Modular Approach for Controlling Multi-Agent Systems with Natural Language

!! DRAFT !! - DO NOT DISTRIBUTE

Christian Brazeau
High Performance Computing (RITB)
Air Force Research Laboratory, Information Directorate
Rome, NY, USA
christian.brazeau@us.af.mil

***Abstract*—The integration of natural language processing (NLP) and multiagent systems has the potential to significantly enhance human-robot collaboration. A modular pipeline is proposed that enables control of multiagent systems using natural language input, designed to be compatible with energy-efficient edge hardware and suitable for deployment on devices constrained by size, weight, and power (SWaP). This pipeline consists of three key components: (1) a large language model (LLM) to extract relevant information from human input, (2) a task controller to assign target states to individual agents, and (3) Cognitive Map Learners (CMLs) that enable agents to navigate towards their target states by learning representations of node states and edge actions in an arbitrary bidirectional graph. The pipeline's effectiveness is demonstrated in a simulated environment, where a swarm of ground robots successfully execute simple natural language commands, such as "form a circle around agent 1" or "form a square in the center". While the current implementation focuses on shape-object-preposition relationships, this work lays the foundation for future research on more complex natural language inputs and heterogeneous machine learning systems.**

***Index Terms*—cognitive map, modularization, robotics, natural language processing, large language models**

I. Introduction

The emergence of robotics and autonomous systems is fundamentally transforming the manner in which complicated tasks are executed, affecting a wide range of activities from search and rescue missions to industrial automation processes. As these systems become more advanced, the necessity for flawless interaction between humans and robots becomes increasingly acute. One of the central challenges in facilitating smooth interaction involves developing natural interfaces that overcome the communication divide between humans and robots and thereby facilitate easy and efficient interactions. Natural Language Processing (NLP) is another promising direction, enabling users to give instructions to machines in ordinary language.

Recent advances in NLP have given rise to large language models (LLMs) that can efficiently retrieve pertinent information from human language, ushering in the era of higher modalities of human-robot interaction. Concurrently, multiagent systems have exhibited impressive capabilities to accomplish complex tasks through the coordinated effort of numerous autonomous agents. Nevertheless, the marriage of NLP and multiagent systems encounters substantial technical challenges, especially in highly constrained environments where energy consumption and hardware space must be minimized.

The growing need for autonomous systems in resource-limited settings underscores the critical importance of size, weight, and power (SWaP) constraints in collaborative multiagent systems. As these systems find their way into diverse applications, spanning from unmanned aerial vehicles to ground-based robots, efficient and compact hardware designs have emerged as a primary concern. Moreover, cost considerations are significant, given that these systems are often intended for widespread deployment. Consequently, there is a pressing need for solutions that harmonize performance with SWaP and cost constraints, paving the way for broader adoption of autonomous systems across various domains. This necessitates innovative approaches to system design, hardware selection, and software optimization, all carefully orchestrated to ensure the effective operation of multiagent collaborative systems in real-world scenarios.

To address the aforementioned challenges, this paper introduces a modular natural language control pipeline for multiagent systems. The pipeline is carefully designed to be edge hardware energy efficient and suitable for devices with limited SWaP. One of the strengths of the modular system is that it is flexible to include new tasks and environments through adapting flexibly and efficiently without requiring full system retraining. Disaggregation of the system into independent, functionally specific modules enables individual module updating or replacement as needed without degrading the system's overall performance. Modular design enables rapid development and deployment of new functions with minimal reduction in the time and cost that would typically be required for large-scale system retraining. The efficiency of the pipeline is proved in a simulated setup, thus providing a foundation for further work on more advanced natural language inputs and heterogeneous machine learning systems. This further boosts the viability of human-robot collaboration in various applications.

II. Methods

The proposed pipeline consists of three primary components: a NLP module, a task controller, and an agent control module. The NLP module utilizes the LLaMA 3.1 [3] language model with 8B parameters to extract relevant information from user-provided natural language input. Specifically, the NLP module extracts the shape, preposition, and object from the input sentence and outputs a Python dictionary containing these extracted keywords to the task controller.

The task controller receives the output from the NLP module and assigns target states to individual agents in the form of fixed (x, y) positions. The task controller masks out the remaining components of the agent's observation vector, such as orientation and velocity, and assigns them their current values. This approach enables the agents to focus on reaching their target positions while maintaining their current velocity and orientation. For initial testing, the task controller implements simple fixed functions for assigning agent positions evenly spaced along the perimeter of the target shape at a fixed distance around the target object.

A. Agent Controller (CMLs)

The cognitive map learner (CML) is a system composed of three interconnected single-layer artificial neural networks (ANNs), collaboratively trained to encode and utilize bidirectional graphs for navigation tasks [1]. Within this framework, each node corresponds to a state, and each edge denotes an action permissible exclusively between two connected nodes. The bidirectional nature of the graph ensures that all actions are reversible. The CML is designed to learn three essential components: internal state representations for each node, the utility of each action for a given node, and the optimal sequence of actions to navigate the graph.

The internal state representations of nodes are encoded in the matrix $\mathbf{W_q} \in \mathbb{R}^{d \times n}$, where n is the size of the agent's observation vector, and d is the dimensionality of the state representation. The utility values of actions are encoded in $\mathbf{W_v} \in \mathbb{R}^{d \times e}$, where e represents the size of the action space. During initialization, $\mathbf{W_q}$ and $\mathbf{W_v}$ are randomly drawn from Gaussian distributions with means of zero and standard deviations of 0.1 and 1.0, respectively. The dimensionality d of the state representations is a user-defined parameter, commonly set to 512 or greater.

In this system, the observation vector $\mathbf{o_t}$ is derived from an agent's observations in a simulated environment, representing the agent's current perception of its state. These observations are encoded in $\mathbf{o_t}$, which is a one-hot vector mapping the agent's perceived state to a specific node in the graph. The CML computes the internal state representation of the current node using

$$s_t = W_q o_t. \tag{1}$$

Since o_t is a one-hot encoded vector, s_t directly corresponds to the column of W_q associated with the observed state.

Actions are represented as one-hot vectors a_t, where each discrete action available to the agent corresponds to a unique index in the action space. These actions are derived from the agent's decision-making process and define the edges in the bidirectional graph. Given a selected action a_t, the CML predicts the next state representation as

$$\hat{s}_{t+1} = s_t + W_v a_t. \tag{2}$$

Unlike previous implementations, training W_k, which encodes the set of permissible actions at each state, has been omitted in this work. Instead, the gating vector g_t, which identifies permissible actions, is computed directly from the agent's lidar sensor. This approach leverages the agent's local observations to dynamically determine allowable actions, simplifying the learning process and improving adaptability to the environment.

Learning occurs through updates to W_q and W_v using a delta learning rule. These updates are computed based on discrepancies between predicted and actual values, weighted by the inputs' transposes. The learning rules are defined as follows:

$$\Delta W_v(t) = \lambda_v (s_{t+1} - \hat{s}_{t+1}) a_t^\top, \tag{3}$$

$$\Delta W_q(t) = \lambda_q (\hat{s}_{t+1} - s_{t+1}) o_{t+1}^\top, \tag{4}$$

where λ_v and λ_q are learning rates, typically set to 0.1. Weight updates are aggregated at the end of each training epoch. Regularization is applied to normalize all vectors, ensuring that W_v is normalized along the e-axis.

Navigation within the graph begins with an agent's observation o_t, which initializes the current state representation $s_t = W_q o_t$. A target observation o^* is chosen, corresponding to the target state $s^* = W_q o^*$. The utility of each action is computed as

$$u_t = W_v^\top (s^* - s_t), \tag{5}$$

and the gating vector g_t, derived from the lidar sensor, identifies the permissible actions. Action selection involves an elementwise multiplication of g_t and u_t, followed by a winner-take-all (WTA) mechanism to produce the one-hot action vector a_t. Using the selected action a_t, the next predicted state $\hat{s}_{t+1}$ is calculated iteratively. This process enables the CML to derive an optimal sequence of actions for reaching the target node from the initial state, even without explicit training for traversal tasks.

III. Experiments

The following experiments were designed to evaluate the performance and adaptability of the CML in a multi-agent robotics environment. The setup tested the system's ability to learn navigation tasks, handle diverse agent types, and respond to high-level commands in both autonomous and mixed-control scenarios. By leveraging a custom environment built on the Petting Zoo Python library, the experiments explored the interaction between agents with varying movement capabilities and the effectiveness of the CML in coordinating their actions. The results provide insights into the strengths and limitations of the approach, highlighting its potential for real-world multi-agent applications.

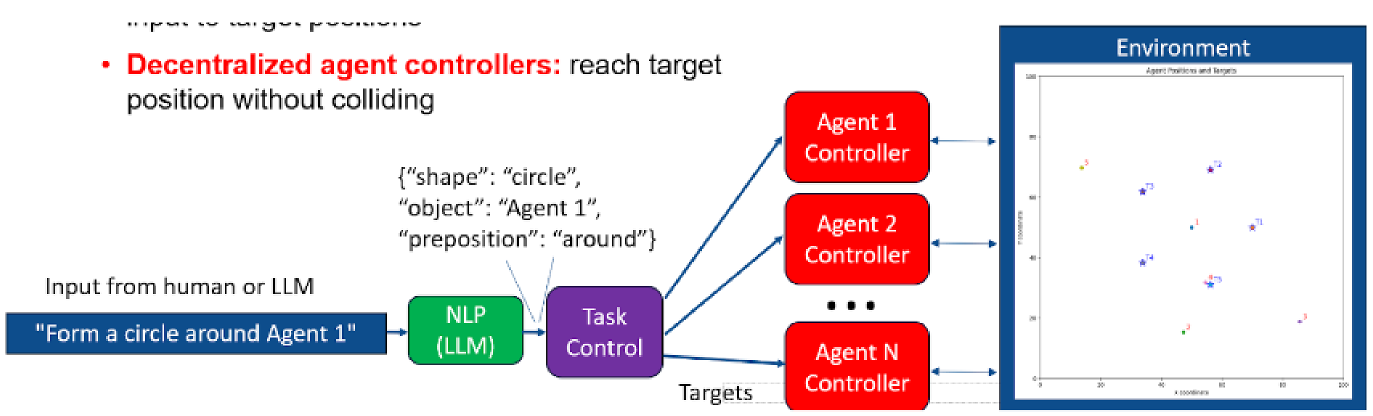
Decentralized agent controllers: reach target position without colliding
{"shape": "circle", "object": "Agent 1", "preposition": "around"}
Input from human or LLM
"Form a circle around Agent 1"
NLP (LLM)
Task Control
Agent 1 Controller
Agent 2 Controller
Agent N Controller
Targets
Environment

Fig. 1. System Block Diagram (TEMPORARY, tikz version for final submission)

A. Environment

This custom multi-agent robotics environment, built using the Petting Zoo Python library, provides a platform for testing various AI pipelines within a simulated setting. It follows the Agent Environment Cycle (AEC), where agents take turns observing the environment and executing actions. The environment features three distinct agent types with unique movement capabilities: holonomic agents with full freedom of motion, tank agents limited to forward/backward movement and rotation, and bicycle agents that adhere to bicycle kinematics with acceleration dynamics [I]. This variety in agent types allows for the exploration of complex multi-agent scenarios and coordination strategies.

B. Training Agent Controllers

Training was conducted with multiple agents simultaneously, all sharing a single replay buffer. This approach exposed the CML algorithm to a diverse set of observations more quickly during training, allowing for accelerated learning of navigation tasks. Agents performed random actions in the environment, storing observation-action-next observation pairs in the replay buffer. During training, minibatches of these experiences were sampled from the replay buffer and used to update the CML using the delta rules outlined in the methods section. The paths followed by agents during this phase were not optimal, nor did the agents need to collaborate to navigate them. Importantly, lidar-based gating was excluded during training, ensuring that learning was driven solely by the graph representation without additional sensory constraints.

While training the CML, the prediction error smoothly reduced to near zero (as shown in Figure 3), indicating that the model successfully captured the underlying dynamics of the environment. However, when measuring the mean distance of the agents from their targets in an evaluation cycle conducted every N minibatches, the performance was found to be highly unstable. This instability highlighted the challenges of generalizing from training data to actual navigation tasks. Incorporating an early stopping criterion significantly improved performance, yielding better results compared to training for a fixed number of timesteps. Early stopping allowed the model to halt training once the CML could sufficiently solve the navigation task before it deviated.

C. Full Pipeline

The experimental pipeline began with user input, processed through a NLP node that extracted relevant keywords. These keywords were passed to the task controller, which generated target states for each agent in the swarm. Each agent then independently attempted to navigate to its assigned target state within a fixed number of timesteps. This setup demonstrated the pipeline's ability to translate high-level user commands into actionable behavior for individual agents.

A player-controlled mode was also implemented to assess the swarm's robustness and adaptability. In this mode, the user controlled one agent directly while the remaining agents operated autonomously. The goal was to evaluate whether the swarm could maintain its formation and execute its tasks effectively despite dynamic and unpredictable changes introduced by the player-controlled agent. This mode provided valuable insights into the ability of the CML to handle mixed control schemes and real-time adjustments in a multi-agent system.

TABLE I
AGENT TYPES AND PROPERTIES

Agent Type	Observation Space	Action Space
Holonomic	Position (x, y) Velocity (v_x, v_y)	Up, Down, Left, Right
Tank	Position (x, y) Orientation (θ) Linear Velocity Angular Velocity	Forward, Backward, Rotate Left, Rotate Right
Bicycle	Position (x, y) Orientation (θ) Steering Angle Linear Velocity Angular Velocity	Forward, Backward, and 4 steering angles

IV. Results

The empirical analysis of the NLP module shows that it is fairly robust to different phrasing, spelling errors, and other input inconsistencies, demonstrating a very high success rate in extracting the intended keywords.

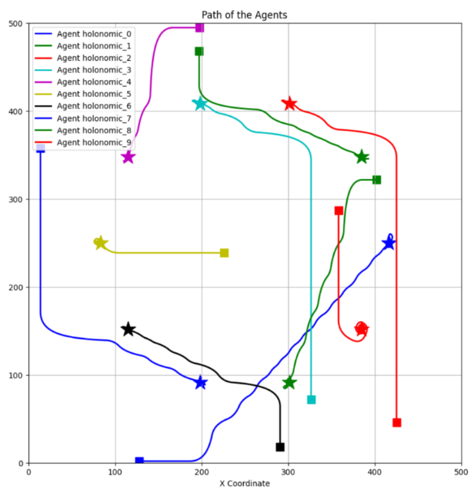

Fig. 2. Paths created during a single episode of the trained CML navigating the agents from their starting locations indicated by squares, to their target locations (stars). Instruction: "Form a circle"

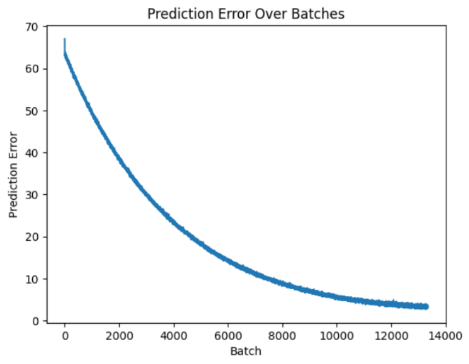

Fig. 3. CML prediction error smoothly reducing to near zero during training (no early stopping)

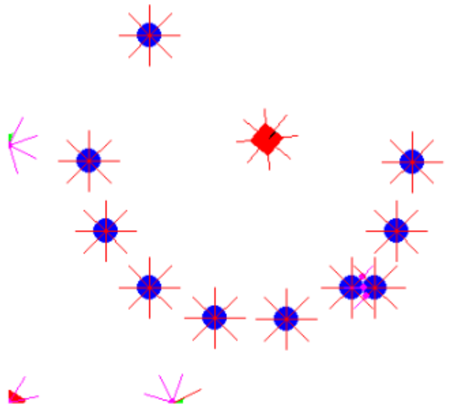

Fig. 4. Agents attempting to form a circle but some agents are not able to find find a path to their target locations

For the agent controllers, the holonomic agents performed efficiently without lidar gating, quickly converging to a solution that allows them to reliably reach their target positions. This is evidenced by the prediction error decreasing during training (figure 2). However, with lidar gating, agents occasionally get "stuck" and are unable to make progress toward their target states, as shown in an environment visualization where some agents remain stationary and in a figure plotting the paths of 10 agents from their starting to target locations during an episode.

The agent controllers for the tank and bicycle agents were unable to converge to a functional model despite a reduction in prediction error over time. The tank controller would frequently get stuck spinning in place, while the bicycle controller would oscillate back and forth, continuously adjusting its steering angle. Further research is required to determine the underlying reasons for these behaviors.

Finally, measuring the distance between agents and their targets at the end of each episode during training reveals that the training process is highly unstable. Early stopping is necessary to halt the CML training before it deviates significantly.

V. Discussion

This study presents a modular pipeline for controlling multiagent systems using natural language, highlighting a mix of successes and limitations that provide a clear direction for future development. While the system demonstrates promising capabilities in certain areas, it also faces challenges that need to be addressed to realize its full potential.

One of the system's key strengths is the robustness of the NLP module. It exhibits a high degree of accuracy in extracting relevant keywords from user input, effectively handling variations in phrasing, spelling errors, and other inconsistencies. This robustness is crucial for bridging the communication gap between humans and robots, enabling natural and intuitive interaction. However, the current implementation relies on a LLM, which may be computationally excessive for the relatively straightforward task of keyword extraction. Exploring alternative, lightweight approaches such as vector symbolic architectures (VSAs) or other methods that leverage hyperdimensional computing could reduce computational demands without compromising performance. This could prove particularly beneficial for deployment on resource-constrained edge devices.

The holonomic agent controllers also demonstrated promising results, particularly in scenarios without lidar gating. These controllers reliably converged to solutions that allowed agents to efficiently reach their target positions. This success highlights the effectiveness of the CML approach in enabling agents to learn representations of their environment and plan optimal paths. However, the performance of the holonomic agents degraded when lidar gating was introduced, with some agents getting "stuck" and unable to progress (figure (figure 4). This suggests that the integration of sensory information, such as lidar data, needs further refinement to ensure that it enhances rather than hinders navigation performance. Exploring methods for fusing lidar data with the CML's internal representations could lead to more robust and adaptable agent behavior.

In contrast, the agent controllers for the tank and bicycle agents faced significant challenges, failing to converge to functional models. The tank controllers frequently became stuck spinning in place, while the bicycle controllers exhibited oscillatory behavior, continuously adjusting their steering angle without stabilizing. These issues indicate that the current CML implementation may not be well-suited for agents with non-holonomic constraints. Further research is needed to understand the underlying reasons for these behaviors and to develop strategies for adapting the CML to accommodate the complexities of non-holonomic motion. This could involve exploring alternative state and action representations, modifying the learning algorithm, or incorporating techniques from control theory to better handle the dynamics of these agent types.

Another area of concern is the instability of CML training in dynamic environments, which necessitates early stopping to prevent significant performance degradation. This instability suggests that the current implementation of CML may not be sufficiently adaptable to changing conditions. Investigating the reasons behind this instability and exploring ways to enhance CML's adaptability and robustness will be crucial for future development. One potential avenue is to explore modular and hierarchical implementations of CML, which could allow for more localized learning and adaptation. Additionally, integrating hyperdimensional computing (HDC) principles, as suggested in recent literature [4], [5], may enhance the scalability and robustness of the CML in more complex environments.

The task controller also requires significant development to better handle dynamic and complex task environments. The current implementation relies on simple, fixed functions for assigning agent positions, which limits its flexibility and applicability to a wider range of scenarios. Developing a more sophisticated task controller capable of interpreting complex instructions, decomposing them into subtasks, and dynamically assigning roles to agents would greatly enhance the system's overall capabilities. This could involve incorporating techniques from hierarchical planning, task allocation, and multi-agent coordination to enable the system to handle more intricate and dynamic tasks.

VI. Conclusion

This work, while still a work in progress, provides valuable insights into the strengths and weaknesses of the current system. The results indicate several promising areas for continued research.

Future work should focus on modular and hierarchical implementations of CMLs, potentially integrating HDC as suggested in recent literature. These approaches may enhance scalability and robustness in more complex environments. Expanding the system to accommodate more intricate instructions, sequential tasks, and improved collaboration among agents is another key avenue for exploration. Efforts to remove hard-coded components will also be essential to achieve a more generalizable system architecture.

Addressing the limitations outlined in this study will require careful evaluation of alternative methods for task control, keyword extraction, and agent training. Comparative analysis with other deep reinforcement learning (Deep RL) and deep multi-agent reinforcement learning (Deep MARL) methods is also necessary to identify areas of improvement and validate the system's performance.

In conclusion, while this study demonstrates substantial progress in several aspects, significant challenges remain. Continued research and development will be essential to fully unlock the potential of this system and address its current limitations.

Acknowledgment

Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the US Government, the Department of Defense, or the Air Force Research Lab.

References

[1] C. Stöckl, Y. Yang, and W. Maass, "Local prediction-learning in high-dimensional spaces enables neural networks to plan", Nat Commun 15, 2344 (2024). https://doi.org/10.1038/s41467-024-46586-0

[2] I. Polykretis and A. Danielescu, "Mapless mobile robot navigation at the edge using self-supervised cognitive map learners," Frontiers in Robotics and AI, vol. 11, p. 1372375, 2024. https://doi.org/10.3389/frobt.2024.1372375

[3] A. Grattafiori et al., "The llama 3 herd of models," arXiv preprint arXiv:2407.21783, 2024.

[4] N. McDonald, "Modularizing and assembling cognitive map learners via hyperdimensional computing," arXiv preprint arXiv:2304.04734, 2023.

[5] N. McDonald and A. Dematteo, "Assembling modular, hierarchical cognitive map learners with hyperdimensional computing," arXiv preprint arXiv:2404.19051, 2024.