Student Conference Proceedings
Vol 2, No 1, 2026, Article ID 2561, 4 Pages

Student Conference Lübeck

Proceedings Article

Adding a Cognitive Architecture to ROS2 for Interactive Task Learning in TurtleBots

Johannes Lopitz [a,*] · Nele Russwinkel [b,*]

[a] Study program Robotics and Autonomous Systems, Universität zu Lübeck, Lübeck, Germany
[b] Institute of Information Systems, Universität zu Lübeck, Lübeck, Germany
[*] Corresponding author, email: johannes.lopitz@student.uni-luebeck.de; nele.russwinkel@uni-luebeck.de

Received 24 March 2026; Accepted 24 August 2026; Published online 25 August 2026

Abstract

This work presents an empirical study of an integrated cognitive model that combines ROS2 actions and sensory data with a cognitive architecture pyACT-R for a turtlebot4 robot. The robot operates in a grid-based environment containing obstacles, while initially not knowing their properties (unknown obstacles). Path planning is performed using the A* algorithm, and encountered unknown obstacles are classified through physical interaction: non-traversable obstacles are classified as solid and avoided in subsequent path planning, while obstacles that turn out to be traversable despite prior evidence are classified as passable. Based on repeated collision experiences, the robot adapts its navigation strategy by switching from shortest-path strategy to a risk-aware one that avoids all obstacles. Experimental results demonstrate that the cognitive extension enables adaptive strategy selection and memory-based learning of obstacle properties. This highlights the potential of cognitive approaches for interactive task learning in mobile robots.

I. Introduction

Autonomous mobile robots are expected to operate in complex and partially unknown environments, where predefined maps and static assumptions are insufficient, for example in emergency scenarios with degraded visibility. Modern robotic middleware such as the Robot Operating System 2 (ROS2) provides a modular infrastructure for perception, communication, and navigation. However, standard ROS2 navigation pipelines primarily focus on geometric path planning and reactive obstacle avoidance, often assuming that environmental properties are known in advance or can be inferred from sensor data alone [1], [2], [3]. This limits the robot's ability to learn from interaction and adapt its behavior in ambiguous environments.

Classical navigation approaches treat such situations as reactive events without integrating experience into a persistent internal representation, causing robots to repeat the same mistakes, reducing robustness [1], [2].

Cognitive architectures address this limitation by providing structured mechanisms for memory, learning, and decision-making [4]. Unlike subsymbolic approaches, they explicitly represent past experiences and use this knowledge to guide future behavior [5]. ACT-R is a cognitive architecture that models cognition through declarative memory, procedural rules, perceptual-motor processes and others, making it suitable for learning from interaction in sequential tasks [6].

In this work we present and evaluate an interface that augments a ROS2-based navigation system with a cognitive layer implemented using pyACT-R, in order to enable interactive task learning in Turtlebot4 robots [7], [8]. The robot incrementally constructs an internal representation of its environment by physically interacting with obstacles; we evaluate this interface using a concrete navigation scenario, investigating whether it enables interactive task learning in dynamical and partially unknown

DOI 10.18416/SCP.2026.2561

environments, rather than aiming to outperform established, geometry-based navigation stacks.

II. Methods and Materials

II.I. System Architecture

The proposed system consists of a layered architecture combining low-level robot control and a second cognitive decision-making layer, connected via an adapter layer [4]. ROS2 is responsible for sensor handling, motion execution, and communication, while the cognitive layer is implemented by using pyACT-R and its extension[9]. On the ROS2 side, a single controller node manages the TurtleBot4 robot: it subscribes to the hazard-detection topic (bumper triggers) and to odometry, and holds two action clients (`DriveDistance`, `RotateAngle`) that issue the actual motion commands to the robot's onboard driver. All low-level functionalities, including bumper sensing, odometry, and velocity control, are implemented as a ROS2 node.

On the cognitive side, pyACT-R's declarative and procedural memory are extended with dedicated chunk types for the agent's position, for path/obstacle state, and for individual obstacle observations (position and status) [9]. Because computations such as A* search, grid-based bookkeeping of obstacle status, and bump-driven memory updates cannot be expressed directly as ACT-R productions, they are implemented in a separate adapter class that intercepts each ACT-R production as it fires and performs the corresponding computation before writing the result back into the goal and imaginal buffers. For example, when a *pathfinding* production fires, the adapter runs an A* search over the current grid representation, blocking only confirmed-solid cells in the shortest-path branch, and additionally blocking all not-yet-confirmed cells in the risk-aware branch, and stores the resulting path; when a *moving* production fires, this is translated into a corresponding call on the ROS2 controller node, which sends a `DriveDistance` or `RotateAngle` action goal to move the physical robot by a fixed step. Conversely, a bumper hazard detected by the ROS2 node is reported back to the adapter, incrementing an internal bump counter and setting a bump flag that the following *evaluation*-phase production reads to decide whether the corresponding grid cell is reclassified as solid or passable.

The environment is represented as a discrete two-dimensional grid-based map (see Fig. 1). Cells can be either free or obstructed, but at the beginning of each run all obstacles are treated as *unknown*. Through physical interaction, an obstacle is reclassified as *solid* (non-traversable, confirmed by collision, avoided in future planning) or *passable* (appeared occupied but was traversed without collision, e.g. due to prior misclassification or a removed object, and is therefore excluded from future avoidance). This unknown/solid/passable classification is used consistently throughout the paper.

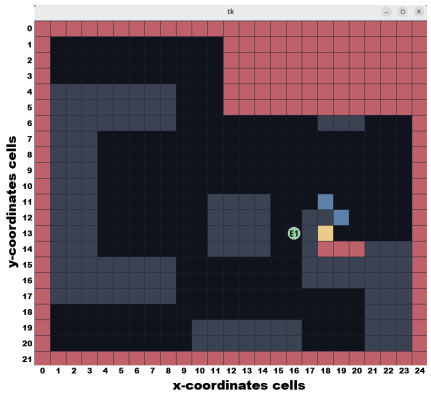

Figure 1: Discrete two-dimensional grid-based map from the parallel running simulation. Each colored square represents an obstacle (red: solid, grey: unknown, blue: passable, yellow: target). The circle represents the robot

The cognitive layer consists of multiple states and phases, as can be seen in Fig. 2: after an initial *locate* phase, the model alternates between *pathfinding*, *moving*, *evaluation* and *retrieval* until the *goal* state is reached. *Retrieval* is entered twice per obstacle cell: once while the planned path is checked, where it queries declarative memory for the cell's stored status (unknown, solid, or passable) and reacts accordingly (replanning around a confirmed-solid cell, skipping a confirmed-passable one, or proceeding to physically test an unknown one) and once after a movement attempt, where the outcome is written back as a new classification. *Evaluation* sits between these two retrievals: it reads a bump flag, set externally when the ROS2 controller reports a bumper hazard, and routes the model into the solid- or passable-retrieval branch accordingly. This memory persists across planning cycles, so the robot bases decisions on both current sensory input and accumulated experience.

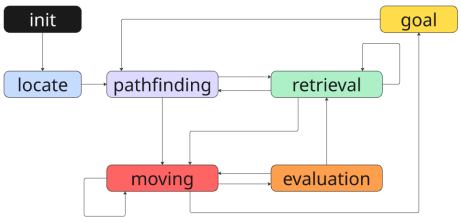

Figure 2: Cognitive model consisting of phases the robot will experience throughout its journey to the goal

II.II. Navigation and Path Planning

Navigation within the grid-based environment is performed using the A* algorithm, which computes the shortest path from the robot's current position to a specified goal cell. The path-planning operates on the discrete

DOI 10.18416/SCP.2026.2561

grid representation and uses a heuristic based on Manhattan distance to estimate the remaining cost to the target. At the start of each new run, only the start and goal positions are known, while all other cells are assumed to be traversable. The robot follows the planned path by executing a sequence of discrete motion commands, each corresponding to a fixed translation of 30 cm or a rotation of 90° or 180°. Movement commands are deliberately kept simple in order to maintain a clear mapping between planned actions and executed motions.

When the robot encounters an obstacle, a collision is detected through the bumper sensors. The robot estimates the previously executed translation and rotation based on odometry data and command history. Using this information, the robot performs a corrective maneuver to return to its last valid position before the collision occurred. This maneuver both recovers the robot from the navigation failure and, at the cognitive level, is interpreted as an interaction that provides information about the encountered obstacle.

II.III. Cognitive Strategy Adaptation

The core contribution of this work lies in the integration of a cognitive learning and strategy adaptation mechanism into the navigation pipeline. Obstacle evaluation results are processed by the pyACT-R model and stored in its declarative memory. Each obstacle is associated with a specific map location and classified based on the outcome of the interaction from the robot, as described in Section II.I.

Once an obstacle has been classified as *solid*, the path-planning is triggered to recompute a new path that avoids the corresponding grid cell. This ensures that previously encountered obstacles are not repeatedly tested, reducing unnecessary collisions and improving navigation efficiency. Obstacles that the robot can pass without a collision are classified as *passable* and stored accordingly in the declarative memory; these obstacles are subsequently ignored during planning, allowing the robot to exploit shortcuts.

Beyond obstacle classification, the cognitive model also monitors the number of collisions of the robot with the environment. After multiple collisions, the model initiates a strategy switch. For demonstration purposes the number of collisions was set to three. Instead of planning the shortest path, the path-planning enters a risk-aware mode in which all *unknown* obstacles are treated as *solid* in addition to obstacles already confirmed as solid from memory, i.e. the risk-aware strategy is strictly more conservative than the shortest-path strategy with respect to both previously known and not-yet-encountered cells, rather than being limited only to already-discovered grid space. This conservative strategy prioritizes safety and reliability over optimality, negating further collisions.

Once the robot reaches the goal location, it returns to the starting position using the fastest available path and initiates a new navigation attempt using the shortest-path strategy. This deliberate alternation between conservative and exploratory behavior enables the robot to refine its internal map and to identify potential shortcuts that were initially excluded. Concretely, task performance improves over repeated executions because each additional run adds newly confirmed *solid* and *passable* classifications to declarative memory, which shortens the effective search space for A* and reduces the number of cells that must be tested through physical collision.

III. Results and Discussion

The experimental results demonstrate that the integration of a cognitive architecture into the ROS2 navigation framework enables adaptive and experience-driven behavior in environments with uncertain obstacle properties. The initial path-planning is based on incomplete environmental knowledge. As a consequence, these paths frequently led the robot into contact with solid obstacles that had not yet been identified (see Fig. 3).

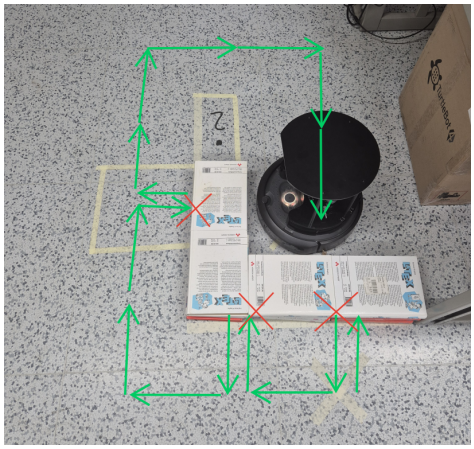

Figure 3: Real world experiment: First Path from Start to goal, with strategy switch after 3 bumps

Through physical interaction with the environment, the robot was able to reliably classify obstacles based on collision feedback and remembered them. This memory-based update mechanism prevented repeated encounters with the same obstacles and significantly reduced collision events over time.

A key observation is the effect of the strategy switching mechanism on navigation performance. The switch led to no further collisions, confirming that the conservative strategy successfully increased navigation safety. However this led to longer paths and increased traversal time.

On the way back to the start the robot was able to exploit previously identified passable obstacles and discovered shorter routes that had been inaccessible during the initial conservative phase.

DOI 10.18416/SCP.2026.2561

The strategy adaptation mechanism illustrates how cognitive approaches can support interactive task learning by combining memory, feedback, and goal-directed behavior[8]. Rather than relying on static planning rules, the robot continuously adapts its navigation strategy based on accumulated experience, demonstrating a key advantage of cognitive integration over purely reactive navigation within the scope of this evaluation.

In addition, the experiments revealed a notable reality gap between the assumed discrete motion model described in Section II.II and the robot's actual physical behavior. In practice, wheel slippage, odometry drift, and actuator inaccuracies led to deviations from these idealized motions, causing a gradual divergence between the robot's true position and its internal grid-based representation.

As a result, the robot occasionally collided with obstacles that should not have been encountered according to the symbolic map. Consequently, some interactions were misinterpreted as obstacle properties rather than localization errors, as the current cognitive framework does not explicitly represent pose uncertainty.

Future work should scale the environment representation from discrete grid maps to continuous spaces, integrate additional sensory modalities, address the reality gap through tighter integration of localization uncertainty, and extend the cognitive model to distinguish static from dynamic obstacles (e.g. humans) for safe real-world deployment.

IV. Conclusion

The experiments demonstrate that the robot can incrementally learn the properties of obstacles through physical interaction and store this knowledge in a persistent cognitive memory. This enables reliable classification of solid and passable obstacles and prevents repeated collisions. Furthermore, the implemented strategy adaptation mechanism allows the robot to dynamically balance efficiency and safety by switching between shortest-path and risk-aware navigation strategies. Through repeated task executions, the robot is able to exploit learned information to improve performance and discover more efficient routes.

The presented approach highlights the potential of cognitive architectures as a complementary layer to existing robotic middleware, particularly in environments where obstacle properties cannot be fully determined through perception alone and where interaction plays a central role in knowledge acquisition.

Acknowledgments

The work has been carried out and was supervised by Nele Russwinkel, Institute of Information Systems, Universität zu Lübeck. Furthermore we would like to thank Bastian Mannerow and Timon Dohnke for their support in this project.

Research funding: The author state no funding involved.

Author's statement

Conflict of interest: Authors state no conflict of interest. Informed consent: Informed consent has been obtained from all individuals included in this study. Ethical approval: The research related to human use complies with all the relevant national regulations, institutional policies and was performed in accordance with the tenets of the Helsinki Declaration, and has been approved by the authors' institutional review board or equivalent committee.

References

[1] A. Alexander, K. Venkatesan, J. Mounsef, and K. Ramanujam. A comprehensive survey of path planning algorithms for autonomous systems and mobile robots: Traditional and modern approaches. *IEEE Access*, 13:176287–176326, 2025, doi:10.1109/ACCESS.2025.3618837.

[2] S. M. Elbouhy, A. Adamanov, P. M. Braun, and H. Wilhelm Rose, Comparative analysis of local trajectory planning algorithms in ros2, in *2025 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 1–6, 2025. doi:10.1109/SIMPAR62925.2025.10979015.

[3] M. Badamasi Aremu, I. K. Kabir, G. Ahmed, and S. El-Ferik. Autonomous mobile robot path planning techniques—a review: Classical and heuristic techniques. *IEEE Access*, 13:117999–118022, 2025, doi:10.1109/ACCESS.2025.3579863.

[4] S. Kahl, S. Wiese, N. Russwinkel, and S. Kopp. Towards autonomous artificial agents with an active self: Modeling sense of control in situated action. *Cognitive Systems Research*, 72, 2021, doi:10.1016/j.cogsys.2021.11.005.

[5] Z. Zhang, N. Russwinkel, and S. Prezenski. Modeling individual strategies in dynamic decision-making with act-r: A task toward decision-making assistance in hci. *Procedia Computer Science*, 145:668–674, 2018, Postproceedings of the 9th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2018 (Ninth Annual Meeting of the BICA Society), held August 22-24, 2018 in Prague, Czech Republic. doi:https://doi.org/10.1016/j.procs.2018.11.064.

[6] 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, 2004.

[7] A. Brasoveanu and J. Dotlacil, The act-r cognitive architecture and its pyactr implementation, in 2020, 7–37, ISBN: 978-3-030-31844-4. doi:10.1007/978-3-030-31846-8_2.

[8] K. A. Gluck, J. Laird, and J. Lupp, [front matter], in *Interactive Task Learning: Humans, Robots, and Agents Acquiring New Tasks through Natural Interactions*, The MIT Press, 2019, ISBN: 9780262349420. doi:10.7551/mitpress/11956.003.0029.

[9] B. Mannerow. (2025). Act-r multi-agent simulation framework (python / pyactr). GitHub repository, URL: https://github.com/BastianMannerow/actr-multi-agent-simulation.

