4th IFSA Winter Conference on Automation, Robotics & Communications for Industry 4.0 / 5.0 (ARCI' 2024), 7-9 February 2024, Innsbruck, Austria

Oral ☐ / Poster ☐ / The same ☐

Topic: *Human-Robot Interaction, Industrial Robots, Cognitive Approach for Robots*

Dialogue-Based Task Instructions and Modifications for Industrial Robots

M. Scheutz[1,2], **B. Oosterveld**[2], **J. Peterson**[2], **and E. Wyss**[2]
[1] Tufts University, 177 College Ave, Medford, MA 02155, USA
[2] Thinking Robots Inc., 12 Channel St., STE 202, Boston, MA 02210, USA
Email: matthias.scheutz@tufts.edu

Summary: Programming robots effectively remains a challenge for small businesses due to the high ongoing costs of robot programming experts. What is missing is a user-friendly software system such as a natural language-enabled cognitive assistant for developing robot programs that (1) does not require any particular training before it can be used, and (2) allows for natural instruction dialogues that let human operators develop programs interactively. In this paper, we introduce such as system, specifically the cognitive robotic TRACS architecture which enables industrial robots to learn from human teachers through natural language dialogues novel tasks that can be executed immediately. We briefly describe the core elements of the architecture and present a sorting task example to showcase how a task can be first instructed and later modified during task execution, all in multiple different spoken natural languages.

Keywords: natural language dialogue tasking, cognitive robotics, human-robot interaction, industrial applications

1. Introduction

A major challenge for a wide-ranging uptake of robotics technology, especially in small businesses, is the high entry cost of robots, not so much due to the robot hardware itself (which has become more affordable over the recent past), but due to the cost of software integration and, most importantly, the ongoing costs of programming robots for various tasks. A recent report emphasizes those challenges:

"SMEs, with production processes that run at a smaller scale, often find integration costs to be prohibitive or unjustifiable due to their smaller production lot sizes. This is because robots in the manufacturing context are currently primarily programmed to do specific tasks, and any change in the assembly process often requires reprogramming the robots or even an overhaul of the integrated manufacturing line." [1] In fact, hiring robot programmers is often not an option for SMEs due to the additional high costs of these experts.

What is still missing is user-friendly software such as natural language-enabled cognitive assistants for developing robot programs that (1) do not require any particular training before they can be used, and (2) allow for natural instruction dialogues that let human operators develop programs interactively, guided by the intelligent assistant. Such interactive task learning has been an important research direction for cognitive systems for a while (e.g., see the various articles in [2]) and is increasingly integrated into robotic systems, in particular, with the advent of large language models (e.g., see the recent attempts by Google and Microsoft, https://www.youtube.com/watch?v=NYd0QcZcS6Q).

In this paper, we introduce such a cognitive assistant that can remove the main hurdle of robot programming and system integration for end users, allowing skilled workers without any robot experience to instruct robots in natural language to perform any number of industrial manufacturing and assembly tasks that they can, furthermore, modify them later quickly if needed.

We will first briefly motivate the design of the system and then provide a brief overview of the system architecture, followed by a description of its operation. We will then demonstrate its utility for factory automation, especially for SMEs, using a simple sorting task that is fully instructed in English and later modified during task execution without. We conclude with a summary of potential applications and future developments of the system.

2. Motivation and Background

Classical cognitive architectures like Soar, ACT-R, and various others have been traditionally used for implementing general ways for artificial agents to learn new tasks for a variety of domains in an ideally domain-general manner (e.g., [3]). While classical cognitive architectures were originally intended to model human cognition and run only on computers without being connected to physical systems, various projects have over the years connected robotic sensors and actuators to inputs and outputs of cognitive architectures like Soar and ACT-R to demonstrate the utility of the hypothesized general cognitive capabilities such as reasoning, problem solving, planning, and communications. Most recently, with the advent of transformer models, in particular large language models (LLMs), new attempts are being made to replace the "cognitive parts" of cognitive architectures with transformer models that can take over communication and interaction with human interlocutors, and possibly other cognitive aspects such as reasoning and planning. In addition, deep neural networks are also used for perceptual processing as well as learning appropriate actions (e.g., in the context of deep reinforcement learning with DQNs). The challenge, however, with using such networks is that they often need massive amounts of data (and computational resources) for training, and even when they have learned to perform their intended tasks very well, they are still subject to their internal stochasticity that might produce unwanted outputs and behaviors, given that their operations is intrinsically one of sampling from probability distributions and not that of running clearly specified algorithms. As a result, any type of robot system that relies on transformers for its operation (be it for natural language understanding, planning, or action execution) runs the risk of exhibiting faulty behaviors and performing the wrong actions. We believe that while there could clearly be some utility for using LLMs in robotic systems (e.g., as a common sense repository to provide heuristics), they should not be used for controlling robots *per se*, e.g., for generating robot decisions and actions, certainly not in critical applications such as autonomous aircraft, surgery robots, or industrial robots where faults can cause irreparable harm to equipment and humans.

Instead, we believe that a robotic system should be able to provide guarantees about its behavior which is clearly the case for systems based on executing programs, i.e., algorithms with a clearly specified semantics. The challenge then is how to combine the predictability and repeatability of robot algorithms (as they traditionally run on robots when programmed by humans) with the flexibility of human natural language task instructions which might contain ambiguities and lack important details? We believe that it is possible to define a small subset of natural language (different from the very comprehensive language LLMs can handle) that is sufficient for instructing common tasks such as those typical for industrial manufacturing and assembly, and thus letting robots learn new tasks through interactive dialogues with their human instructors. Specifically, we believe that learned representations should be explicit and transparent *symbol structures* with clearly defined compositional meanings that can be used to verify any step in the learned task (either through a graphical interface that represents the instruction or through a verbal rendition in natural language). Then, once the details of a task have been verified by the human instructor, the robot's task representation can be approved by the human instructor and the robot is ready to execute the task at any time. Moreover, any such task can then immediately be shared with other, possibly heterogeneous robots (either through local networks or through the cloud) which can then also perform the task without any additional training. The critical part here is the explicit symbolic task representation that the robot will learn which makes its knowledge easily explainable to humans, different from the kinds of distributed numeric neural representations where it is impossible to say precisely what a particular numeric activation or weight vector represents, and where all explanations are essentially *posthoc* "interpretations" of the data (without any assurance that the hypothesized interpretation is correct).

In the following, we will introduce our proposed architecture framework that meets the above expectations for explicit explainable task representations and allows for execution guarantees about all learned behaviors in ways that current LLM-based models will never be able to provide.

3. The TRACS Architecture Framework

The Thinking Robots Autonomous Cognitive System (TRACS) architecture is the latest extension of our successful Distributed Integrated Affect Reflection Cognition (DIARC) cognitive architecture [4] which has been successfully demonstrated on a variety of autonomous mobile and stationary robots.

TRACS is an open, modular, distributed, cognitive robotic architecture that can operate mixed heterogeneous multi-robot and IoT systems and learn to symbolic task representations from natural language instructions. Learned tasks can be narrated or graphically displayed, used to provide assessments about expected performance, executed at any time, and modified on the fly. Different from LLMs and other neural architectures, task performance is formally guaranteed, i.e., the outcome of a program is provable and if failure probabilities are known for each action, the overall failure probability can be guaranteed (e.g., see [5] for an example of how the system can assess its own performance and determine the overall success and failure probabilities).

TRACS combines a set of unique capabilities that make it the perfect integration platform for many industrial robot applications (such as factory automarion), but also different types of service and social robots:

• deeply integrated natural language capabilities allowing for complex task instructions, including dialogues about expected task performance

• multi-modal interactions through speech, text, and GUIs based on context allowing for multiple physical and virtual representations of the system in different locations (e.g., on the factory floor or in a remote office)

• one-shot learning from natural language instructions, demonstrations, and observations enabling rapid task learning (e.g., based on [6]), but also rapid correction of learned knowledge and adaptation (e.g., [7]), either through human instruction (e.g., "if X is true, do Y, otherwise do Z" to clarify when to do Y and when to do Z) or through automated reasoning (e.g., analogical generalization)

• assured performance with guarantees through explicit pre-, operating, and post-conditions on all actions (e.g., constraints in linear temporal logic are incorporated into stochastic policies that minimize constraint violation, conditional assurance that "if the system detects X, it will react to it", etc.) and explanations

TRACS is implemented in the TRADE middleware which provides unprecedented introspection features, dynamic mixed cloud-based edge-based distributed configurations, and hybrid symbolic-subsymbolic machine learning methods with provable guarantees, providing the basis for software assurance. TRADE is based on the successful agent development environment (ADE) middleware (e.g., [8]) which to this day has several unique features compared to other open-source middleware like ROS (e.g., dynamic system-wide introspection and notification of components about available services).

real-time needs. This also allows for running redundant modules to address hardware and software failures.

TRACS has extensive integrated fault detection, fault exploration, and recovery methods that use system-wide as well component-based introspection, including the ability to automatically restart components on compute nodes on other available computers.

Fig. 1 shows an overview of TRACS where boxes indicate components with particular functions with connecting arrows indicating the information flow and data types exchanged between components. The "Goal and Skill Manager" manages the system goals, involving knowledge stored in the "STM/LTM Inference KB" component. It also includes an action execution component that robustly executes high-level plans produced by the planner/scheduler manager, on

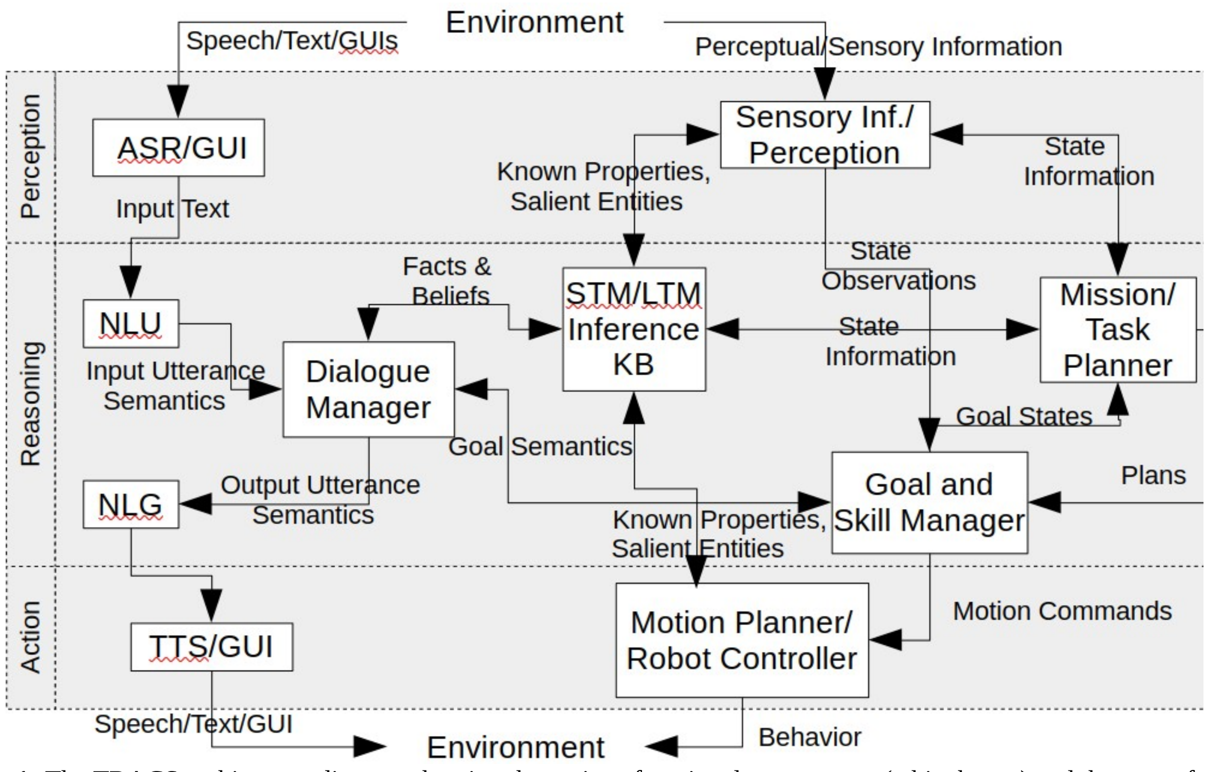

Fig. 1: The TRACS architecture diagram showing the various functional components (white boxes) and the type of data exchanged among them (labels on links), see text for details.

TRACS allows for easy integration with commercial off-the-shelf components and third-partymodules, components, and software libraries, through "component-wrapping" (e.g., dynamically loading libraries), "component participation" (e.g., third party components utilizing the same TRADE middleware connection mechanisms as other TRACS component), or "socket connections" (e.g., to standalone systems that are not available for wrapping or direct participation). Any of these methods can be performed dynamically during system operation.

TRACS systems are highly parallel and can be run within a single computational processes in a highly threaded fashion, or with components can be distributed over multiple OS processes across different operating systems and host computers to meet computational and

schedule, by coordinated commanding of multiple subsystems. The natural language subsystem in consists of the "ASR/GUI" (automated speech recognizer/GUI input), "NLU" (natural language understanding), "Dialogue manager", "NLG" (natural language generation), "Speech Synthesis/GUI" (speech or GUI output) corresponds to the "human-machine interactions manager".

4. Demonstration of TRACS in a Sorting Task

We demonstrate the operation of the architecture and the types of instruction dialogues it enables on an ABB GoFa robot in conjunction with a PLC controlled conveyor belt (a demo video of the task instructions for a GoFa robot in Robot studio can be found here: https://www.youtube.com/watch?v=xZaEk5pbVZk).

The demo scenario (shown in Fig. 2) is a sorting task in a conveyor belt setting where the GoFa robot has to pick up parts from the conveyor belt and place them on one of two tables depending on their length.

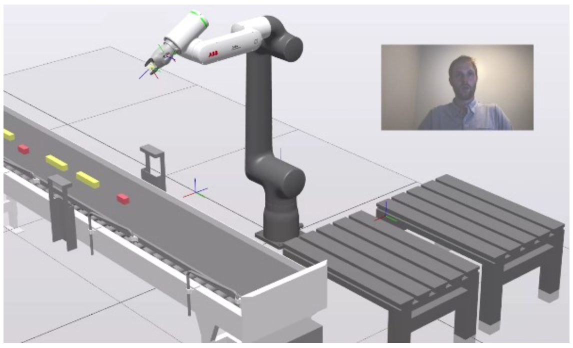
Fig. 2: The demo scenario showing a conveyor belt with various colored items of different length that need to be sorted onto two tables.

The teaching interacting for the sorting task then starts with the human instructor teaching the robot the various steps of the task:

Human instructor	*TRACS*
I will teach you how to sort.	OK
First, advance the conveyor belt.	OK
Then go to pose conveyor.	OK
Then verify that you can see a part.	OK
If the part is longer than 75mm then get the part on table A.	OK
Otherwise put the part on table B.	OK
That is how you sort.	OK
Start sorting.	OK

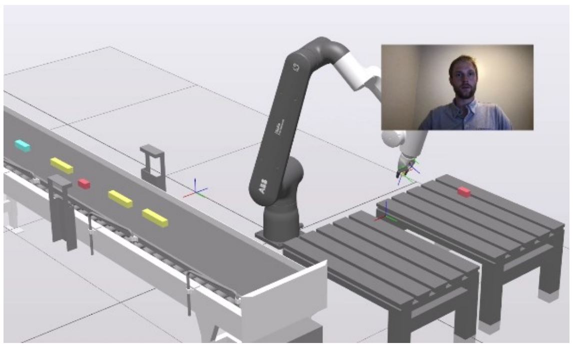
Fig. 3: The task state execution state when the human instructor modifies the task.

Fig. 3 shows the state of the task environment, in particular, the short red parts (less than 75mm in length) with one of them on Table B, the much longer yellow pars (over 100mm in length), and one blue part (80mm in length). The latter two are supposed to be placed on Table A.

While the robot is executing the sorting task, the instructor now modifies the task, specifically changing the sort criterion.

Human instructor	*TRACS*
Modify action sort.	OK, what will change?
Replace the part is longer than 75mm with the part is longer than 100mm.	OK, are there any other changes?
That is all.	

The change takes effect immediately and the robot starts placing only items longer than 100mm on table A as per the modified instruction.

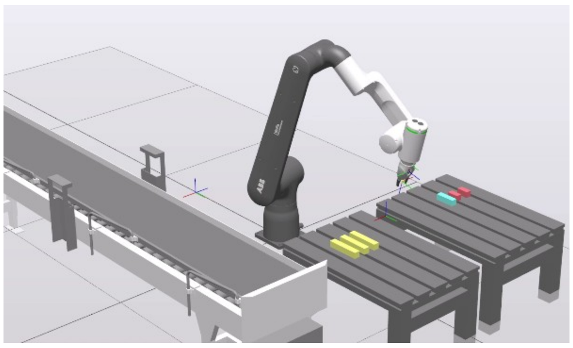
Fig. 4: The final state of the environment when task execution completes.

At the end when the task is completed, the yellow items longer than 100mm are placed on Table A, all other items are placed on Table B.

5. Discussion

The above demonstration example was chosen to be simply on purpose in order to illustrate the core aspects of teaching new tasks and modifying them on the fly. More complex task can be instructed that involve multiple robots, different types of objects and actions that need to be performed on those objects with outcomes that have to be observed or measured. For example, filling a container with parts from another container until a certain weight is reached might require the use of an IoT scale to measure the weight and a pouring action that allows the robot to pour items at a certain rate (by tipping and maybe shaking the container from which it is pouring). Assembly of a part might require the robot to put various pieces in place and screw in screws using an automated screwdriver mounted on the robot. It might also require multiple robots to operate on the same part at the same time (e.g., one robot holding the piece in place while the other attaches a part). Inspection tasks might require the robot to take precise measurements using additional tools and devices. All of these tasks can be instructed in TRACS as long as task instructions eventually ground out in primitive actions and perceptions the system understands (e.g., a "grasp" action where the robot can detect appropriate grasp points on an object it can perceive). Hence, before TRACS can be used for verbal task instructions, end users need to ensure that those required capabilities are in place (this includes any interactions with IoT or other devices for which special interface are needed).

6. Conclusions and Future Work

We provided a brief overview of the TRACS architecture and showed how it can be used to teach robots new tasks and quickly modify them on the fly. The presented types of interactions work on any number of robot arms (e.g., on Universal Robotics, Mitsubishi, etc.), mobile platforms, PLCs and other IoT devices as well as for any number of sorting, assembly, or other manufacturing tasks as long as interfaces to those other devices are in place and primitive actions using those devices are implemented and accessible through natural language.

In the future, we aim to provide additional learning capabilities that will enable robots to furthermore learn such primitive actions together with primitive perceptions without expert intervention as well. The idea is to integrate versions of reinforcement learning that can utilize simulated environments to quickly learn policies for primitive actions based on object perceptions that are co-learned.

References

[1] L Sanneman, C. Fourie, and J. Shah. The State of Industrial Robotics: Emerging Technologies, Challenges, and Key Research Directions. MIT Research Brief 15, November 2020.

[2] Kevin A. Gluck and John E. Laird (eds.) Interactive Task Learning: Humans, Robots, and Agents Acquiring New Tasks through Natural Interactions. MIT Press, 2019.

[3] P. Ye, T. Wang, and F-Y. Wan. A Survey of Cognitive Architectures in the Past 20 Years. *IEEE Transactions on Cybernetics,* Vol. *48,* No. *12,* December 2018.

[4] M. Scheutz, T. Williams, E. Krause, B. Oosterveld, V. Sarathy, and T. Frasca. An Overview of the Distributed Integrated Affect Reflection Cognition DIARC Architecture. In: Aldinhas Ferreira, M., Silva Sequeira, J., Ventura, R. (eds) *Cognitive Architectures. Intelligent Systems, Control and Automation: Science and Engineering*, vol 94. Springer, Cambridge, 2019.

[5] T. Frasca M. Scheutz. "Robot Self-Assessment of Expected Task Performance". *IEEE Robotics Automation Letters*. 2022.

[6] T. Frasca, B. Oosterveld, E. Krause, and M. Scheutz. One-Shot Interaction Learning from Natural Language Instruction and Demonstration, *Advances in Cognitive Systems*, 6, 159–176.

[7] T. Frasca, B. Oosterveld, M. Chita-Tegmark, and M. Scheutz. Enabling Fast Instruction-Based Modification of Learned Robot Skills. *Proceedings of AAAI*. 2021.

[8] M. Scheutz. ADE – Steps Towards a Distributed Development and Runtime Environment for Complex Robotic Agent Architectures. *Applied Artificial Intelligence*, 20, 4-5, 275–304. 2006.