Controlling a General Purpose Service Robot By Means Of a Cognitive Architecture

Jordi-Ysard Puigbo[1], Albert Pumarola[1], and Ricardo Tellez[2]

[1] Technical University of Catalonia
[2] Pal Robotics ricardo.tellez@pal-robotics.com

Abstract. In this paper, a humanoid service robot is equipped with a set of simple action skills including navigating, grasping, recognizing objects or people, among others. By using those skills the robot has to complete a voice command in natural language that encodes a complex task (defined as the concatenation of several of those basic skills). To decide which of those skills should be activated and in which sequence no traditional planner has been used. Instead, the SOAR cognitive architecture acts as the reasoner that selects the current action the robot must do, moving it towards the goal. We tested it on a human size humanoid robot Reem acting as a general purpose service robot. The architecture allows to include new goals by just adding new skills (without having to encode new plans).

1 Introduction

Service robotics is an emerging application area for human-centered technologies. Even if there are several specific applications for those robots, a general purpose robot control is still missing, specially in the field of humanoid service robots [1]. The idea behind this paper is to provide a control architecture that allows service robots to generate and execute their own plan to accomplish a goal. The goal should be decomposable into several steps, each step involving a one step skill implemented in the robot. Furthermore, we want a system that can openly be increased in goals by just adding new skills, without having to encode new plans.

Typical approaches to general control of service robots are mainly based on state machine technology, where all the steps required to accomplish the goal are specified and known by the robot before hand. In those controllers, the list of possible actions that the robot can do is exhaustively created, as well as all the steps required to achieve the goal. The problem with this approach is that everything has to be specified beforehand, preventing the robot to react to novel situations or new goals.

An alternative to state machines is the use of planners [2]. Planners decide at running time which is the best sequence of skills to be used in order to achieve the goal specified, usually based on probabilistic approaches. A different approach to planners is the use of cognitive architectures. Those are control systems that

try to mimic some of the processes of the brain in order to generate a decision [3][4][5][6][7][8].

There are several cognitive architectures available: SOAR [9], ACT-R [10, 11], CRAM [12], SS-RICS [5], [13]. From all of them, only CRAM has been designed with direct application to robotics in mind, having been applied to the generation of pan cakes by two service robots [14]. Recently SOAR has also been applied to simple tasks of navigation on a simple wheeled robot [15].

At time of creating this general purpose service robot, CRAM was only able to build plans defined beforehand, that is, CRAM is unable to solve unspecified (novel) situations. This limited the actions the robot could do to the ones that CRAM had already encoded in itself. Because of that, in our approach we have used the SOAR architecture to control a human sized humanoid robot Reem equipped with a set of predefined basic skills. SOAR selects the required skill for the current situation and goal, without having a predefined list of plans or situations.

The paper is structured as follows: in section 2 we describe the implemented architecture, in section 3, the robot platform used. Section 4 presents the results obtained and we end the paper with the conclusions.

2 Implementation

The system is divided into four main modules that are connected to each other as shown in the figure 1. First, the robot listens a vocal command and translates it to text using the automatic speech recognition system (ASR). Then, the semantic extractor divides the received text into grammatical structures and generates a goal with them. In the reasoner module, the goal is compiled and sent to the cognitive architecture (SOAR). All the actions generated by SOAR are translated into skill activations. The required skill is activated through the action nodes.

2.1 Automatic Speech Recognition

In order to allow natural voice communication the system incorporates a speech recognition system capable of processing the speech signal and returns it as text for subsequent semantic analysis. This admits a much natural way of Human-Robot Interaction (HRI). The ASR is the system that allows translation of voice commands into written sentences.

The ASR software used is based on the open source infrastructure Sphinx developed by Carnegie Mellon University [16]. We use a dictionary that contains 200 words which the robot understands. In case the robot receives a command with a non-known word the robot will not accept the command and is going to request for a new command.

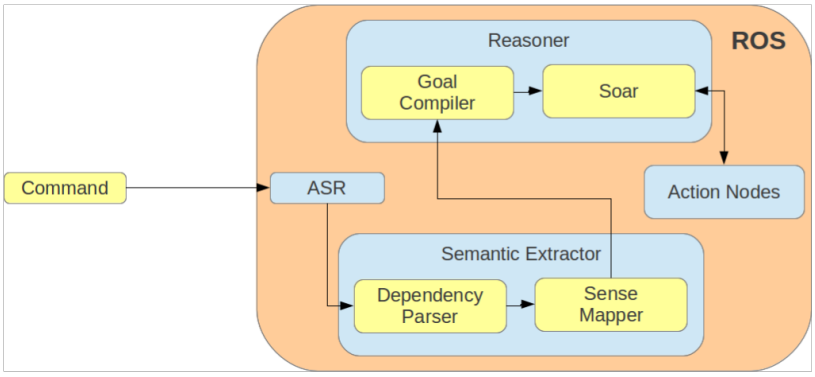

Fig. 1. Diagram of the system developed

2.2 Semantic Extractor

The semantic extractor is the system in charge of processing the imperative sentences received from the ASR, extracting and retrieving the relevant knowledge from it.

The robot can be commanded using two types of sentences:

Category I The command is composed by one or more short, simple and specific subcommands, each one referring to very concrete action.

Category II The command is under-specified and requires further information from the user. The command can have missing information or be composed of categories of words instead of specific objects (ex. *bring me a coke* or *bring me a drink*. First example does not include information about where the drink is. Second example does not explain which kind of drink the user is asking for).

The semantic extractor implemented is capable of extracting the subcommands contained on the command, if these actions are connected in a single sentence by conjunctions (*and*), transition particles (*then*) or punctuation marks. Should be noticed that, given that the output comes from an ASR software, all punctuation marks are omitted.

We know that a command is commonly represented by an imperative sentence. This denotes explicitly the desire of the speaker that the robot performs a certain action. This action is always represented by a verb. Although a verb may convey an occurrence or a state of being, as in *become* or *exist*, in the case of imperative sentences or commands the verb must be an action. Knowing this, we asume that any command will ask the robot to do something and these actions might be performed involving a certain object (*grasp a coke*), location (*navigate to the kitchen table*) or a person (*bring me a drink*). In *category I* commands, the semantic extractor should provide the specific robot action and the object, location or person that this action has to act upon. In *category II*, commands do

not contain all the necessary information to be executed. The semantic extractor must figure out which is the action, and identify which information is missing in order to accomplish it.

For semmantic extraction we constructed a parser using the Natural Language ToolKit (NLTK) [17]. A context-free grammar (CFG) was designed to perform the parsing. Other state-of-the-art parsers like Stanford Parser [18] or Malt Parser [19] were discarded for not having support for imperative sentences, having been trained with deviated data or needing to be trained beforehand. It analyses dependencies, prepositional relations, synonyms and, finally, co-references.

Using the CFG, the knowledge retrieved from each command by the parser is stored on a structure called parsed-command. It contains the following information:

- Which action is needed to perform
- Which location is relevant for the given action
- Which object is relevant for the given action
- Which person is relevant for the given action

The parsed-command is enough to define most goals for a service robot at home, like *grasp - coke* or *bring - me - coke*. For multiple goals (like in the category I sentences), an array of parsed-commands is generated, each one populated with its associated information.

The process works as follows: first the sentence received from the ASR is tokenized. Then, NLTK toolkit and Stanford Dependency Parser include already trained Part-Of-Speech (POS) tagging functions for English. Those functions complement all the previous tokens with tags that describe which is the POS more plausible for each word. By applying POS-tagging, the verbs are found. Then, the action field of the parsed-command is filled with the verb.

At this point the action or actions that are needed to eventually accomplish the command have been already extracted. Next step is to obtain their complements. To achieve this a combination of two methods is used:

1. Identifying from all the nouns in a sentence, which words are objects, persons or locations, using an ontology.
2. Finding the dependencies between the words in the sentence. Having a dependency tree allows identification of which parts of the sentence are connected to each other and, in that case, identify which connectors do they have. This means that finding a dependency tree (like for example, the Stanford Parser), allows to find which noun acts as a direct object of a verb. Additionally, looking for the direct object, allows us to find the item over which the action should be directed. The same happens with the indirect object or even locative adverbials.

Once finished this step, the full parsed-command is completed. This structure is sent to the next module, where it will be compiled into a goal interpretable by the reasoner.

2.3 Reasoner

Goal Compiler A compiler has been designed to produce the goal in a format understandable by SOAR from the received parsed-command, called the compiled-goal.

It may happen that the command lacks some of the relevant information to accomplish the goal (*category II*). This module is responsible for asking the questions required to complete this missing information. For example, in the command "*bring me a drink*", knowing that a *drink* is a category, the robot will ask for which drink is asking the speaker. Once the goals are compiled they are sent to SOAR module.

SOAR SOAR module is in charge of deciding which skills must be executed in order to achieve the compiled-goal. A loop inside SOAR selects the skill that will move Reem one step closer to the goal. Each time a skill is selected, a petition is sent to an action node to execute the corresponding action. Each time a skill is executed and finished, SOAR selects a new one. SOAR will keep selecting skills until the goal is accomplished.

The set of skills that the robot can activate are encoded as operators. This means that there is, for each possible action:

- A rule proposing the operator, with the corresponding name and attributes.
- A rule that sends the command through the output-link if the operator is accepted.
- One or several rules that depending on the command response, fire and generate the necessary changes in the world.

Given the nature of the SOAR architecture, all the proposals will be treated at the same time and will be compared in terms of preferences. If one is best than the others, this one is the only operator that will execute and a new deliberation phase will begin with all the new available data. It's important to know that all the rules that match the conditions are treated as if they fired at the same time, in parallel. There is no sequential order [20].

Once the goal or list of goals have been sent to SOAR the world representation is created. The world contains a list of robots, and a list of objects, persons and locations. Notice that, at least, there is always one robot represented, the one that has received the command, but, instead of just having one robot, one can generate a list of robots and because of the nature of the system they will perform as a team of physical agents to achieve the current goal.

SOAR requires an updated world state, in order to make the next decision. The state is updated after each skill execution, in order to reflect the robot interactions with the world. The world could be changed by the robot itself or other existing agents. Changes in the world made by the robot actions directly reflect the result of the skill execution in the robot world view. Changes in the world made by other agents, may make the robot fail the execution of the current skill, provoking the execution of another skill that tries to solve the impasse (for

example, going to the place where the coke is and finding that the coke is not there any more, will trigger the *search for object* skill to figure out where the coke is).

This means that after the action resolves, it returns to SOAR an object describing the success/failure of the action and the relevant changes it provoked. This information is used to change the current knowledge of the robot. For instance, if the robot detected a beer bottle and its next skill is to grasp it, it will send the command 'grasp.item = beer bottle', while the action response after resolving should only be a 'succeeded' or 'aborted' message that is interpreted in SOAR as 'robot.object = beer bottle'.

In the current state of the system 10 different skills are implemented. The amount of productions checked in every loop step is of 77 rules.

It may happen that there is no plan for achieving the goal. In those situations SOAR implements several mechanisms to solve them:

- Subgoal capacity [21], allows the robot to find a way to get out of an impasse with the current actions available in order to achieve the desired state. This would be the case in which the robot could not decide the best action in the current situation with the available knowledge because there is no distinctive preference.
- Chunking ability [21][22][23], allows the production of new rules that help the robot adapt to new situations and, given a small set of primitive actions, execute full featured and specific goals never faced before.
- Reinforcement learning [24], together with the two previous features, helps the robot in learning to perform maintained goals such as keeping a room clean or learning by the use of user-defined heuristics in order to achieve, not only good results like using chunking, but near-optimal performances.

The two first mechanisms were activated for our approach. Use of the reinforcement learning will be analysed in future works. Those two mechanisms are specially important because thanks to them, the robot is capable of finding its own way to achieve any goal achievable with the current skills of the robot. Also, chunking makes decisions easier when the robot faces similar situations early experienced. This strengths allow the robot to adapt to new goals and situations without further programming than defining a goal or admit the expansion of its capabilities by simply defining a new skill.

2.4 Action Nodes

The action nodes are ROS software modules. They are modular pieces of software implemented to make the robot capable of performing each one of its abilities, defined in the SOAR module as the possible skill. Every time that SOAR proposes an skill to be performed calls the action node in charge of that skill.

When an action node is executed it provides some feedback to SOAR about its succees or failure. The feedback is captured by the interface and sent to SOAR in order to update the current state of the world.

The set of skills implemented and their associated actions are described in table 1

Skill	**Action**
Introduce himself	Talks about himself
Follow person	Follows a specific person in font of him
Search objects	Looks for objects in front of him
Search person	Looks for some person in the area
Grasp object	Grasps an specific object
Deliver object	Delivers an object to the person or place in front
Memorize person	Learns a person's face and stores his name
Exit apartment	Looks for the nearest exit and exits the area
Recognize person	Checks if the person in front was already known and retrieves its name
Point at an object	Points the location of an specific object

Table 1. Table of skills available at the robot and their associated actions

3 Platform: Reem

The robot platform used for testing the system developed is called Reem 2, a humanoid service robot created by PAL Robotics. Its weight is about 90 Kg, 22 degrees of freedom and an autonomy of about 8 hours. Reem is controlled by OROCOS for real time operations and by ROS for skill depletion. Among other abilities, it can recognize and grasp objects, detect faces, follow a person and even clean a room of objects that do not belong to it. In order to include robust grasping and gesture detection, a kinnect sensor on a headset on her head has been added to the commercial version.

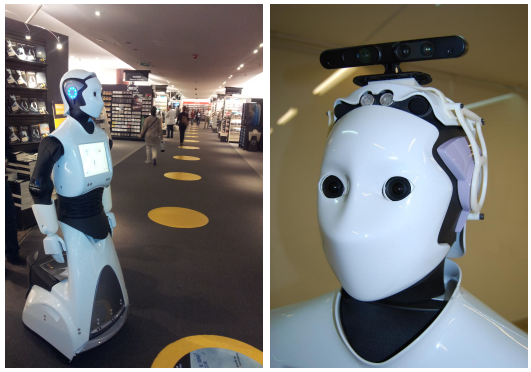

Fig. 2. (a) Reem humanoid robot and (b) Reem head with kinect included

The robot is equipped with a Core 2 Duo and an ATOM computer, which provide all the computational power required to perform all tasks control. This means that all the algorithms required to plan and perform all the abilities are executed inside the robot.

4 Results

The whole architecture has been put to test in an environment that mimics that of the RoboCup@Home League at the GPSR test [25] (see figure 3). In this test, the robot has to listen three different types of commands with increased difficulty, and execute the required actions (skills) to accomplish the command. For our implementation, only the two first categories have been tested, as described in section 2.2.

Fig. 3. Reem robot at the experiments environment that mimics a home

Testing involved providing the robot with a spoken command, and checking that the robot was able to perform the required actions to complete the goal.

Examples of sentences the robot has been tested with (among others):

Category I *Go to the kitchen, find a coke and grasp it*
Sequence of actions performed by the robot:
understand command, go to kitchen, look for coke, grasp coke
Go to reception, find a person and introduce yourself
Sequence of actions performed by the robot:
understand command, go to reception, look for person, go to person, introduce yourself
Find the closest person, introduce yourself and follow the person in front of you
Sequence of actions performed by the robot:
look for a person, move to person, introduce yourself, follow person

Category II *Point at a seating*
Sequence of actions performed by the robot:

understand command, ask questions, acknowledge all information, navigate to location, search for seating, point at seating
Carry a Snack to a table
Sequence of actions performed by the robot:
understand command, ask questions, acknowledge all information, navigate to location, search for snack, grasp snack, go to table, deliver snack
Bring me an energy drink (figure 4)
Sequence of actions performed by the robot:
understand command, ask questions, acknowledge all information, navigate to location, search for energy drink, grasp energy drink, return to origin, deliver energy drink

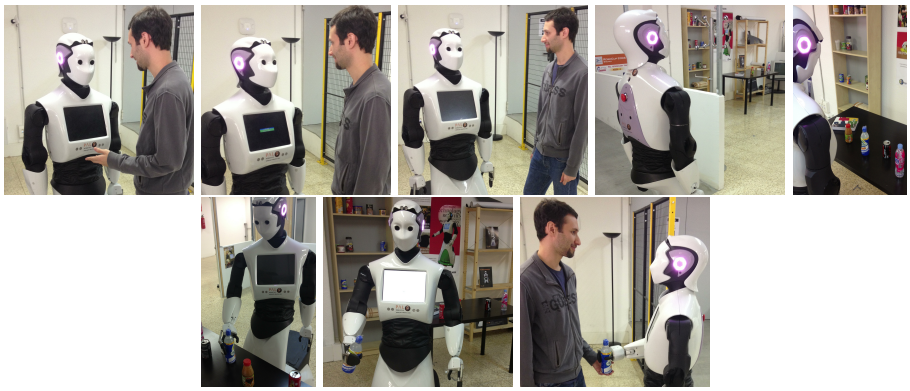

Fig. 4. Sequence of actions done by Reem to solve the command *Bring me an energy drink*

The system we present in this paper guarantees that the actions proposed will lead to the goal, so the robot will find a solution, although it can not be assured to be the optimal one. For instance, in some situations, the robot moved to a location that was not the correct one, before moving on a second action step to the correct one. However, the completion of the task is assured since the architecture will continue providing steps until the goal is accomplished.

5 Conclusions

The architecture presented allowed to command a commercial humanoid robot to perform a bunch of tasks as a combination of skills, without having to specify before hand how the skills have to be combined to solve the task. The whole approach avoids AI planning in the classical sense and uses instead a cognitive approach (SOAR) based on solving the current situation the robot faces. By solving the current situation skill by skill the robot finally achieves the goal (if

it is achievable). Given a goal and a set of skills, SOAR itself will generate the necessary steps to fulfil the goal using the skills (or at least try to reach the goal). Because of that, we can say that it can easily adapt to new goals effortlessly.

SOAR cannot detect if the goal requested to the robot is achiebable or not. If the goal is not achiebable, SOAR will keep trying to reach it, and send skill activations to the robot forever. In our implementation, the set of goals that one can ask the robot are restricted by the speech recognition system. Our system ensures that all the accepted vocal commands are achievable by a SOAR execution.

The whole architecture is completely robot agnostic, and can be adapted to any other robot provided that the skills are implemented and available to be called using the same interface. More than that, adding and removing skills becomes as simple as defining the conditions to work with them and their outcomes.

The current implementation can be improved in terms of robustness, solving two known issues:

First, if one of the actions is not completely achieved (for example, the robot is not able o reach a position in the space because it is occupied, or the robot cannot find an object that is in front of it), the skill activation will fail. However, in the current implementation the robot has no means to discover the reason of the failure. Hence the robot will detect that the state of the world has not changed, and hence select the same action (retry) towards the goal accomplishment. This behaviour could lead to an infinite loop of retries.

Second, this architecture is still not able to solve commands when errors in sentences are encountered (category III of the GPSR Robocup test). Future versions of the architecture will include this feature by including semantic and relation ontologies like Wordnet [26] and VerbNet [27], making this service robot more robust and general.

References

1. Haidegger, T., Barreto, M., Gonçalves, P., Habib, M.K., Ragavan, S.K.V., Li, H., Vaccarella, A., Perrone, R., Prestes, E.: Applied ontologies and standards for service robots. Robotics and Autonomous Systems (June 2013) 1–9
2. Stuart Russell, P.N.: Artificial Intelligence: A Modern Approach
3. Pollack, J.B.: Book Review : Allen Newell , Unified Theories of Cognition *
4. Jones, R.M.: An Introduction to Cognitive Architectures for Modeling and Simulation. (1987) (2004)
5. Kelley, T.D.: Developing a Psychologically Inspired Cognitive Architecture for Robotic Control : The Symbolic and Subsymbolic Robotic Intelligence Control System. Internation Journal of Advanced Robotic Systems **3**(3) (2006) 219–222
6. Langley, P., Laird, J.E., Rogers, S.: Cognitive architectures: Research issues and challenges. Cognitive Systems Research **10**(2) (June 2009) 141–160
7. Laird, J.E., Wray III, R.E.: Cognitive Architecture Requirements for Achieving AGI. In: Proceedings of the Third Conference on Artificial General Intelligence. (2010)

8. Chen, X., Ji, J., Jiang, J., Jin, G., Wang, F., Xie, J.: Developing High-level Cognitive Functions for Service Robots. AAMAS '10 Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems **1** (2010) 989–996
9. Laird, J.E., Kinkade, K.R., Mohan, S., Xu, J.Z.: Cognitive Robotics using the Soar Cognitive Architecture. In: Proc. of the 6th Int. Conf.on Cognitive Modelling. (2004) 226–230
10. Anderson, J.R.: ACT: A Simple Theory of Complex Cognition. American Psychologist (1995)
11. Stewart, T.C., West, R.L.: Deconstructing ACT-R. In: Proceedings of the Seventh International Conference on Cognitive Modeling. (2006)
12. Beetz, M., Lorenz, M., Tenorth, M.: CRAM A C ognitive R obot A bstract M achine for Everyday Manipulation in Human Environments. In: International Conference on Intelligent Robots and Systems (IROS). (2010)
13. Wei, C., Hindriks, K.V.: An Agent-Based Cognitive Robot Architecture. (2013) 54–71
14. Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mösenlechner, L., Pangercic, D., Rühr, T., Tenorth, M.: Robotic Roommates Making Pancakes. In: 11th IEEE-RAS International Conference on Humanoid Robots, Bled, Slovenia (October, 26–28 2011)
15. Hanford, S.D.: A Cognitive Robotic System Based on Soar. PhD thesis (2011)
16. Ravishankar, M.K.: Efficient algorithms for speech recognition. Technical report (1996)
17. Bird, S.: NLTK : The Natural Language Toolkit. In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. (2005) 1–4
18. Klein, D., Manning, C.D.: Accurate Unlexicalized Parsing. ACL '03 Proceedings of the 41st Annual Meeting on Association for Computational Linguistics **1** (2003) 423–430
19. Hall, J.: MaltParser An Architecture for Inductive Labeled Dependency Parsing. PhD thesis (2006)
20. Wintermute, S., Xu, J., Laird, J.E.: SORTS : A Human-Level Approach to Real-Time Strategy AI. (2007) 55–60
21. Laird, J.E., Newell, A., Rosenbloom, P.S.: SOAR: An Architecture for General Intelligence. Artificial Intelligence (1987)
22. Howes, A., Young, R.M.: The Role of Cognitive Architecture in Modelling the User : Soar s Learning Mechanism. (01222) (1996)
23. SoarTechnology: Soar : A Functional Approach to General Intelligence. Technical report (2002)
24. Nason, S., Laird, J.E.: Soar-RL : Integrating Reinforcement Learning with Soar. In: Cognitive Systems Research. (2004) 51–59
25. : Robocup@home rules and regulations
26. Miller, G.A.: WordNet: A Lexical Database for English. Communications of the ACM **38**(11) (1995) 39–41
27. Palmer, M., Kipper, K., Korhonen, A., Ryant, N.: Extensive Classifications of English verbs. In: Proceedings of the 12th EURALEX International Congress. (2006)