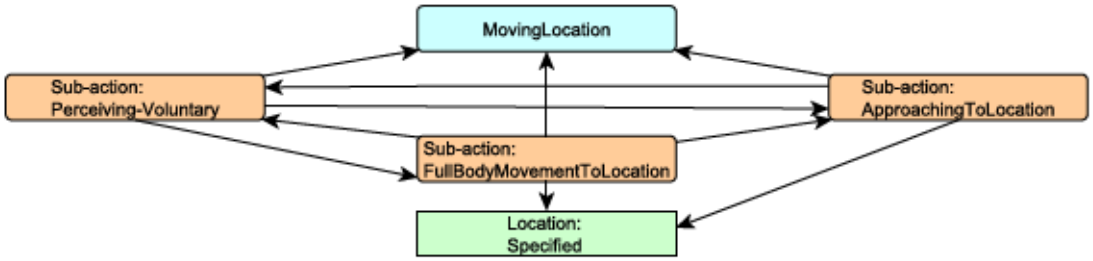


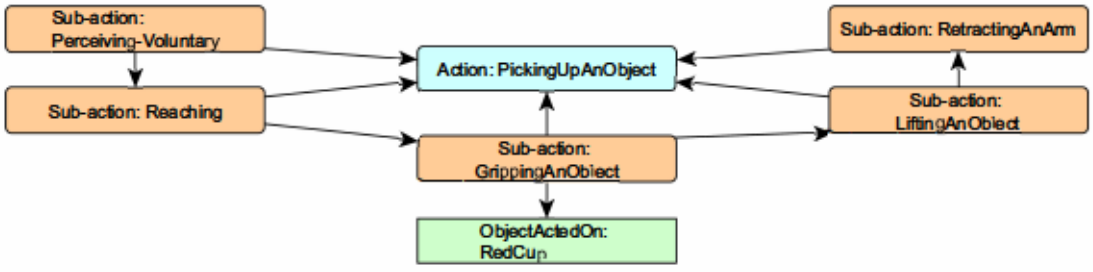
Figure B.1: Task tree of robots picking and placing a cup from the counter to the table.

As mentioned before, actions can repeat. In this example, they repeat for the cup and the bowl. The sub-actions involved in solving this task are shown in Figure B.2. Each sub-action for each action presented in Figure B.1 can be seen. In the case of the action `MovingLocation`, first, take the value of `KitchenIslandCounter` and later the `Table` instead of `Specified`.

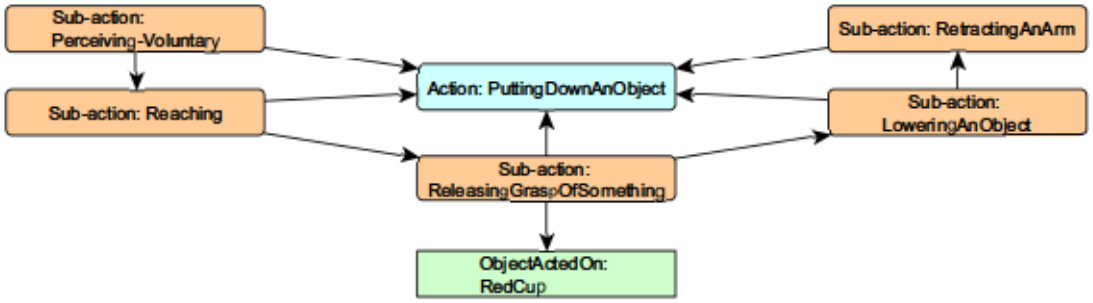
The plan for *picking up* an object looks like the Lisp Code 10. It is important to note that the sub-action `perceive` receives the object or object's type of interest. Then, the system passes the location to which the robot will `reach` for the object to `grab` it.



(a) Moving to a location action.



(b) Picking up an object action.



(c) Putting down an object action.

Figure B.2: Actions and sub-actions involved in the task of picking and placing a cup and bowl from the counter to the table.

Lisp Code 10 Pick-Up plan with sub-actions in a desired execution.

```
(def-plan pick-up (?object-to-pick)
  (with-robot-at-location (?location-at-which-to-pick)
    (perform
      (an action
        (type picking-up)
        (object (an object (type ?object-to-pick)))
        (arm ?arm-to-be-used)
        ;; Sub-actions
        (perceive ?object-to-pick)
        (reach ?arm-to-be-used ?location-at-which-to-grasp)
        (grasp ?arm-used ?grasp-pose)
        (lift ?lift-pose-to-be-used)
        (retract ?arm-used)
      )
    )
  )
)
```

B.2 Bring a drink from the fridge to the table task

The bring a drink to a specific location task requires of the actions showed in Figure B.3.

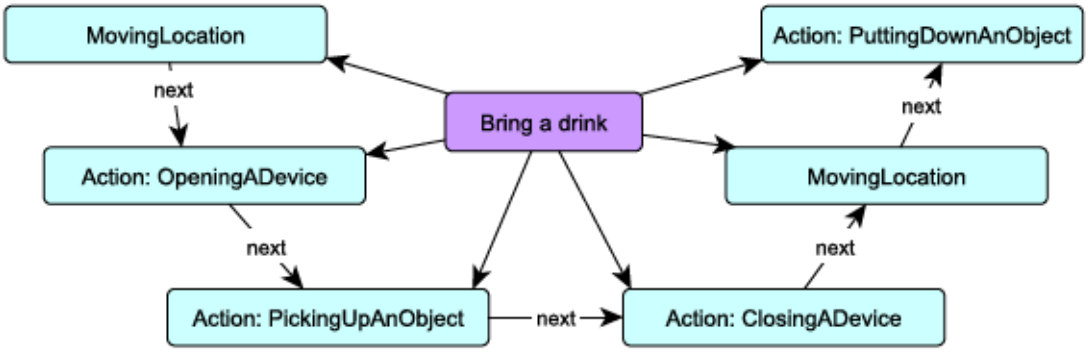
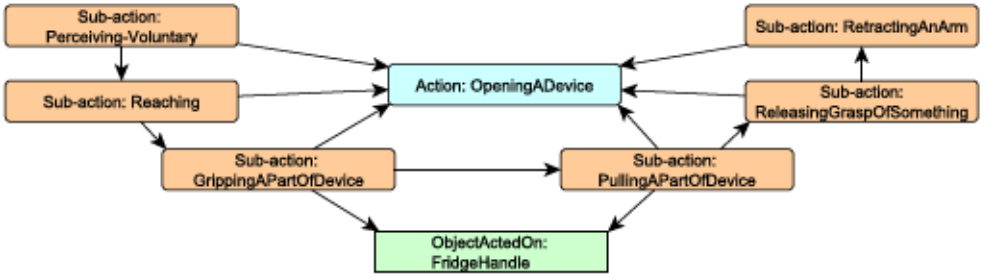
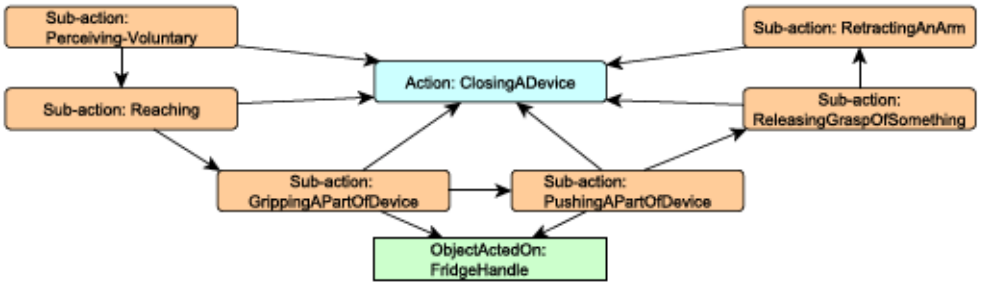


Figure B.3: Task tree of robots bringing a drink from the fringe to the table.

Some of the actions are repeated from the previous pick and place task, for that reason, only `Opening a device` and `Closing a device` action trees are shown in Figure B.4.



(a) Opening a device (fridge) action.



(b) Closing a device (fridge) action.

Figure B.4: Specific actions and sub-actions to solve the bringing a drink from the fringe to the table task.

B.3 Serving soup task

The serving soup task requires of the actions showed in Figure B.5.

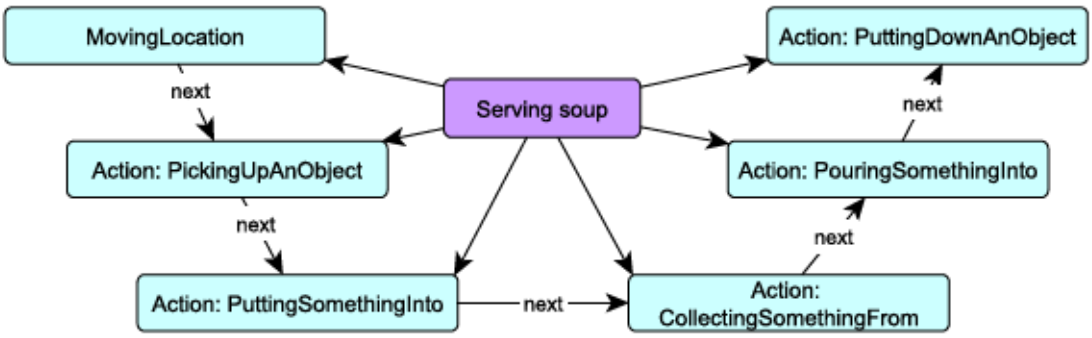
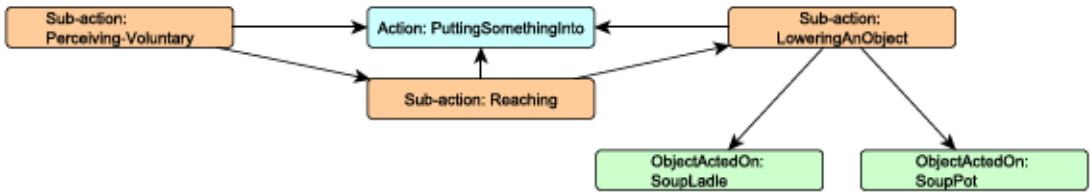
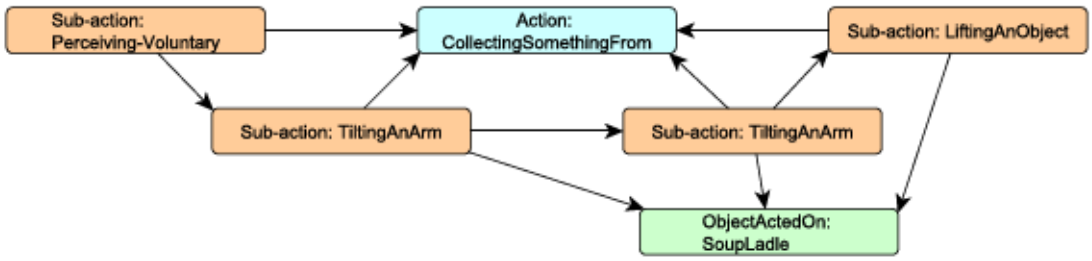


Figure B.5: Task tree of robots serving soup.

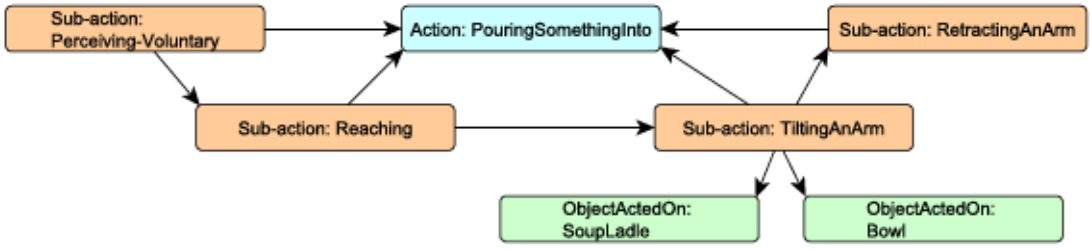
Similarly to the previous task, some of the actions are already repeated from the previous pick and place task. Now, only `Putting soup ladle into pot`, `Collecting soup from the pot` and `Pouring soup to the bowl` action trees are shown in Figure B.6.



(a) Putting soup ladle into pot action.



(b) Collecting soup from the pot action.



(c) Pouring soup to the bowl action.

Figure B.6: Specific actions and sub-actions required to serving soup.

B.4 Serving juice task

The serving juice task requires of the actions showed in Figure B.7.

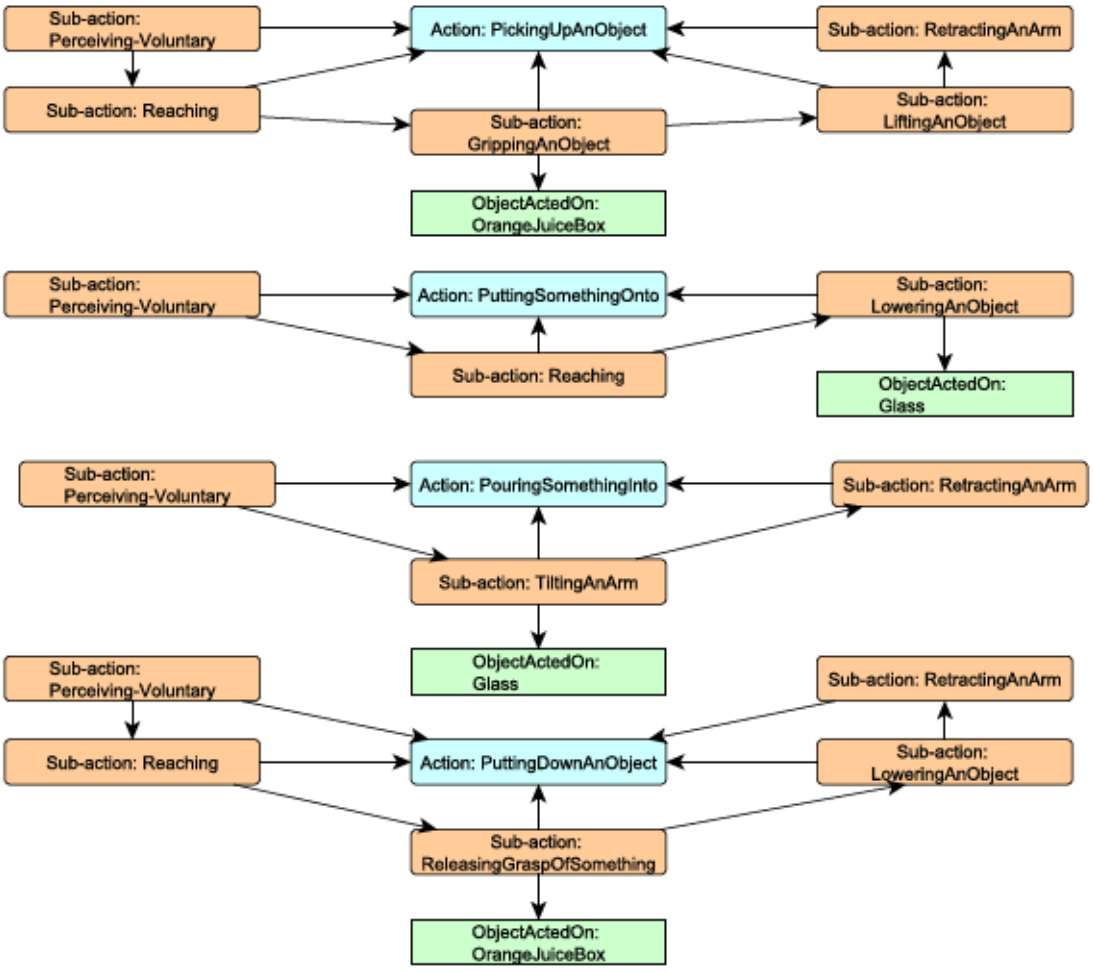


Figure B.7: Task tree of robots serving juice.

As this task is very similar to the serving soup task, there are no new actions represented in more detail.

B.5 Mixing ingredients for batter task

The mixing ingredients for a batter task requires of the actions showed in Figure B.8.

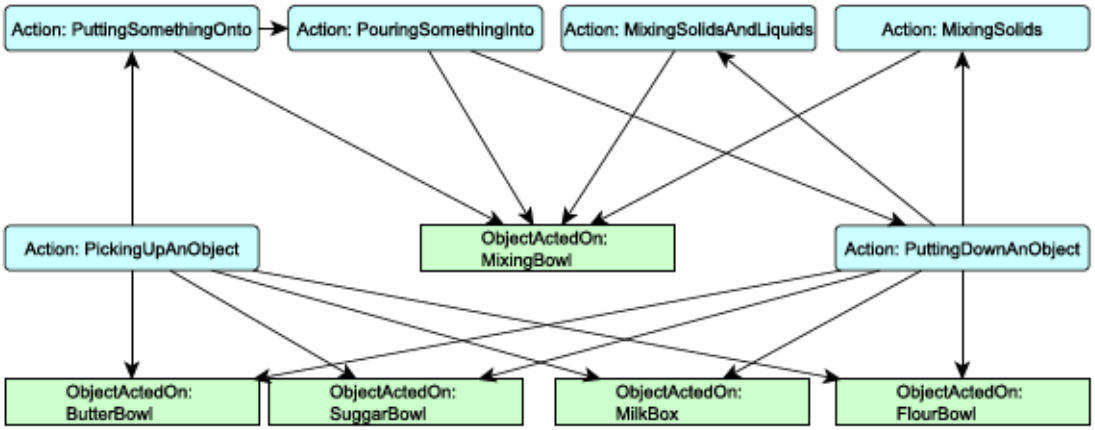


Figure B.8: Task tree of robots mixing ingredients for batter.

Similarly to the previous tasks, some of the actions are already repeated. In this case, only `Mixing solids`, `Mixing solids and fluids` action trees are shown in Figure B.9.

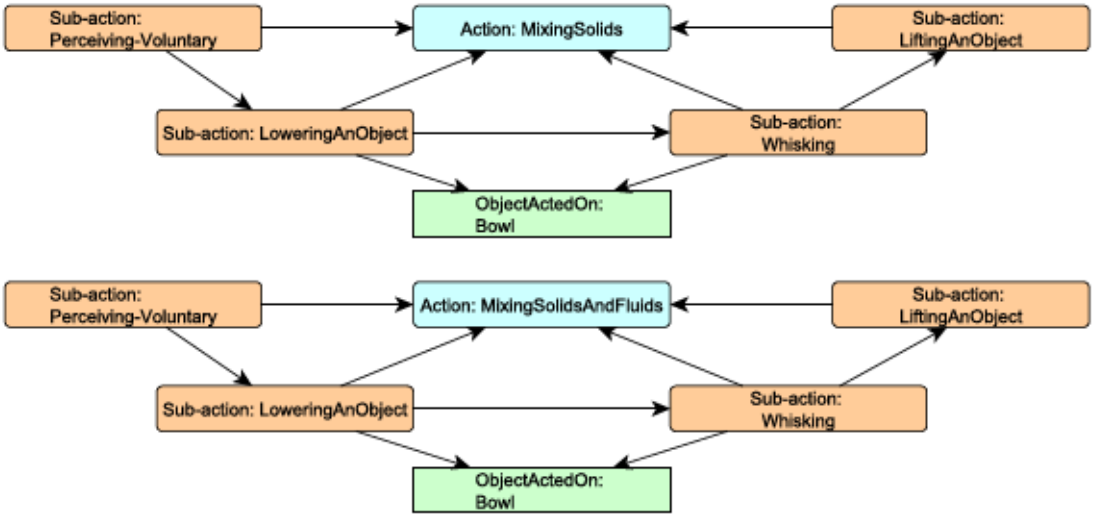


Figure B.9: Task tree of robots mixing sub-actions.

Appendix C

List of Publications

C.1 Own published papers

This thesis work is based on prior work published. All the parts of this work drawing on content from prior publications referenced the prior works where appropriate. For the sake of completeness, this section presents a complete list of prior publications related to this thesis work.

Conference Papers

L. Salinas Pinacho, A. Wich, F. Yazdani, and M. Beetz. Acquiring knowledge of object arrangements from human examples for household robots. In F. Trollmann and A.-Y. Turhan, editors, *KI 2018: Advances in Artificial Intelligence*, pages 131–138, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00111-7

Other Publications

L. Salinas Pinacho and M. Beetz. Object grasping and arrangements extracted from virtual reality for a household robot. Poster on the "workshop on concepts in action: Representation, learning, and application", 2018. URL `https://conceptresearch.github.io/CARLA/files/carla_2018/Salinas.pdf`

L. Salinas Pinacho. From knowledge of humans performing everyday activities to the service robots. Extended abstract on the "Recent Trends in Knowledge Compilation (Dagstuhl Seminar 17381)" 9, Dagstuhl, Germany, 2018. URL `http://drops.dagstuhl.de/opus/volltexte/2018/8589`

C.2 Awards

During my PhD research studies, I was granted with the following awards:

- DAAD Fellowship: Scholarship for Ph. D. research award by German Academic Exchange Service (DAAD). From October 2015 until September 2019. Scholarship code: .
- DAAD Student Travel Support: I was awarded by the German Academic Exchange Service (DAAD), with a travel support to participate at Dagstuhl.
- Heidelberg Laureate Forum 2015: I was accepted to participate as Young researcher at the 5th Heidelberg Laureate Forum where only 200 young researchers are accepted.
- 10 out of 200: I was selected between the 10 out of 200 young researchers from around the world at the Heidelberg Laureate Forum.

Glossary

avatar in Hinduism refers to the material appearance or incarnation of a deity on earth. 182

cognitive architecture is commonly used in the AI and cognitive sciences communities to designate the organization of systems designed to model the human mind [Alami et al., 2006]. 7, 9, 15, 37–39, 41, 42, 44, 49, 55, 59, 60, 64, 66, 81, 124, 125, 127, 128, 132, 151, 152, 158, 165, 167, 168, 170, 175, 177, 178

cognitive robot is an autonomous robot that is capable of inference, perception, and learning mimicking the cognitive mechanisms of the brain [?]. 15, 36, 37, 121

designator is a symbolic descriptions that specify more detailed information about actions, describing e.g. objects and locations created by CPL [Beetz et al., 2010b] 42, 50, 52, 133, 153, 187

humanoid robot is a robot which has a similar look as a human. They do not have to be totally equal, but to include a similar look. Also, is a high dimensional movement systems for which classical system identification and control techniques are often insufficient due to unknown sources of non-linearities inherent in these systems [Vijayakumar et al., 2002]. Humanoid robots consists of biologically inspired features, human-like appearance, and intelligent behavior that naturally elicit social responses [Okita et al., 2009]. A human like autonomous robot which is capable to adapt itself with the changing of its environment and continue to reach its goal [Benavidez et al., 2015] 17, 128

independent living support is aimed at executing tasks that service dogs perform, such as picking up or fetching household objects [Hashimoto et al., 2013]. 1

middleware is a software technology that enables the combination of various functional robotic components via a communication network 18, 19, 50, 59, 180, 184

ontology is a knowledge representation that consists of a taxonomy of concepts and relations between these concepts. 42, 64, 73–76, 78, 79, 83, 89, 92

Personal Service Robot is the service robot used for a non-commercial task, usually by lay persons. Examples are domestic servant robot, automated wheelchair, and personal mobility assist robot [Elfving et al., 2012]. 1–3, 5, 7, 10, 13, 15, 16, 37, 47, 49, 57, 65, 76, 78, 85, 91, 120, 121, 161, 166, 168, 179, 180, 182, 184, 186

query A query is an operation that returns information about a theory without changing it.Darwiche and Marquis [2002] 76, 77, 88

RoboCup@Home is a category of the RoboCup competition funded in 2006, which main goal is testing safety and autonomy of service robots. The robot participants have a specific amount of time to perform. Rules are known in advance but the actual scenario is not. 18, 179, 183, 184, 188

robot is an actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks. Autonomy in this context means the ability to perform intended tasks based on current state and sensing, without human intervention [Elfving et al., 2012]. 1–3, 5–10, 12, 15–17, 19, 35–37, 41, 47, 49, 55–61, 65, 66, 68, 71, 73, 77, 91, 92, 99, 126, 128, 130, 145, 148, 150, 151, 153, 156–163, 169–171, 173, 175, 176, 183, 193

RoCKIn is an EU project running for over the next three years. It consists of robot competitions, symposiums, educational camps and technology transfer workshops. 179

service robot is a robot that performs useful tasks for humans or equipment excluding industrial automation application. Note that the classification of a robot into industrial robot or service robot is done according to its intended application [Elfving et al., 2012].A service robot is an entity that is able to perform a number of basic behaviors and compose them in the execution of complex tasks [Pineda et al., 2015]. 16, 19, 42, 75, 180, 183

Acronyms

CRAM Cognitive Robot Abstract Machine first 7, 10, 42, 49, 50, 52, 57, 60, 64, 132, 133, 148, 152, 168, 177, 182, 187

AI Artificial Intelligence vi, 68, 69, 73

AM-EvA automated probabilistic model of everyday activities 75

ANN 27–30, 32, 73, 98, 112, 124

ARS activity recognition systems 95

ASIMO Advanced Step in Innovative MObility 183

Cosero COgnitive SErvice RObot 187, 188

DoF Degrees of Freedom 18

EKB Episodic Knowledge Base vii, 10, 11, 55, 61, 78, 79, 87–89, 127, 130, 136, 137, 150, 151, 153, 156, 157, 166, 171

EM Episodic Memory 5, 9, 11, 39, 42, 52, 60, 61, 64, 88, 101, 121, 123, 125–130, 132, 136, 139, 140, 143, 150, 151, 153, 154, 156, 157, 166, 167, 170–172

HAR human activity recognition 10, 11, 13, 49, 60–62, 75, 95–98, 119, 169

HERB Home Exploring Robotic Butler 188

HRI Human-Robot Interaction 39, 44, 179

HRP Humanoid Robotics Platform 183

HSR Human Support Robot 149–151, 157–161, 163, 165, 166, 175, 179, 180, 183

IROS International Conference on Intelligent Robots 180

ISO International Organization for Standardization 17

KB knowledge base 9, 10, 35, 37, 44, 46, 51, 52, 55, 56, 61, 62, 66, 72, 74, 75, 78, 79, 81, 83–85, 87, 89, 119, 130, 136, 140, 143, 145, 148–151, 156, 166, 169, 171, 173, 175, 177

KR Knowledge Representation 66, 68–70, 74, 77

LfD Learning from Demonstration 7, 10, 11, 92, 98, 99, 166

LSTM Long Short Term Memory 34, 119, 123

LTM Long Term Memory 9, 12, 34, 37, 39, 60, 66, 68, 112, 121, 123, 124, 126, 128, 129, 166, 171

Mocap motion caption system 94, 95

ORO OpenRobots Ontology 76

OS Operating System 19

OWL Ontology Web Language 55, 67, 75, 76, 82, 84, 129, 135

PKB Procedural Knowledge Base 10, 11, 53, 55, 56, 62, 78, 85–87, 89, 130, 133, 134, 140, 141, 150, 151, 153, 156, 157, 166, 172, 173

PM procedural memory 5, 9, 11, 39, 42, 52, 60, 62, 63, 85, 121, 123–125, 127–130, 132, 133, 135, 140, 141, 143, 145, 150, 151, 153, 154, 156, 166, 167, 170, 172

PR2 Personal Robot 2 149–151, 156–161, 163, 166, 175, 176, 182, 183, 187

PSR Personal Service Robot 1–3, 5, 7–13, 15–19, 37, 42, 47, 49, 50, 57, 65, 66, 76, 78, 85, 88, 91, 92, 120, 121, 130, 133, 157, 161, 163, 166, 168, 170, 176, 177, 179, 180, 182–184, 186–189

RDF Resource Description Framework 75, 76

RL Reinforcement Learning 44, 53, 124, 133, 171

RNN Recurrent Neural Network 32–34, 98, 103, 107, 112, 117, 119, 124, 169

RoCKIn Robot Competitions Kick Innovation 179

ROS Robot Operating System 18, 19, 50, 59, 148

SKB Semantic Knowledge Base 10, 11, 55, 62–64, 78, 79, 81, 83–85, 88, 89, 117, 135, 141, 142, 145, 150, 151, 153, 156, 161, 162, 166, 173, 175

SM Semantic memory 5, 9, 11, 39, 42, 60, 61, 64, 93, 121, 123, 125, 127–130, 132, 135, 136, 139–142, 150, 151, 153, 154, 156, 166, 167, 170, 173

SOAR State, Operator And Result 38, 41, 42, 44, 45

STM Short Term Memory 9, 34, 37, 39, 60, 66, 112, 114, 121, 123, 126, 128, 129

VR Virtual Reality 11, 61, 81, 92, 94, 100, 101, 112, 119, 135

W3C 75

WM Working memory 5, 9, 11, 39–42, 47, 52, 58, 60–64, 82, 86, 96, 121, 123–125, 127–130, 133, 135, 136, 139–141, 143, 145, 150, 151, 153, 154, 156, 160, 162, 165–167, 170, 171, 173

Bibliography

D. Abdlkarim, V. Ortenzi, T. Pardi, M. Filipovica, A. Wing, K. Kuchenbecker, and M. Di Luca. Prendosim: Proxy-hand-based robot grasp generator. pages 60–68, 01 2021. doi: 10.5220/0010549800600068.

N. Abdo, C. Stachniss, L. Spinello, and W. Burgard. Organizing objects by predicting user preferences through collaborative filtering. *The International Journal of Robotics Research*, 35(13):1587–1608, 2016. doi: 10.1177/0278364916649248. URL http://ijr.sagepub.com/content/35/13/1587.abstract.

J. Aggarwal and L. Xia. Human activity recognition from 3d data: A review. *Pattern Recognition Letters*, 48:70 – 80, 2014. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2014.04.011. URL http://www.sciencedirect.com/science/article/pii/S0167865514001299. Celebrating the life and work of Maria Petrou.

R. Alami, R. Chatila, A. Clodic, S. Fleury, M. Herrb, V. Montreuil, and E. A. Sisbot. Towards human-aware cognitive robots. In *AAAI, Stanford Spring Symposium*, 2006. read.

L. K. Alberts. *YMIR: An ontology for engineering design*. PhD thesis, Department Of Computer Science, Universiteit Twente, 1995.

J. S. Albus. 4d/rcs: a reference model architecture for intelligent unmanned ground vehicles. In *Unmanned Ground Vehicle Technology IV*, volume 4715, pages 303–310. International Society for Optics and Photonics, 2002.

J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of logic and computation*, 4(5):531–579, 1994.

L. Alpoim, A. F. da Silva, and C. P. Santos. Human activity recognition systems: State of art. In *2019 IEEE 6th Portuguese Meeting on Bioengineering (ENBENG)*, pages 1–4, February 2019. doi: 10.1109/ENBENG.2019.8692468. read.

E. Altuntas and A. Sekeroglu. Mechanical behavior and physical properties of chicken egg as affected by different egg weights. *Journal of Food Process Engineering*, 33:115 – 127, 07 2008. doi: 10.1111/j.1745-4530.2008.00263.x.

J. R. Anderson. Act: A simple theory of complex cognition. *American psychologist*, 51(4):355, 1996.

J. R. Anderson. *The architecture of cognition*. Psychology Press, 2013.

J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004. doi: 10.1037/0033-295X.111.4.1036.

M. Andres, B. Pelgrims, E. Olivier, and G. Vannuscorps. The left supramarginal gyrus contributes to finger positioning for object use: a neuronavigated transcranial magnetic stimulation study. *European Journal of Neuroscience*, 46(12):2835–2843, 2017. doi: 10.1111/ejn.13763.

B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009. ISSN 0921-8890. doi: http://doi.org/10.1016/j.robot.2008.10.024. URL `http://www.sciencedirect.com/science/article/pii/S0921889008001772`. read.

R. C. Arkin and D. C. MacKenzie. Planning to behave: A hybrid deliberative/reactive robot control architecture for mobile manipulation. *SMARTech*, 1994.

T. Asfour, M. Waechter, L. Kaul, S. Rader, P. Weiner, S. Ottenhaus, R. Grimm, Y. Zhou, M. Grotz, and F. Paus. Armar-6: A high-performance humanoid for human-robot collaboration in real-world scenarios. *IEEE Robotics Automation Magazine*, 26(4):108–121, 2019. doi: 10.1109/MRA.2019.2941246.

Z. Astolfi-Filho, E. B. De Oliveira, J. Coimbra, and J. telis Romero. Friction factors, convective heat transfer coefficients and the colburn analogy for industrial sugarcane juices. *Biochemical Engineering Journal*, 60:111–118, 01 2012. doi: 10.1016/j.bej.2011.10.011.

A. Baddeley. The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences*, 4(11):417–423, 2000. ISSN 1364-6613. doi: https://doi.org/10.1016/S1364-6613(00)01538-2. URL `http://www.sciencedirect.com/science/article/pii/S1364661300015382`.

M. Bajracharya, J. Borders, D. Helmick, T. Kollar, M. Laskey, J. Leichty, J. Ma, U. Nagarajan, A. Ochiai, J. Petersen, K. Shankar, K. Stone, and Y. Takaoka. A mobile manipulation system for one-shot teaching of complex tasks in homes. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11039–11045, 2020. doi: 10.1109/ICRA40945.2020.9196677.

L. W. Barsalou, C. Breazeal, and L. B. Smith. Cognition as coordinated non-cognition. *Cognitive Processing*, 8(2):79–91, 2007. ISSN 1612-4790. doi: 10.1007/s10339-007-0163-1. URL `http://dx.doi.org/10.1007/s10339-007-0163-1`.

B. Bäuml, F. Schmidt, T. Wimböck, O. Birbach, A. Dietrich, M. Fuchs, W. Friedl, U. Frese, C. Borst, M. Grebenstein, O. Eiberger, and G. Hirzinger. Catching flying balls and preparing coffee: Humanoid rollin'justin performs dynamic and sensitive tasks. In *IEEE International*

Conference on Robotics and Automation, pages 3443–3444, May 2011. doi: 10.1109/ICRA.2011.5980073. read.

P. Baxter and W. Browne. Memory-based cognitive framework: A low-level association approach to cognitive architectures. In G. Kampis, I. Karsai, and E. Szathmáry, editors, *Revised Selected Papers of the 10th European Conference on Advances in Artificial Life. Darwin Meets von Neumann (ECAL) Part I*, pages 402–409, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-21283-3. doi: 10.1007/978-3-642-21283-3_50. URL http://dx.doi.org/10.1007/978-3-642-21283-3_50. read.

S. Bedaf, G. J. Gelderblom, and L. de Witte. Overview and categorization of robots supporting independent living of elderly people: What activities do they support and how far have they developed. *Assistive Technology*, 27(2):88–100, 2015. doi: 10.1080/10400435.2014.978916. URL https://doi.org/10.1080/10400435.2014.978916. PMID: 26132353.

M. Beetz, J. Bandouch, D. Jain, and M. Tenorth. Towards automated models of activities of daily life. In *First International Symposium on Quality of Life Technology – Intelligent Systems for Better Living*, Pittsburgh, Pennsylvania USA, 2010a. read.

M. Beetz, L. Mösenlechner, and M. Tenorth. Cram – a cognitive robot abstract machine for everyday manipulation in human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1012–1017, 2010b. read.

M. Beetz, D. Jain, L. Mösenlechner, M. Tenorth, L. Kunze, N. Blodow, and D. Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471, 2012.

M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z. C. Márton. Robosherlock: Unstructured information processing for robot perception. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1549–1556, May 2015. doi: 10.1109/ICRA.2015.7139395. read.

M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoglu, and G. Bartels. Knowrob 2.0 – a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 2018.

M. Beetz, G. Kazhoyan, and D. Vernon. The cram cognitive architecture for robot manipulation in everyday activities, 2023.

P. Bellavista, A. Corradi, L. Foschini, and A. Pernafini. Data distribution service (dds): A performance comparison of opensplice and rti implementations. In *IEEE symposium on computers and communications (ISCC)*, pages 377–383. IEEE, 2013.

P. Benavidez, M. Kumar, S. Agaian, and M. Jamshidi. Design of a home multi-robot system for the elderly and disabled. In *10th System of Systems Engineering Conference (SoSE)*, pages 392–397, May 2015. doi: 10.1109/SYSOSE.2015.7151907. read.

A. Billard and D. Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446), 2019. ISSN 0036-8075. doi: 10.1126/science.aat8414. URL https://science.sciencemag.org/content/364/6446/eaat8414.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

B. S. Bloom, D. R. Krathwohl, and B. B. Masia. *Taxonomy of educational objectives*. David Mckay, 1971.

A. Bobick. Movement, activity and action: The role of knowledge in the perception of motion. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 352: 1257–65, September 1997. doi: 10.1098/rstb.1997.0108.

R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. pages 187–202, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-49594-9.

A. K. Bozcuoglu, G. Kazhoyan, Y. Furuta, S. Stelter, M. Beetz, K. Okada, and M. Inaba. The exchange of knowledge using cloud robotics. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4768–4775, 2018. doi: 10.1109/ICRA.2018.8460187.

C. Breazeal. Role of expressive behaviour for robots that learn from people. *Philosophical Transactions of the Royal Society B*, 364:3527–3538, 2009. doi: 10.1098/rstb.2009.0157. read.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

W. Bridewell and P. Bello. Incremental object perception in an attention-driven cognitive architecture. In *CogSci*, 2015.

A. G. Brooks, J. Gray, G. Hoffman, A. Lockerd, H. Lee, and C. Breazeal. Robot's play: Interactive games with sociable machines. *Computers in Entertainment*, 2(3):1–18, July 2004. ISSN 1544-3574. doi: 10.1145/1027154.1027171. URL http://doi.acm.org/10.1145/1027154.1027171. read.

R. Brooks. A hardware retargetable distributed layered architecture for mobile robot control. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 106–110. IEEE, 1987.

G.-P. Castellote and P. Bolton. Distributed real-time applications now have a data distribution protocol. *Real-Time Innovations, Inc*, 2002.

Y.-S. Cha, K. Kim, J.-Y. Lee, J. Lee, M. Choi, M.-H. Jeong, C. Kim, B.-J. You, and S.-R. Oh. Mahru-m: A mobile humanoid robot platform based on a dual-network control system and coordinated task execution. *Robotics and Autonomous Systems*, 59(6):354–66,

2011. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2011.01.003. URL http://www.sciencedirect.com/science/article/pii/S0921889011000108.

L. Chen and T. T. Rogers. Revisiting domain-general accounts of category specificity in mind and brain. *Wiley Interdisciplinary Reviews: Cognitive Science*, 5(3):327–344, 2014.

X. Chen, J. Xie, J. Ji, and Z. Sui. Toward open knowledge enabling for human-robot interaction. *Human-Robot Interaction*, 1(2):100–117, 2012. ISSN 2163-0364. doi: 10.5898/JHRI.1.2.Chen.

Y. Chen, F. Wu, W. Shuai, and X. Chen. Robots serve humans in public places-kejia robot as a shopping assistant. *International Journal of Advanced Robotic Systems*, 14 (3):1729881417703569, 2017. doi: 10.1177/1729881417703569. URL https://doi.org/10.1177/1729881417703569.

S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.

J.-W. Choi, G.-M. Park, and J.-H. Kim. Sr-em: episodic memory aware of semantic relations based on hierarchical clustering resonance network. *IEEE Transactions on Cybernetics*, pages 1–3, 2021. doi: 10.1109/TCYB.2021.3081762.

T. Christaller. Cognitive robotics: a new approach to artificial intelligence. *Artificial Life and Robotics*, 3(4):221–224, 1999. ISSN 1614-7456. doi: 10.1007/BF02481184. URL http://dx.doi.org/10.1007/BF02481184.

J. Claassens. The adhesion-cohesion, static friction and macrostructure of certain butters. iv. factors affecting static friction measurements of butter. *South African Journal of Agricultural Science*, 2(3):387–408, 1959.

A. Clark and R. Grush. Towards a cognitive robotics. *Adaptive Behavior*, 7(1):5–16, 1999. doi: 10.1177/105971239900700101. URL http://dx.doi.org/10.1177/105971239900700101.

H. M. Collins. Humans, machines, and the structure of knowledge. *Knowledge Management Tools*, pages 145–163, 1995.

M. Conforth and Y. Meng. Charisma: A context hierarchy-based cognitive architecture for self-motivated social agents. In *International Joint Conference on Neural Networks*, pages 1894–1901. IEEE, 2011.

M. T. Cox, Z. Alavi, D. Dannenhauer, V. Eyorokon, H. Munoz-Avila, and D. Perlis. Midca: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

F. Cruz, G. I. Parisi, and S. Wermter. Contextual affordances for action-effect prediction in a robotic-cleaning task. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

F. Cruz, G. I. Parisi, and S. Wermter. Learning contextual affordances with an associative neural architecture. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 665–670, 2016.

D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision (ECCV)*, 2018.

A. Darwiche and P. Marquis. A knowledge compilation map. *Artificial Intelligence Research*, 17:229–264, 2002.

J. P. Das, B. C. Kar, and R. K. Parrila. *Cognitive planning: The psychological basis of intelligent behavior*. Sage Publications, Inc, 1996.

P. Das, C. Xu, R. F. Doell, and J. J. Corso. A thousand frames in just a few words: Lingual description of videos through latent topics and sparse object stitching. In *IEEE conference on computer vision and pattern recognition*, pages 2634–2641, 2013.

R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? *AI Magazine*, 14 (1):17–33, 1993. doi: https://doi.org/10.1609/aimag.v14i1.1029.

E. De-La-Hoz-Franco, P. Ariza-Colpas, J. M. Quero, and M. Espinilla. Sensor-based datasets for human activity recognition – a systematic review of literature. *IEEE Access*, 6:59192–59210, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2873502.

A. K. Deklel, A. M. Hamdy, and E. M. Saad. Multi-objective symbolic regression using long-term artificial neural network memory (ltann-mem) and neural symbolization algorithm (nsa). *Neural Computing and Applications*, 29(4):935–942, Feb 2018. ISSN 1433-3058. doi: 10.1007/s00521-016-2500-8. URL `https://doi.org/10.1007/s00521-016-2500-8`.

T. Deutsch, A. Gruber, R. Lang, and R. Velik. Episodic memory for autonomous agents. In *Conference on Human System Interactions (HSI)*, pages 621–626, 2008.

D. Di Marco, P. Levi, R. van de Molengraft, and A. Clifford Perzylo. A deliberation layer for instantiating robot execution plans from abstract task descriptions. In *ICAPS*, 2013.

R. Diaz Flauzino, J. A. Wilhelms Gut, C. C. Tadini, and J. Telis-Romero. Flow properties and tube friction factor of milk cream: influence of temperature and fat content. *Journal of food process engineering*, 33(5):820–836, 2010.

M. Diehl, C. Paxton, and K. Ramirez-Amaro. Automated generation of robotic planning domains from observations. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6732–6738, 2021. doi: 10.1109/IROS51168.2021.9636781.

W. Dodd and R. Gutierrez. The role of episodic memory and emotion in a cognitive robot. In *IEEE International Workshop on Robots and Human Interactive Communication*, pages 692–697, 2005.

J. Elfring, S. Jansen, R. van de Molengraft, and M. Steinbuch. Active object search exploiting probabilistic object–object relations. In S. Behnke, M. Veloso, A. Visser, and R. Xiong, editors, *RoboCup 2013: Robot World Cup XVII*, pages 13–24, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44468-9.

S. Elfving, A. Dryden, B. Stanton, and K. Widström. Robots and robotic devices – vocabulary. Technical Report 8373:2012, ISO, March 2012.

J. Euzenat and P. Shvaiko. *Ontology matching*, volume 18. Springer, 2007.

C. Evans, M. G. Edwards, L. J. Taylor, and M. Ietswaart. Perceptual decisions regarding object manipulation are selectively impaired in apraxia or when tdcs is applied over the left ipl. *Neuropsychologia*, 86:153–166, 2016.

U. Faghihi, P. Fournier-Viger, and R. Nkambou. Celts: a cognitive tutoring agent with human-like learning capabilities and emotions. In *Intelligent and Adaptive Educational-Learning Systems*, pages 339–365. Springer, 2013.

T. Feix, J. Romero, H. Schmiedmayer, A. M. Dollar, and D. Kragic. The grasp taxonomy of human grasp types. *IEEE Transactions on Human-Machine Systems*, 46(1):66–77, February 2016. ISSN 2168-2291. doi: 10.1109/THMS.2015.2470657.

A. Ferrein, T. Niemueller, S. Schiffer, and G. Lakemeyer. Lessons learnt from developing the embodied ai platform caesar for domestic service robotics. In *AAAI Spring Symposium Series*, pages 21–26, 2013.

R. J. Firby. *Adaptive execution in complex dynamic worlds*. PhD thesis, Yale University, 1990.

J. R. Flanagan and R. S. Johansson. Action plans used in action observation. *Nature*, 424(6950): 769–771, 2003. doi: 10.1038/nature01861.

J. R. Flanagan, M. C. Bowman, and R. S. Johansson. Control strategies in object manipulation tasks. *Current Opinion in Neurobiology*, 16(6):650–659, 2006. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2006.10.005. URL `http://www.sciencedirect.com/science/article/pii/S0959438806001450`. Motor systems / Neurobiology of behaviour.

J. A. Fodors. *The Language of Thought*. Harvard University Press, 1975.

S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

E. Freud, T. Ganel, I. Shelef, M. D. Hammer, G. Avidan, and M. Behrmann. Three-dimensional representations of objects in dorsal cortex are dissociable from those in ventral cortex. *Cerebral Cortex*, 27(1):422–434, 10 2015. ISSN 1047-3211. doi: 10.1093/cercor/bhv229. URL `https://doi.org/10.1093/cercor/bhv229`.

J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

M. Gee, P. Tomlins, A. Calver, R. Darling, and M. Rides. A new friction measurement system for the frictional component of touch. *Wear*, 259(7):1437–1442, 2005. ISSN 0043-1648. doi: https://doi.org/10.1016/j.wear.2005.02.053. URL https://www.sciencedirect.com/science/article/pii/S0043164805001857. 15th International Conference on Wear of Materials.

T. Gijssels and D. Casasanto. Hand-use norms for dutch and english manual action verbs: Implicit measures from a pantomime task. *Behavior Research Methods*, 52(4):1–23, March 2020. doi: 10.3758/s13428-020-01347-x.

C. Goddard. *Minimal English for a global world*. Springer, 2018.

B. Goertzel. Cogprime: An integrative architecture for embodied artificial general intelligence. *Dynamical Psychology: An International, Interdisciplinary Journal of Complex Mental Processes*, 2012.

B. Goertzel and C. Pennachin. The novamente artificial intelligence engine. In *Artificial general intelligence*, pages 63–129. Springer, 2007.

J. B. Gordon, R. J. Passonneau, and S. L. Epstein. Learning to balance grounding rationales for dialogue systems. In *SIGDIAL 2011 Conference*, pages 266–271, 2011.

T. Gruber. It is what it does: The pragmatics of ontology for knowledge sharing. In *Proceedings of the International CIDOC CRM Symposium, Available online at*, 2003.

T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

N. Guarino, D. Oberle, and S. Staab. What is an ontology? In S. Staab and R. Studer, editors, *Handbook on Ontologies*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-540-92673-3. doi: 10.1007/978-3-540-92673-3_0. URL https://doi.org/10.1007/978-3-540-92673-3_0.

A. Haidu and M. Beetz. Action recognition and interpretation from virtual demonstrations. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2833–2838, Daejeon, South Korea, 2016. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7759439.

G. Handjaras, E. Ricciardi, A. Leo, A. Lenci, L. Cecchetti, M. Cosottini, G. Marotta, and P. Pietrini. How concepts are encoded in the human brain: A modality independent, category-based cortical organization of semantic knowledge. *NeuroImage*, 135:232 – 242, 2016. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.2016.04.063. URL http://www.sciencedirect.com/science/article/pii/S1053811916301021.

M. Hanheide and G. Sagerer. Active memory-based interaction strategies for learning-enabling behaviors. In *17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 101–106, August 2008. doi: 10.1109/ROMAN.2008.4600650.

K. Hashimoto, F. Saito, T. Yamamoto, and K. Ikeda. A field study of the human support robot in the home environment. In *IEEE Workshop on Advanced Robotics and its Social Impacts*, pages 143–150, November 2013. doi: 10.1109/ARSO.2013.6705520.

U. Hasson, I. Levy, M. Behrmann, T. Hendler, and R. Malach. Eccentricity bias as an organizing principle for human high-order object areas. *Neuron*, 34(3):479–490, 2002.

C. Havasi, R. Speer, and J. Alonso. Conceptnet 3: a flexible, multilingual semantic network for common sense knowledge. In *Recent Advances in Natural Language Processing*, September 2007.

C. Havasi, R. Speer, K. Arnold, H. Lieberman, J. Alonso, and J. Moeller. Open mind common sense: Crowd-sourcing for common sense. In *2nd AAAI Conference on Collaboratively-Built Knowledge Sources and Artificial Intelligence*, AAAIWS'10-02, pages 53–53. AAAI Press, 2010. URL `http://dl.acm.org/citation.cfm?id=2908523.2908534`.

S. Haykin. *Neural Networks and Learning Machines, 3/E*. Pearson Education India, 2010.

M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

M. Henaff, A. Szlam, and Y. LeCun. Recurrent orthogonal networks and long-memory tasks. In *33rd International Conference on International Conference on Machine Learning*, volume 48 of *ICML'16*, pages 2034–2042. JMLR.org, 2016. URL `http://dl.acm.org/citation.cfm?id=3045390.3045605`.

T. C. Henderson, A. Joshi, and W. Wang. The cognitive symmetry engine. Technical report, School of Computer Science, University Utah, Salt Lake City, USA, Rep. UUCS-13-004, 2013.

T. Hobbes. *Leviathan*. Andrew Crooke, 1651.

L. Iocchi, G. Kraetzschmar, D. Nardi, P. U. Lima, P. Miraldo, E. Bastianelli, and R. Capobianco. *RoCKIn@Home: Domestic Robots Challenge*. In tech open, August 2017. doi: 10.5772/intechopen.70015.

H. Iwata and S. Sugano. Design of human symbiotic robot twendy-one. In *IEEE International Conference on Robotics and Automation*, pages 580–586, May 2009. doi: 10.1109/ROBOT.2009.5152702.

R. S. Johansson and J. R. Flanagan. Coding and use of tactile signals from the fingertips in object manipulation tasks. *Nature Reviews Neuroscience*, 10:345, 2009. doi: 10.1038/nrn2621.

A. Johnson. *Procedural Memory and Skill Acquisition*, chapter 18. American Cancer Society, 2012. ISBN 9781118133880. doi: 10.1002/9781118133880.hop204018. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118133880.hop204018.

M. J. Johnson, M. A. Johnson, J. S. Sefcik, P. Z. Cacchione, C. Mucchiani, T. Lau, and M. Yim. Task and design requirements for an affordable mobile service robot for elder care in an all-inclusive care for elders assisted-living setting. *International Journal of Social Robotics*, Nov 2017. ISSN 1875-4805. doi: 10.1007/s12369-017-0436-5. URL https://doi.org/10.1007/s12369-017-0436-5.

K. Kaneko, F. Kanehiro, M. Morisawa, K. Akachi, G. Miyamori, A. Hayashi, and N. Kanehira. Humanoid robot hrp-4 - humanoid robotics platform with lightweight and slim body. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4400–4407, September 2011. doi: 10.1109/IROS.2011.6094465.

K. Kawamura, S. M. Gordon, P. Ratanaswasd, E. Erdemir, and J. F. Hall. Implementation of cognitive control for a humanoid robot. *International Journal of Humanoid Robotics*, 5(4): 547–586, 2008.

G. Kazhoyan and M. Beetz. Programming robotic agents with action descriptions. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 103–108, 2017. doi: 10.1109/IROS.2017.8202144.

G. Kazhoyan and M. Beetz. Executing underspecified actions in real world based on online projection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5156–5163, 2019. doi: 10.1109/IROS40897.2019.8967867.

G. Kazhoyan, A. Hawkin, S. Koralewski, A. Haidu, and M. Beetz. Learning motion parameterizations of mobile pick and place actions from observing humans in virtual environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9736–9743, 2020a. doi: 10.1109/IROS45743.2020.9341458.

G. Kazhoyan, A. Niedzwiecki, and M. Beetz. Towards plan transformations for real-world mobile fetch and place. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11011–11017, 2020b. doi: 10.1109/ICRA40945.2020.9197446.

G. Kazhoyan, S. Stelter, F. K. Kenfack, S. Koralewski, and M. Beetz. The robot household marathon experiment. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9382–9388, 2021. doi: 10.1109/ICRA48506.2021.9560774.

F. Kenghagho Kenfack, M. Neumann, P. Mania, T. Tan, F. Siddiky, R. Weller, G. Zachmann, and M. Beetz. Naivphys4rp -towards human-like robot perception: "physical reasoning based on embodied probabilistic simulation". 11 2022. doi: 10.1109/Humanoids53995.2022.10000153.

M. Kerzel, E. Strahl, S. Magg, N. Navarro-Guerrero, S. Heinrich, and S. Wermter. Nico – neuro-inspired companion: A developmental humanoid robot platform for multimodal interaction. In

2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pages 113–120, August 2017. doi: 10.1109/ROMAN.2017.8172289.

Ö. Kiliç and P. Wang. Nars as a normative model of cognition. In *CogSci*, 2015.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, pages 340–347, New York, NY, USA, 1997. ACM. ISBN 0-89791-877-0. doi: 10.1145/267658.267738. URL http://doi.acm.org/10.1145/267658.267738.

B. Kokinov, V. Nikolov, and A. Petrov. Dynamics of emergent computation in dual. *Artificial intelligence: methodology, systems, applications. IOS Press, Amsterdam*, 1996.

T. Konkle and A. Oliva. A familiar-size stroop effect: real-world size is an automatic property of object representation. *Journal of Experimental Psychology: Human Perception and Performance*, 38(3):561–569, 2012.

S. Koralewski, G. Kazhoyan, and M. Beetz. Self-specialization of general robot plans based on experience. *IEEE Robotics and Automation Letters*, 4(4):3766–3773, 2019. doi: 10.1109/LRA.2019.2928771.

I. Kotseruba and J. K. Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 2018. doi: 10.1007/s10462-018-9646-y.

J. Laaksonen and E. Oja. Classification with learning k-nearest neighbors. In *Proceedings of international conference on neural networks (ICNN'96)*, volume 3, pages 1480–1483. IEEE, 1996.

J. Laird. Recovery from incorrect knowledge in soar. In *Seventh AAAI National Conference on Artificial Intelligence*, AAAI'88, pages 618–623. AAAI Press, 1988. URL http://dl.acm.org/citation.cfm?id=2887965.2888075.

J. E. Laird and S. Mohan. A case study of knowledge integration across multiple memories in soar. *Biologically Inspired Cognitive Architectures*, 8:93–99, 2014. ISSN 2212-683X. doi: http://dx.doi.org/10.1016/j.bica.2014.03.006. URL http://www.sciencedirect.com/science/article/pii/S2212683X14000164.

A. Lally, S. Bachi, M. A. Barborak, D. W. Buchanan, J. Chu-Carroll, D. A. Ferrucci*, M. R. Glass, A. Kalyanpur, E. T. Mueller, J. W. Murdock, S. Patwardhan, J. M. Prager, and C. A. Welty. Watsonpaths: Scenario-based question answering and inference over unstructured information. IBM Research Report RC25489 (WAT1409-048), IBM Research Division, Yorktown Heights, NY 10598, September 2014.

C. Lebiere, F. Jentsch, and S. Ososky. Cognitive models of decision making processes for human-robot interaction. In *International Conference on Virtual, Augmented and Mixed Reality*, pages 285–294. Springer, 2013.

Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. Efficient backprop. In G. B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_2. URL http://dx.doi.org/10.1007/3-540-49430-8_2.

S. Lemaignan, R. Ros, L. Mösenlechner, R. Alami, and M. Beetz. Oro, a knowledge management platform for cognitive architectures in robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3548–3553, 2010. doi: 10.1109/IROS.2010.5649547.

I. Levy, U. Hasson, G. Avidan, T. Hendler, and R. Malach. Center–periphery organization of human object areas. *Nature Neuroscience*, 4(5):533–539, 2001. ISSN 1546-1726. doi: 10.1038/87490.

R. Lewis, C. Menardi, A. Yoxall, and J. Langley. Finger friction: Grip and opening packaging. *Wear*, 263(7):1124–1132, 2007. ISSN 0043-1648. doi: https://doi.org/10.1016/j.wear.2006.12.024. URL https://www.sciencedirect.com/science/article/pii/S0043164807003080. 16th International Conference on Wear of Materials.

W. C. Lewis, M. Moll, and L. E. Kavraki. How much do unstated problem constraints limit deep robotic reinforcement learning? 2019. doi: 10.25611/AZ5Z-XT37. URL http://hdl.handle.net/1911/107403.

P. Lima, D. Nardi, G. Kraetzschmar, R. Bischoff, and M. Matteucci. Rockin and the european robotics league: Building on robocup best practices to promote robot competitions in europe. In *RoboCup 2016: Robot World Cup XX*, pages 181–192, November 2017. ISBN 978-3-319-68791-9. doi: 10.1007/978-3-319-68792-6_15.

M. Lloyd-Kelly, F. Gobet, and P. C. Lane. Piece of mind: Long-term memory structure in act-r and chrest. In *CogSci*, 2015.

Q. T. C. Ltd. Sanbot (robot). https://www.wikiwand.com/en/Sanbot_(robot), 2019.

N. Lucci, G. F. Preziosa, and A. M. Zanchettin. Learning human actions semantics in virtual reality for a better human-robot collaboration. In *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 785–791, 2022. doi: 10.1109/RO-MAN53752.2022.9900824.

J. A. Lum, G. Conti-Ramsden, D. Page, and M. T. Ullman. Working, declarative and procedural memory in specific language impairment. *Cortex*, 48(9):1138–1154, 2012. ISSN 0010-9452. doi: https://doi.org/10.1016/j.cortex.2011.06.001. URL http://www.sciencedirect.com/science/article/pii/S0010945211001705.

R. C. Luo, J.-W. Zhan, W.-H. Cheng, and N.-W. Chang. Network-based multimodal human-robot interactions in ubiquitous computing environment. In *IEEE International Conference on Robotics and Biomimetics*, pages 131–136, 2008.

W. Lutz, W. Sanderson, and S. Scherbov. The coming acceleration of global population ageing. *Nature*, 451:716, 2008. doi: 10.1038/nature06516.

C. Magri, B. Long, R. Chiou, and T. Konkle. Behavioral and neural associations between object size and curvature. *Journal of Vision*, 19(10):30c, 2019. doi: 10.1167/19.10.30c.

B. Z. Mahon. Missed connections: A connectivity constrained account of the representation and organization of object concepts. *The conceptual mind: New directions in the study of concepts*, 79, 2015.

B. Z. Mahon and A. Caramazza. What drives the organization of object knowledge in the brain? *Trends in cognitive sciences*, 15(3):97–103, 2011.

J. B. Marshall. Metacat: A self-watching cognitive architecture for analogy-making. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 24, 2002.

A. Martin. Circuits in mind: The neural foundations for object concepts. 2009.

E. Martinez-Martin and A. P. del Pobil. Personal robot assistants for elderly care: An overview. In A. Costa, V. Julian, and P. Novais, editors, *Personal Assistants: Emerging Computational Technologies*, pages 77–91. Springer International Publishing, Cham, 2018. ISBN 978-3-319-62530-0. doi: 10.1007/978-3-319-62530-0_5. URL `https://doi.org/10.1007/978-3-319-62530-0_5`.

M. Matamoros and M. Luz. Robocup@home scoring history. `https://github.com/RoboCupAtHome/AtHomeCommunityWiki/wiki/Scores`, 2018.

M. Matamoros, C. Rascon, S. Wachsmuth, A. W. Moriarty, J. Kummert, J. Hart, S. Pfeiffer, M. van der Brugh, and M. St-Pierre. Robocup@home 2019: Rules and regulations (draft). `http://www.robocupathome.org/rules/2019_rulebook.pdf`, 2019.

D. L. McGuinness and F. van Harmelen. Owl web ontology language-reference. *W3C Recommendation [Online], Available at: http://www. w3. org/TR/2004/REC-owl*, pages 1–53, 2004.

A. Mehra and M. Nissen. Case study: Intelligent software supply chain agents using ade. *AAAI Technical Report WS-98-10*, 1998.

G. Metta, P. Fitzpatrick, and L. Natale. Yarp: yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1):8, 2006.

S. Meyer zu Borgsen, T. Korthals, F. Lier, and S. Wachsmuth. Tobi – team of bielefeld: Enhancing robot behaviors and the role of multi-robotics in RoboCup@Home. In S. Behnke, R. Sheh, S. Sarıel, and D. D. Lee, editors, *RoboCup 2016: Robot World Cup XX*, pages 577–588, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68792-6.

G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11): 39–41, 1995.

M. Minsky. A framework for representing knowledge. *AI Memos*, 1974.

J. W. Murdock and A. K. Goel. Meta-case-based reasoning: self-improvement through self-understanding. *Journal of Experimental & Theoretical Artificial Intelligence*, 20(1):1–36, 2008.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(82)90012-1. URL http://www.sciencedirect.com/science/article/pii/0004370282900121.

A. Newell, P. S. Rosenbloom, and J. E. Laird. Foundations of cognitive science. In M. I. Posner, editor, *Symbolic Architectures for Cognition*, pages 93–131. MIT Press, Cambridge, MA, USA, 1989. ISBN 0-262-16112-5. URL http://dl.acm.org/citation.cfm?id=102953.102956.

G. W. Ng, Y.-S. Tan, X. H. Xiao, and R. Z. Chan. Dso cognitive architecture in mobile surveillance. In *Workshop on Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pages 111–115. IEEE, 2012.

N. J. Nilsson. Shakey the robot. Technical Report 323, Artificial Intelligence Center, Computer Science and Technology Division, 1984.

N. J. Nilsson. *Introduction to Machine Learning*. Stanford University, 1996.

R. Novianto and M.-A. Williams. Operant conditioning in asmo cognitive architecture. *Biologically Inspired Cognitive Architecture*, 2014.

A. Nuxoll and J. E. Laird. A cognitive model of episodic memory integrated with a general cognitive architecture. In *International Conference on Cognitive Modeling*, pages 220–225, 2004.

A. M. Nuxoll and J. E. Laird. Extending cognitive architecture with episodic memory. In *22nd National Conference on Artificial Intelligence*, volume 2 of *AAAI'07*, pages 1560–1565. AAAI Press, 2007. ISBN 978-1-57735-323-2. URL http://dl.acm.org/citation.cfm?id=1619797.1619895.

A. M. Nuxoll and J. E. Laird. Enhancing intelligent agents with episodic memory. *Cognitive Systems Research*, 17:34–48, 2012. ISSN 1389-0417. doi: http://dx.doi.org/10.1016/j.cogsys.2011.10.002. URL http://www.sciencedirect.com/science/article/pii/S1389041711000428.

N. Nwakuba, O. Chukwuezie, F. Uzoigwe, and P. Chukwu. Friction coefficients of local food grains on different structural surfaces. *Journal of Engineering Research and Reports*, 08 2019. doi: 10.9734/jerr/2019/v6i316951.

D. Nyga and M. Beetz. Cloud-based probabilistic knowledge services for instruction interpretation. In A. Bicchi and W. Burgard, editors, *Robotics Research*, volume 2, pages 649–664. Springer International Publishing, Cham, 2018. ISBN 978-3-319-60916-4. doi: 10.1007/978-3-319-60916-4_37. URL https://doi.org/10.1007/978-3-319-60916-4_37.

D. Nyga, M. Picklum, S. Koralewski, and M. Beetz. Instruction completion through instance-based learning and semantic analogical reasoning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4270–4277, 2017. doi: 10.1109/ICRA.2017.7989491.

G. H. Ogasawara. A distributed, decision-theoretic control system for a mobile robot. *ACM SIGART Bulletin*, 2(4):140–145, 1991.

M. Ojala and G. C. Garriga. Permutation tests for studying classifier performance. *Journal of machine learning research*, 11(6), 2010.

H. Okada, J. T. Chuan Tan, M. Chacin, H. Tamukoh, L. Iocchi, M. Mizukawa, M. Rajesh Elara, M. Sano, R. Mochizuki, S. Owada, T. Nagai, T. Nakayama, T. Ohashi, T. Matsui, T. Inamura, Y. Hagiwara, Y. Inoue, and Y. Ishida. Partner robot challenge real space: Rules & regulations. Technical Report Rev-2.6, World Robot Challenge, October 2018.

S. Y. Okita, V. Ng-Thow-Hing, and R. Sarvadevabhatla. Learning together: Asimo developing an interactive learning partnership with children. In *The 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 1125–1130, September 2009. doi: 10.1109/ROMAN.2009.5326135.

G. B. Orr and K.-R. Müller. *Neural networks: tricks of the trade*. Springer, 2003.

D. N. Osherson and E. E. Smith. On the adequacy of prototype theory as a theory of concepts. *Cognition*, 9(1):35–58, 1981.

J. Pages, L. Marchionni, and L. Ferro. Tiago: the modular robot that adapts to different research needs. International Workshop on Robot Modularity, IROS, 2016.

A. K. Pandey. Socially intelligent robots, the next generation of consumer robots and the challenges. In G. Stojanov and A. Kulakov, editors, *ICT Innovations 2016*, pages 41–46, Cham, 2018. Springer International Publishing. ISBN 978-3-319-68855-8.

G. Pardo-Castellote and S. Schneider. The network data delivery service: A real-time data connectivity system. In *IEEE International Conference on Robotics and Automation*, pages 2870–2876. IEEE, 1994.

K. Patterson, P. J. Nestor, and T. T. Rogers. Where do you know what you know? the representation of semantic knowledge in the human brain. *Nature Reviews Neuroscience*, 8: 976, 2007. doi: 10.1038/nrn2277.

D. Paulius and Y. Sun. A survey of knowledge representation in service robotics. *Robotics and Autonomous Systems*, 118:13–30, August 2019. ISSN 0921-8890. doi: 10.1016/j.robot.2019.03.005. URL http://dx.doi.org/10.1016/j.robot.2019.03.005.

B. Pause, A. Zlomuzica, K. Kinugawa, J. Mariani, R. Pietrowsky, and E. Dere. Perspectives on episodic-like and episodic memory. *Frontiers in Behavioral Neuroscience*, 7, 2013. ISSN 1662-5153. doi: 10.3389/fnbeh.2013.00033.

D. Pecher and R. A. Zwaan. *Grounding cognition: The role of perception and action in memory, language, and thinking*. Cambridge University Press, 2005.

B. Peeters. *Semantic primes and universal grammar: Empirical evidence from the Romance languages*, volume 81. John Benjamins Publishing, 2006.

B. Pelgrims, E. Olivier, and M. Andres. Dissociation between manipulation and conceptual knowledge of object use in the supramarginalis gyrus. *Human brain mapping*, 32(11): 1802–1810, 2011.

G. Pezzulo, G. Calvi, and C. Castelfranchi. Dipra: Distributed practical reasoning architecture. In *IJCAI*, pages 1458–1463, 2007.

E. Phillips, K. E. Schaefer, D. R. Billings, F. Jentsch, and P. A. Hancock. Human-animal teams as an analog for future human-robot teams: Influencing design and fostering trust. *J. Hum.-Robot Interact.*, 5(1):100–125, Mar. 2016. ISSN 2163-0364. doi: 10.5898/JHRI.5.1.Phillips. URL https://doi.org/10.5898/JHRI.5.1.Phillips.

J. L. Phillips and D. C. Noelle. A biologically inspired working memory framework for robots. In *IEEE International Workshop on Robot and Human Interactive Communication (ROMAN)*, pages 599–604, August 2005. doi: 10.1109/ROMAN.2005.1513845.

L. A. Pineda, I. V. Meza, H. Avilés, C. Gershenson, C. Rascon, M. Alvarado, and L. Salinas. Ioca: An interaction-oriented cognitive architecture. In *Journal Research in Computing Science*, Special Issue on Artificial Intelligence, pages 273–284, 2010. ISBN 1870-4069.

L. A. Pineda, L. Salinas, I. V. Meza, C. Rascon, and G. Fuentes. Sitlog: A programming language for service robot tasks. *International Journal of Advanced Robotic Systems*, 10 (358):1–12, 2013. ISSN 1729-8806. doi: 10.5772/56906.

L. A. Pineda, A. Rodríguez, G. Fuentes, C. Rascón, and I. V. Meza. Concept and functional structure of a service robot. *International Journal of Advanced Robotic Systems*, 12(6):1–15, 2015. doi: 10.5772/60026.

G. Pobric, E. Jefferies, and M. A. Lambon Ralph. Amodal semantic representations depend on both anterior temporal lobes: evidence from repetitive transcranial magnetic stimulation. *Neuropsychologia*, 48(5):1336–1342, 2010.

L. Postman. Toward a general theory of cognition. *Social psychology at the crossroads*, pages 242–272, 1951.

J.-Y. Puigbo, A. Pumarola, and R. A. Téllez. Controlling a general purpose service robot by means of a cognitive architecture. In *AIC@AI*IA*, 2013.

M. Puskaric, B. von Helversen, and J. Rieskamp. How social and non-social information influence classification decisions: A computational modelling approach. *The Quarterly Journal of Experimental Psychology*, 70(8):1516–1534, 2017.

D. V. Pynadath, P. S. Rosenbloom, and S. C. Marsella. Reinforcement learning for adaptive theory of mind in the sigma cognitive architecture. In *International Conference on Artificial General Intelligence*, pages 143–154. Springer, 2014.

C. Qixin, Z. Zhen, and G. Jiajun. A distributed control and simulation system for dual arm mobile robot. In *International Symposium on Computational Intelligence in Robotics and Automation*, pages 450–455, June 2007. doi: 10.1109/CIRA.2007.382851.

M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

J. R. Quinlan. *C4.5: programs for machine learning*. Elsevier, 1993.

U. Ramamurthy and B. J. Baars. Lida: A working model of cognition. In *International Conference on Cognitive Modeling*, 2006.

C. Ramírez and B. Valdes. A general knowledge representation model of concepts. In C. Ramírez, editor, *Advances in Knowledge Representation*, pages 43–76. InTech, 2012. ISBN 978-953-51-0597-8.

K. Ramírez Amaro. *Inferring Human Activities from Observation via Semantic Reasoning: A novel method for transferring skills to robots*. PhD thesis, Fakultät für Elektrotechnik und Informationstechnik, Technischen Universität München, 2014.

K. Ramirez-Amaro, T. Inamura, E. Dean-León, M. Beetz, and G. Cheng. Bootstrapping humanoid robot skills by extracting semantic representations of human-like activities from virtual reality. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 438–443, November 2014. doi: 10.1109/HUMANOIDS.2014.7041398.

A. Remazeilles, C. Leroux, and G. Chalubert. Sam: A robotic butler for handicapped people. In *The 17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 315–321, August 2008. doi: 10.1109/ROMAN.2008.4600685.

J. Richter-Klug, P. Mania, G. Kazhoyan, M. Beetz, and U. Frese. Improving object pose estimation by fusion with a multimodal prior – utilizing uncertainty-based cnn pipelines for robotics. *IEEE Robotics and Automation Letters*, 7(2):2282–2288, 2022. doi: 10.1109/LRA.2022.3140450.

M. Riesenhuber. Appearance isn't everything: news on object representation in cortex. *Neuron*, 55(3):341–344, 2007.

M. Riesenhuber. How the mind sees the world. *Nature Human Behaviour*, 4(11):1100–1101, 2020. doi: 10.1038/s41562-020-00973-x.

S. J. Rigatti. Random forest. *Journal of Insurance Medicine*, 47(1):31–39, 2017.

F. E. Ritter, J. L. Bittner, S. E. Kase, R. Evertsz, M. Pedrotti, and P. Busetta. Cojack: A high-level cognitive architecture with demonstrations of moderators, variability, and implications for situation awareness. *Biologically Inspired Cognitive Architectures*, 1:2–13, 2012.

M. Rohrbach, S. Amin, M. Andriluka, and B. Schiele. A database for fine grained activity detection of cooking activities. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1194–1201, June 2012. doi: 10.1109/CVPR.2012.6247801.

A. Romero-Garcés, L. V. Calderita, J. Martınez-Gómez, J. P. Bandera, R. Marfil, L. J. Manso, P. Bustos, and A. Bandera. The cognitive architecture of a robotic salesman. *environment*, 15 (6):16, 2015.

D. A. Rosenbaum, K. M. Chapman, M. Weigelt, D. J. Weiss, and R. van der Wel. Cognition, action, and object manipulation. *Psychological bulletin*, 138(5):924–946, 2012. doi: doi: 10.1037/a0027839.

J. Ruesch, M. Lopes, A. Bernardino, J. Hornstein, J. Santos-Victor, and R. Pfeifer. Multimodal saliency-based bottom-up attention a framework for the humanoid robot icub. In *IEEE International Conference on Robotics and Automation*, pages 962–967. IEEE, 2008.

S. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2003.

M. S. Ryoo and J. K. Aggarwal. Semantic representation and recognition of continued and recursive human activities. *International Journal of Computer Vision*, 82(1):1–24, Apr 2009. ISSN 1573-1405. doi: 10.1007/s11263-008-0181-1. URL `https://doi.org/10.1007/s11263-008-0181-1`.

Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2478–2483, September 2002. doi: 10.1109/IRDS.2002.1041641.

K. S. Saladin. *Anatomy & physiology: the unity of form and function*. McGraw-Hill Higher Education, 2004.

R. Salgado, F. Bellas, P. Caamaño, B. Santos-Díez, and R. J. Duro. A procedural long term memory for cognitive robotics. In *IEEE Conference on Evolving and Adaptive Intelligent Systems*, pages 57–62, May 2012. doi: 10.1109/EAIS.2012.6232805.

L. Salinas Pinacho. From knowledge of humans performing everyday activities to the service robots. Extended abstract on the "Recent Trends in Knowledge Compilation (Dagstuhl Seminar 17381)" 9, Dagstuhl, Germany, 2018. URL http://drops.dagstuhl.de/opus/volltexte/2018/8589.

L. Salinas Pinacho and M. Beetz. Object grasping and arrangements extracted from virtual reality for a household robot. Poster on the "workshop on concepts in action: Representation, learning, and application", 2018. URL https://conceptresearch.github.io/CARLA/files/carla_2018/Salinas.pdf.

L. Salinas Pinacho, A. Wich, F. Yazdani, and M. Beetz. Acquiring knowledge of object arrangements from human examples for household robots. In F. Trollmann and A.-Y. Turhan, editors, *KI 2018: Advances in Artificial Intelligence*, pages 131–138, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00111-7.

A. V. Samsonovich, K. A. de Jong, and A. Kitsantas. The mental state formalism of gmu-bica. *International Journal of Machine Consciousness*, 1(1):111–130, 2009.

J. Savage, M. Negrete, M. Matamoros, J. Cruz, R. Lagunas, J. Marquez, J. Marentes, and F. Villasenor. The role of robotics competitions for the development of service robots. Workshop on Autonomous Mobile Service Robots, IJCAI, 2016.

M. Scheutz, T. Williams, E. Krause, B. Oosterveld, V. Sarathy, and T. Frasca. An overview of the distributed integrated cognition affect and reflection diarc architecture. In M. I. Aldinhas Ferreira, J. Silva Sequeira, and R. Ventura, editors, *Cognitive Architectures*, pages 165–193. Springer International Publishing, Cham, 2019. ISBN 978-3-319-97550-4. doi: 10.1007/978-3-319-97550-4_11. URL https://doi.org/10.1007/978-3-319-97550-4_11.

M. Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106, 2004.

T. Segaran, C. Evans, and J. Taylor. *Programming the Semantic Web: Build Flexible Applications with Graph Data*. OŔeilly Media, Inc., 2009.

V. Seib, F. Polster, and D. Paulus. Evaluation of approaches combining 2d and 3d data for object recognition developed for the mobile robot lisa. In *Proceedings Volume 10253, 2016 International Conference on Robotics and Machine Vision*, volume 10253, pages 102531–10255, 2017. doi: 10.1117/12.2266256. URL https://doi.org/10.1117/12.2266256.

M. Seo, S. Han, K. Sim, S. H. Bang, C. Gonzalez, L. Sentis, and Y. Zhu. Deep imitation learning for humanoid loco-manipulation through human teleoperation, 2023.

W. Shuai and X.-p. Chen. Kejia: towards an autonomous service robot with tolerance of unexpected environmental changes. *Frontiers of Information Technology & Electronic Engineering*, 20(3):307–317, March 2019. ISSN 2095-9230. doi: 10.1631/FITEE.1900096. URL https://doi.org/10.1631/FITEE.1900096.

B. Siciliano and O. Khatib. *Handbook of Robotics*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 354023957X.

F. A. Siddiky, M. S. Alam, and T. Ahsan. High performance reliable obstacle detection and height measurement by stereo camera for intelligent home service robot. In *10th international conference on computer and information technology*, pages 1–6, December 2007. doi: 10.1109/ICCITECHN.2007.4579364.

L. Sigal, A. O. Balan, and M. J. Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International journal of computer vision*, 87(1-2):4, 2010.

M. Sigalas, M. Maniadakis, and P. Trahanias. Time-aware long-term episodic memory for recurring hri. In *ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, pages 287–288, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4885-0. doi: 10.1145/3029798.3038307. URL http://doi.acm.org/10.1145/3029798.3038307.

A. Sloman. Cogaff architecture schema. *Presented at Gatsby Computational Neuroscience Unit*, 2002.

K. Song, S. Jiang, and S. Wu. Safe guidance for a walking-assistant robot using gait estimation and obstacle avoidance. *IEEE/ASME Transactions on Mechatronics*, 22(5):2070–2078, October 2017. ISSN 1083-4435. doi: 10.1109/TMECH.2017.2742545.

K. Srihasam, J. L. Vincent, and M. S. Livingstone. Novel domain formation reveals proto-architecture in inferotemporal cortex. *Nature neuroscience*, 17(12):1776–1783, 2014.

S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5, November 2009. ISSN 1573-7527. doi: 10.1007/s10514-009-9160-9. URL https://doi.org/10.1007/s10514-009-9160-9.

C. Stahl, D. Anastasiou, and T. Latour. Social telepresence robots: The role of gesture for collaboration over a distance. In *Proceedings of the 11th PErvasive Technologies Related to Assistive Environments Conference*, PETRA '18, pages 409–414, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6390-7. doi: 10.1145/3197768.3203180. URL http://doi.acm.org/10.1145/3197768.3203180.

J. A. Starzyk and J. Graham. Mlecog: Motivated learning embodied cognitive architecture. *IEEE Systems Journal*, 11(3):1272–1283, 2015.

J. Stückler, D. Holz, and S. Behnke. Demonstrating everyday manipulation skills in RoboCup@Home. *IEEE Robotics & Automation Magazine*, 19(2):34–42, June 2012.

F. Stulp, A. Fedrizzi, L. Mösenlechner, and M. Beetz. Learning and reasoning with action-related places for robust mobile manipulation. *Artificial Intelligence Research*, 43(1):1–42, January 2012. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=2387915.2387916.

S. Subramanian and R. Viswanathan. Bulk density and friction coefficients of selected minor millet grains and flours. *Journal of Food Engineering*, 81(1):118–126, 2007. ISSN 0260-8774. doi: https://doi.org/10.1016/j.jfoodeng.2006.09.026. URL https://www.sciencedirect.com/science/article/pii/S0260877406006479.

R. Sun. Desiderata for cognitive architectures. *Philosophical Psychology*, 17(3):341–373, 2004. doi: 10.1080/0951508042000286721. URL http://dx.doi.org/10.1080/0951508042000286721.

R. Sun. *Anatomy of the Mind: Exploring Psychological Mechanisms and Processes with The: Exploring Psychological Mechanisms and Processes with the Clarion Cognitive Architecture*. Oxford University Press USA, 2016.

T. Taipalus and K. Kosuge. Development of service robot for fetching objects in home environment. In *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA*, pages 451–456, July 2005. ISBN 0-7803-9355-4. doi: 10.1109/CIRA.2005.1554318.

K. Takaya, T. Asai, V. Kroumov, and F. Smarandache. Simulation environment for mobile robots testing using ros and gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 96–101. IEEE, 2016.

A. T. G. Tapeh and M. Z. Naser. Artificial intelligence, machine learning, and deep learning in structural engineering: A scientometrics review of trends and best practices. *Archives of Computational Methods in Engineering*, 30(1):115–159, 2023. ISSN 1886-1784. doi: 10.1007/s11831-022-09793-w.

J. Telis-Romero, V. Telis, and F. Yamashita. Friction factors and rheological properties of orange juice. *Journal of Food Engineering*, 40(1):101–106, 1999. ISSN 0260-8774. doi: https://doi.org/10.1016/S0260-8774(99)00045-X. URL https://www.sciencedirect.com/science/article/pii/S026087749900045X.

M. Tenorth and M. Beetz. Knowrob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research*, 32(5):566–590, 2013. doi: 10.1177/0278364913481635.

M. Tenorth and M. Beetz. Representations for robot knowledge in the knowrob framework. *Artificial Intelligence*, 247(Supplement C):151–169, 2017. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2015.05.010. URL http://www.sciencedirect.com/science/article/pii/S0004370215000843. Special Issue on AI and Robotics.

M. Tenorth, J. Bandouch, and M. Beetz. The tum kitchen data set of everyday manipulation activities for motion tracking and action recognition. In *IEEE 12th International Conference on Computer Vision Workshops (ICCV)*, pages 1089–1096, September 2009. doi: 10.1109/ICCVW.2009.5457583.

M. Tenorth, D. Jain, and M. Beetz. Knowledge processing for cognitive robots. *KI - Künstliche Intelligenz*, 24(3):233–240, 2010a. ISSN 1610-1987. doi: 10.1007/s13218-010-0044-0. URL http://dx.doi.org/10.1007/s13218-010-0044-0.

M. Tenorth, L. Kunze, D. Jain, and M. Beetz. Knowrob-map - knowledge-linked semantic object maps. In *Humanoids*, 2010b.

M. Tenorth, A. C. Perzylo, R. Lafrenz, and M. Beetz. Representation and exchange of knowledge about actions, objects, and environments in the roboearth framework. *IEEE Transactions on Automation Science and Engineering*, 10(3):643–651, July 2013. ISSN 1545-5955. doi: 10.1109/TASE.2013.2244883.

M. Tenorth, G. Bartels, and M. Beetz. Knowledge-based specification of robot motions. In *Twenty-first European Conference on Artificial Intelligence*, ECAI'14, pages 873–878, Amsterdam, The Netherlands, The Netherlands, 2014. IOS Press. ISBN 978-1-61499-418-3. doi: 10.3233/978-1-61499-419-0-873. URL https://doi.org/10.3233/978-1-61499-419-0-873.

M. M. Tenorth. *Knowledge Processing for Autonomous Robots*. PhD thesis, Fakultät für Informatik, Technischen Universität München, 2011.

J.-P. Thibaut and L. Toussaint. Developing motor planning over ages. *Journal of Experimental Child Psychology*, 105(1):116–129, 2010. ISSN 0022-0965. doi: https://doi.org/10.1016/j.jecp.2009.10.003. URL http://www.sciencedirect.com/science/article/pii/S0022096509001817.

E. Tulving. How many memory systems are there? *American psychologist*, 40(4):385, 1985.

E. Tulving. Study of memory - processes and systems. In J. Foster and M. Jelicic, editors, *Memory: Systems, Process, or Function?*, pages 11–30. Oxford University Press, 1998.

E. Tulving. Concepts of memory. In E. Tulving and F. Craik, editors, *The Oxford Handbook of Memory*, pages 33–43. Oxford University Press, 2000.

E. Tulving. Episodic memory and common sense: how far apart? *Philosophical Transactions of the Royal Society of London Series B: Biological Sciences*, 356(1413):1505–1515, 2001.

E. Tulving. Are there 256 kinds of memory? In J. Nairne, editor, *The Foundations of Remembering: Essays in Honor of Henry L. Roediger, III*, volume 3, pages 39–52. Psychology Press, 2007.

E. Tulving. 32 episodic memory. page 152. Cambridge University Press, 2016.

E. Tulving and M. Lepage. Where in the brain is awareness of one's past? In D. Schacter and E. Scarry, editors, *Memory, Brain, and Belief*, pages 208–228. Harvard University Press, 2000.

E. Tulving and K. Szpunar. Episodic memory, 2009.

E. Tyugu and Ė. K. Tyugu. *Algorithms and architectures of artificial intelligence*, volume 159. IOS Press, 2007.

R. R. Vallacher and D. M. Wegner. What do people think they're doing? action identification and human behavior. *Psycological Review*, 94(1):3–15, 1987.

M. M. van Pinxteren, R. W. Wetzels, J. Rüger, M. Pluymaekers, and M. Wetzels. Trust in humanoid robots: implications for services marketing. *Journal of Services Marketing*, 2019. doi: 10.1108/JSM-01-2018-0045.

J. C. Vasquez Tieck, H. Donat, J. Kaiser, I. Peric, S. Ulbrich, A. Roennau, M. Zöllner, and R. Dillmann. Towards grasping with spiking neural networks for anthropomorphic robot hands. In A. Lintas, S. Rovetta, P. F. Verschure, and A. E. Villa, editors, *Artificial Neural Networks and Machine Learning – ICANN*, pages 43–51, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68600-4.

D. Vernon. *Artificial cognitive systems: A primer*. MIT Press, 2014.

D. Vernon, C. von Hofsten, and L. Fadiga. Desiderata for developmental cognitive architectures. *Biologically Inspired Cognitive Architectures*, 18:116–127, 2016. ISSN 2212-683X. doi: http://dx.doi.org/10.1016/j.bica.2016.10.004. URL `http://www.sciencedirect.com/science/article/pii/S2212683X16300822`.

R. Vidal, J. Bruna, R. Giryes, and S. Soatto. Mathematics of deep learning. *ArXiv e-prints*, December 2017.

G. Vigliocco, D. P. Vinson, M. F. Damian, and W. Levelt. Semantic distance effects on object and action naming. *Cognition*, 85(3):B61–B69, 2002. ISSN 0010-0277. doi: https://doi.org/10.1016/S0010-0277(02)00107-5. URL `http://www.sciencedirect.com/science/article/pii/S0010027702001075`. read.

S. Vijayakumar, A. D'souza, T. Shibata, J. Conradt, and S. Schaal. Statistical learning for humanoid robots. *Autonomous Robots*, 12(1):55–69, 2002. ISSN 1573-7527. doi: 10.1023/A:1013258808932. URL `http://dx.doi.org/10.1023/A:1013258808932`.

D. P. Vinson and G. Vigliocco. A semantic analysis of grammatical class impairments: semantic representations of object nouns, action nouns and action verbs. *Journal of Neurolinguistics*, 15(3):317–351, 2002. ISSN 0911-6044. doi: https://doi.org/10.1016/S0911-6044(01)00037-9. URL `http://www.sciencedirect.com/science/article/pii/S0911604401000379`.

B. Walther-Franks, J. Smeddinck, P. Szmidt, A. Haidu, M. Beetz, and R. Malaka. Robots, pancakes, and computer games: Designing serious games for robot imitation learning. In *33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3623–3632, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702552. URL `http://doi.acm.org/10.1145/2702123.2702552`.

P. Wang. From nars to a thinking machine. In *Advance of Artificial General Intelligence*, 2006.

Y. Wang. Cognitive learning methodologies for brain-inspired cognitive robotics. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, 9(2):37–54, April 2015. ISSN 1557-3958. doi: 10.4018/IJCINI.2015040103. URL http://dx.doi.org/10.4018/IJCINI.2015040103.

A. Wierzbicka. The search for universal semantic primitives. *Thirty Years of Linguistic Evolution. Amsterdam/Philadelphia: John Benjamins*, pages 215–42, 1992.

A. Wierzbicka. Semantic primes: Fifty years later. *Russian Journal of Linguistics*, 25(2): 317–342, 2021.

G. E. Wimmer, Y. Liu, N. Vehar, T. E. J. Behrens, and R. J. Dolan. Episodic memory retrieval success is associated with rapid replay of episode content. *Nature Neuroscience*, 23(8): 1025–1033, 2020. ISSN 1546-1726. doi: 10.1038/s41593-020-0649-z.

J. Winkler, M. Tenorth, A. K. Bozcuoğlu, and M. Beetz. CRAMm – memories for robots performing everyday manipulation activities. *Advances in Cognitive Systems*, 3:47–66, 2014.

J. Winkler, A. K. Bozcuoğlu, M. Pomarlan, and M. Beetz. Task parametrization through multimodal analysis of robot experiences (extended abstract). In *International Conference on Autonomous Agents*, AAMAS '17, pages 1754–1756, 2017.

M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich. Fetch and freight: Standard platforms for service robot applications. In *Workshop on autonomous mobile service robots*, 2016.

T. Wisspeintner, T. van der Zant, L. Iocchi, and S. Schiffer. RoboCup@Home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies*, 10(3): 393–428, 2009.

T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase. Development of human support robot as the research platform of a domestic mobile manipulator. *ROBOMECH Journal*, 6(1):4, April 2019. ISSN 2197-4225. doi: 10.1186/s40648-019-0132-3. URL https://doi.org/10.1186/s40648-019-0132-3.

C. Yan, T. B. Christophel, C. Allefeld, and J.-D. Haynes. Categorical working memory codes in human visual cortex. *NeuroImage*, 274:120149, 2023. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.2023.120149. URL https://www.sciencedirect.com/science/article/pii/S1053811923003002.

Y. Yang, Y. Aloimonos, C. Fermuller, and E. Erdal Aksoy. Learning the semantics of manipulation action. *arXiv e-prints*, art. arXiv:1512.01525, December 2015a.

Y. Yang, Y. Li, C. Fermüller, and Y. Aloimonos. Robot learning manipulation action plans by "watching" unconstrained videos from the world wide web. In *29th Conference on Artificial Intelligence (AAAI) and 27th Innovative Applications of Artificial Intelligence Conference (IAAI)*, volume 5, pages 3686–3692. AI Access Foundation, 6 2015b.

S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, July 2002. ISSN 1558-2590. doi: 10.1109/MPRV.2002.1037720.

J. M. Zacks and K. M. Swallow. Event segmentation. *Current Directions in Psychological Science*, 16(2):80–84, 2007. doi: 10.1111/j.1467-8721.2007.00480.x. URL https://doi.org/10.1111/j.1467-8721.2007.00480.x.

J. M. Zacks, S. Kumar, R. A. Abrams, and R. Mehta. Using movement and intentions to understand human activity. *Cognition*, 112(2):201–216, 2009. ISSN 0010-0277. doi: https://doi.org/10.1016/j.cognition.2009.03.007. URL http://www.sciencedirect.com/science/article/pii/S0010027709000705.

N. Zhang, T. Qi, and Y. Zhao. Real-time learning and recognition of assembly activities based on virtual reality demonstration. *Sensors*, 21(18), 2021. ISSN 1424-8220. doi: 10.3390/s21186201. URL https://www.mdpi.com/1424-8220/21/18/6201.

X. Zhao, A. M. Naguib, and S. Lee. Octree segmentation based calling gesture recognition for elderly care robot. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '14, pages 1–8, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2644-5. doi: 10.1145/2557977.2558030. URL http://doi.acm.org/10.1145/2557977.2558030.

Z. Zhen, C. Qixin, C. Lo, and Z. Lei. A corba-based simulation and control framework for mobile robots. *Robotica*, 27(3):459–468, 2009. doi: 10.1017/S026357470800489X.

Q. Zhu, V. Perera, M. Wächter, T. Asfour, and M. Veloso. Autonomous narration of humanoid robot kitchen task experience. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 0–0, 2017.

M. Ziaeefard and R. Bergevin. Semantic human activity recognition: A literature review. *Pattern Recognition*, 48(8):2329–2345, 2015. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2015.03.006. URL http://www.sciencedirect.com/science/article/pii/S0031320315000953.

C. Zins. Conceptual approaches for defining data, information, and knowledge. *Journal of the American Society for Information Science and Technology*, 58(4):479–493, 2007. doi: 10.1002/asi.20508. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.20508.