State Space Exploration with Large Language Models for Human-Robot Cooperation in Mechanical Assembly

Attique Bashir
Industrial Robotics and Human-Robot Cooperation
ZeMA gGmbH
Saarbruecken, Germany
a.bashir@zema.de

Raja Moktafi
Industrial Robotics and Human-Robot Cooperation
ZeMA gGmbH
Saarbruecken, Germany
r.moktafi@zema.de

Marco Giangreco
Assembly System Planning and Digitalization
ZeMA gGmbH
Saarbruecken, Germany
m.giangreco@zema.de

Rainer Mueller
Head of division assembly system
ZeMA gGmbH
Saarbruecken, Germany
rainer.mueller@zema.de

Abstract—**In the realm of human-robot cooperation (HRC) for mechanical assembly, determining the task allocation between human operators and robots is crucial. Traditionally, this requires extensive pre-planning of the product's state space. We propose a novel system utilizing a Large Language Model (LLM) to dynamically reason and determine the next assembly state based on real-time product description. A camera monitors the assembly process, and the captured data is pre-processed and fed to a LLM, which predicts the next assembly state and subsequently instructs the robot on its task. This approach enables a more flexible and adaptive assembly process without the need for exhaustive pre-planning.**

Keywords—Human-Robot Cooperation, state space exploration, mechanical assembly, large language Models, chain-of-thought, dynamic task planning

I. Introduction

Human-Robot Cooperation (HRC) is a semi-automated process [1, 2], with significant potential in mechanical assembly. This concept leverages the strengths of humans, such as sensitivity, flexibility, and adaptability to unforeseen circumstances, alongside the advantages of robots, including consistent accuracy, continuous process quality, and the ability to handle heavy weights without fatigue [3]. Effective HRC can reduce errors, alleviate ergonomic strain on human operators, and streamline the assembly process by enabling robots to assist with specific tasks, such as tool handover or part positioning [4]. However, current HRC applications often limit operator flexibility and do not fully exploit the system's potential due to technical constraints, raising questions about their efficacy [5].

Automated mechanical assembly processes often rely on predefined task plans, with each step meticulously planned by an engineer, considering every relevant or realistic step. In semi-automated such as HRC, the engineer must predefine the process sequence and allocate tasks to either the robot or the assembly operator. This rigid approach restricts flexibility and often fails to accommodate deviations from the plan, despite the potential for multiple viable assembly sequences [6]. Consequently, these constraints can lead to inefficiencies and operator dissatisfaction.

Efforts to introduce more flexibility in automated processes have included broad state space modeling of feasible assembly states. In these systems, sensors detect the current state and controller anticipates subsequent states [7]. Classical planning methods, based on mathematical logic, such as those using Planning Domain Definition Language (PDDL)[8], allow for state space exploration (SSE) by defining boundary conditions, system capabilities, and goals [9]. However, modeling a comprehensive state space is increasingly unmanageable as complexity grows, necessitating advanced programming skills or sophisticated simulation software.

We propose utilizing Large Language Models (LLMs) for state space exploration to enable reasoning and prediction of the next possible state.

In the following sections, we first introduce the basic use case of HRC. We then explain the composition and execution of the product domain within the LLMs. Subsequently, we discuss the integration of the LLM with object detection model You Only Look Once (YOLO) [10] into HRC system. Finally, we compare the results and provide an outlook on future developments.

II. Approach and Contribution

The basis for every assembly process whether automated or manual is the product and its assembly process. The planning engineer performs a product analysis and creates a physical assembly order. The product usually allows for multiple physical assembly sequences. Fig. 1 depicts a so called and/or-graph [11] as an example for the physical assembly possibilities of a product. The nodes of the graph represent a product state whereas the edges are interpreted as assembly processes.

We present a use case where an LLM performs state space exploration dynamically during the mutual assembly process between human and robot, rather than relying on pre-explored states, thus facilitating the integration of new assemblies on the same production resources. In this setup, a sensor system consisting of camera continuously captures the assembly progress and feeds the information to the LLM. The LLM directs the robot to perform the appropriate tasks. This method eliminates the need for advanced programming skills or simulation tools, enabling real-time state space exploration, which enhances flexibility.

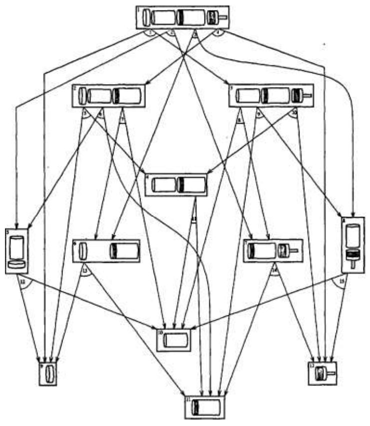

Fig. 1. And/Or-Graph as state space for mechanical assembly.

III. State of Art

A. State Space Exploration in the Context of Human-Robot Collaboration for Mechanical Assembly

State space representation is a fundamental concept in dynamical systems, encompassing all possible states of a system. In discrete dynamical systems, state spaces are often depicted as graphs where nodes represent specific states and edges indicate transitions between states [8]. In mechanical assembly, this translates to numerous possible combinations of assembled product parts, each constituting a unique state. As products increase in complexity, the state space grows, necessitating robust exploration techniques.

State Space Exploration (SSE) employs search algorithms to identify target states within a state space graph. Algorithms like Dijkstra's [12], depth-first [13], and breadth-first [14] traverse the graph to find target states, often returning a sequence of steps or instructions. While these algorithms operate in a brute-force manner, others, such as the A*-algorithm [15], use heuristics to expedite the search process [8]. Despite their efficiency, these algorithms require a fully defined state space.

Planning languages like PDDL and STRIPS [7] offer an alternative approach, enabling formalized problem definitions and rule-based state transitions [9]. These languages reduce the need for exhaustive state space pre-definition but still demand precise rule specification and programming expertise [9]. The complexity of state space modeling often necessitates advanced proficiency in programming languages or simulation tools.

To address these challenges, we can leverage LLMs to dynamically reason about the next assembly step, reducing the need for pre-defined state spaces and rigid programming structures. By describing the product and assembly process in natural language the LLM will determine the next step in the mechanical assembly process, which is then executed by the robot [10][11].

B. Current state of Large Language Models

Large Language Models (LLMs) are advanced foundation models trained on extensive textual data, enabling them to generate human-like text and respond to queries [16]. These models can be tailored for specific tasks through various customization methods, enhancing their utility in diverse applications. In general, LLMs generate text based on prompts, allowing them to continue a given input or provide contextually relevant responses [17].

In contrast to LLM, Large Vision Models (LVMs) are designed to process, understand and interpret visual data, such as images and videos, and are used in tasks like object recognition, classification, and segmentation. Notable examples of LVMs include Vision Transformers (ViT) [18] and ResNet [19]. Meanwhile, Large Multimodal Models (LMMs) are capable of processing and integrating multiple data modalities, such as text, audio, and images. For example, GPT-4 [20] and Gemini [21] are used for LMMs tasks like image captioning or visual question answering.

Customizing LLMs involves techniques such as fine-tuning on domain-specific data, reinforcement learning, and prompt engineering. Fine-tuning adjusts the model's parameters to better suit specific tasks, while reinforcement learning optimizes the model's performance based on feedback [16]. Prompt engineering involves designing effective input prompts to elicit desired responses from the LLM [17].

Additionally, there are several methods to enhance the output of LLMs by improving their reasoning capabilities through techniques such as chain-of-thought (CoT) prompting [22]. This technique involves breaking down complex problems into a series of intermediate steps that guide the model to reach the solution. Other methods, such as few-shot prompting, use a limited number of examples to enable the model to perform better. Combining CoT with zero-shot prompting can also be effective e.g. by simply adding “Let’s think step by step” before each answer [23].

The method used in this paper is a combination of CoT and few-shot prompting, indicating the model how to reason based on desired outputs.

IV. Use-Case Description

A. Description of the HRC Workstation

The cooperation between the assembly operator and the robot takes place at a one-person workstation, as illustrated in Fig. 2. It comprises a table on which a collaborative robot (Universal Robot UR10e) is mounted. The cobot is equipped with an adaptive two-finger gripper (ROBOTIQ 2F-140). The table is divided into two distinct areas. The first area, called bin area, is designated for storing the necessary assembly parts needed for the product. These parts are embedded in foam cushion shadow boards. The second area, referred to as the assembly area, is situated closer to the operator and it is where the assembly process takes place. A Red-Green-Blue-Depth (RGBD) camera (Microsoft Azure Kinect) is mounted above the workstation, capturing all necessary areas on the table. Although the camera can capture point clouds of components, this application currently requires only RGB information to perform object detection.

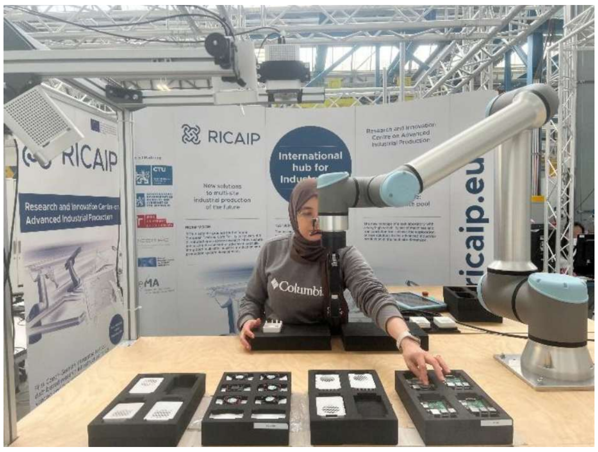
Fig. 2. Cooperative human-robot workstation.

B. Product Description

For mutual assembly, we consider two different products: the “Mini-Controller” and the “Sensor-Box”. The Mini-Controller, illustrated in Fig. 3, consists of five components: Bottom Case, Circuit Board, Fan, Top Case, and four Screws.

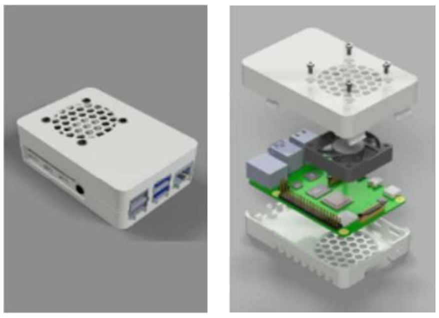
Fig. 3. Composition of the Mini-Controller product.

The product is assembled in a sequential layered manner, allowing only one feasible assembly sequence, as depicted in Fig. 4. The assembly process begins with the Bottom Case, on which the Circuit Board is placed. Next, the Fan is positioned above the Circuit Board, followed by the Top Case. Finally, the assembly is secured using four screws.

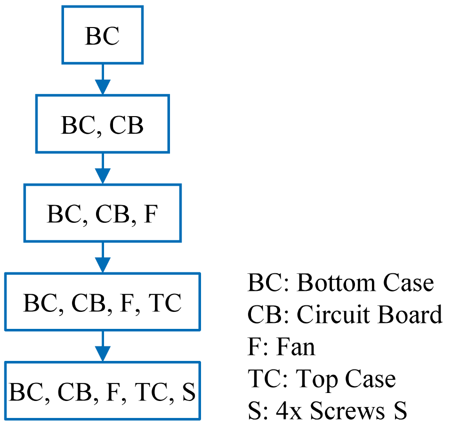

Fig. 4. State space of the Mini-Controller assembly.

The Sensor-Box, depicted in Fig. 5, is comparable in size to the Mini-Controller. It also consists of a Bottom Case and a Top Case, however, unlike the Mini-Controller, it includes two cubes (Cube 1, Cube 2) and three sets of screws (S1, S2, and S3) from different types. Cube 1 and Cube 2 are separately placed over the Bottom Case and secured successively by two screws of type S1 and S2. Then, the Top Case is fixed to the Bottom Case with four screws of type S3.

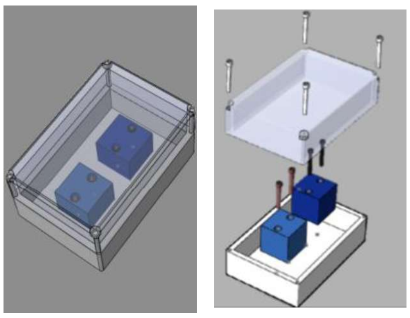
Fig. 5. Composition of the Sensor-Box.

In contrast to the first product, the mechanical assembly of the Sensor-Box allows for multiple assembly sequences and requires a total of seven steps to complete. This is due to the unconstrained placement of the two cubes. Fig. 6 illustrates the assembly state space of the Sensor-Box for a better understanding.

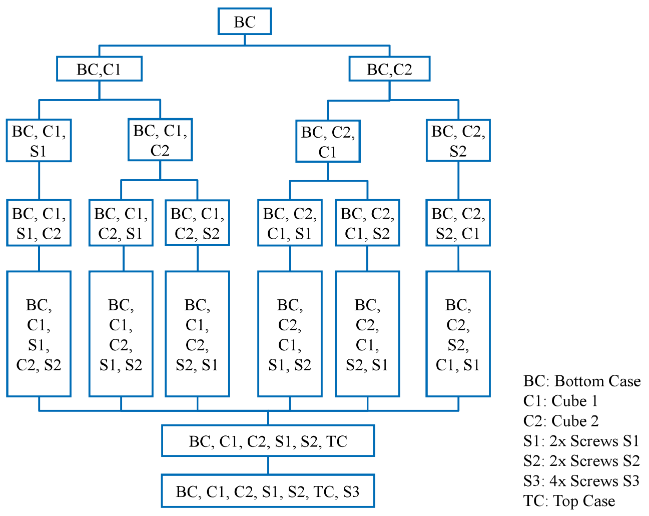

Fig. 6. State space for Sensor-Box assembly.

C. Software Architecture

Although the LLM is the centerpiece of this work, it constitutes only part of the architecture. The software is composed of three main modules: Object Detection, Action Planner and Robot Controller, as illustrated in Fig. 7. Communication and data are managed using Robot Operating System 1 (ROS 1) [24], enabling real-time, asynchronous, and reliable inter-process communication.

The Object Detection module utilizes Azure Kinect to stream RGBD frames, which are processed to identify objects and determine the bin and assembly area state. The Action Planner, powered by the LLM, receives the area state to generate robot actions in JSON format. These actions are then executed by the Robot Controller, which handles tasks such as trajectory path planning and gripper actuation. It also communicates the robot's state back to the Action Planner to ensure accurate replanning.

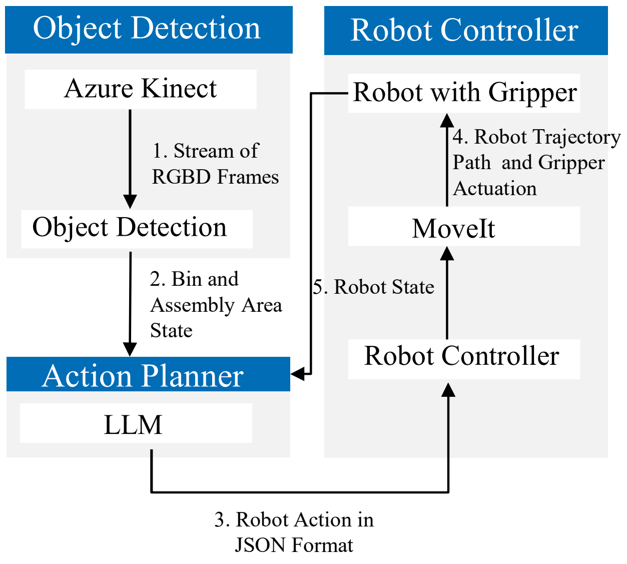

Fig. 7. Software architecture

D. *Object Detection Module*

The Object Detection Module does not utilize a multimodal LLM, rather it relies on real-time sensors to capture the environment. The Object Detection module employs a camera to retrieve RGB data. The camera continuously captures the workspace as depicted in fig. 7, and feeds the images through a ROS topic to the object detection module. This module is implemented using the Convolutional Neural Network (CNN) based object detection model 'YOLOv5' (You Only Look Once version 5). The process includes both object recognition and localization within the camera's coordinate system. YOLOv5 identifies objects by returning their class name, center-point and parameters such as height, width and orientation, which define the bounding box around each object. The YOLOv5 model is trained with custom data in a supervised learning manner. The custom data consists of 850 annotated high-resolution images (1920×1080), with 650 used for the training used for training, 100 for testing, and 100 for evaluating the trained model. The model has to identify the assembly parts on the table which for the first product means identifying: Bottom Case (bottom), Circuit Board (raspi), Fan (fan), Top Case (top) and Screws (screw).

To evaluate the performance of the trained YOLOv5 model, we use a common metric called mean Average Precision (mAP), calculated with an Intersection over Union (IoU) threshold of 0.5. Additionally, the average mAP is calculated using IoU thresholds ranging from 0.5 to 0.95 in 0.05 increments. For our trained model the mAP values range between 0.89 and 0.96. With these results, we can sufficiently detect the assembly parts and their location. Fig. 8 shows an example of the model's output, where a bounding box surrounds each identified assembly part, displaying the class name, confidence score and its orientation in degree.

Having identified the assembly items in the image as well as their position, this information is fed to the Action Planner Module.

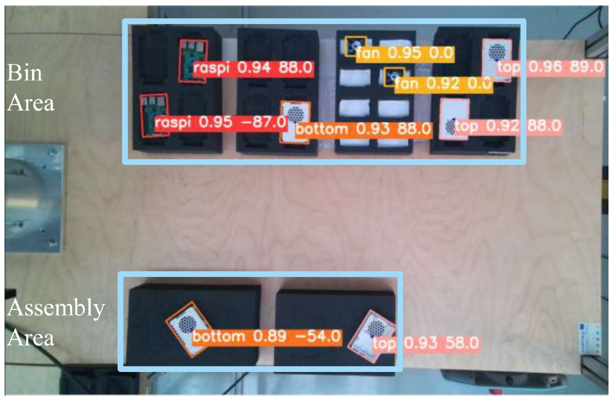

Fig. 8. Top view on the workstation and object detection result.

E. *Action Planner Module*

The Action Planner module serves as the cognitive center of the robot, responsible for processing the current state of the product and the workspace, determining the subsequent state and deriving the corresponding action for the robot to execute. To achieve this, the Action Planner relies on two essential inputs: a pre-defined textual description of the LLM task and the product structure, and the current state of the product and workspace. These inputs are sufficient to cooperatively assemble the product.

As we are not benchmarking all available LLMs, our LLM of choice is GPT-4, known for its robustness, large model size, and accessibility, despite not being free of charge.

The prompt template is based on role prompting, which involves providing the LLM with a specific "role" or context to guide its output generation. The prompt template includes two distinct roles: system and user. The first role, "system," is prepared only once and incorporates multiple techniques such as few-shot learning and chain-of-thought reasoning. These techniques address complexity, particularly with the Sensor-Box, as it allows for multiple assembly sequences. For example, the second step could involve either Cube 1 or Cube 2, and subsequent steps vary based on the chosen component. The second role, "user," communicates with the object detection module to update the LLM with the current state of the bin and assembly area. Fig. 9 illustrates an example.

> The assembly already has 0 raspi, 1 bottom, 0 fan, 0 top and 1 case, There are 3 raspi, 2 bottom, 3 fan and 3 top in the bin. Prepare final product including the existingcomponents if possible.

Fig. 09. Example of prompt

Fig. 10 presents a schematic of the prompt template used to set up the CoT reasoning for this use case. The prompt header outlines the general objective, specifying the role of the LLM as a robot, the general task of mutual assembly of a product, and detailed instructions, including examples. Following this, the product description is introduced without specifying which steps should be assigned to the human operator or the robot. The description includes the number of assembly parts, a numbered list of assembly parts, and a description of the product structure. The order of items in the textual description does not reflect the assembly order as long as the text accurately describes the assembly structure. For example, there is no difference between "the Circuit Board is placed on the Bottom Case" and "the Bottom Case houses the Circuit Board".

We then define a response template, which consists of four intermediate steps and a final output. The first intermediate step is to name the component that was mounted by the operator, the second step lists all components assembled by the operator, the third step provides a summary of what has been assembled so far and what remains, and the fourth step indicates which assembly parts should be assembled by the robot. This reasoning process ensures that the response is more robust and deterministic. The final response should be formatted in JSON to facilitate direct communication with the robot. The JSON output is structured as in:

[{"action": "action 1", "object": "object 1", "location": "location 1"}]

The actions primarily involve either pick and place task. The object represents a single component of the product, and the location is specially designed as either the source of pick task, which is bin area.

At the end of the prompt template, we explicitly remind that the human operator may skip one necessary assembly step, in which case the robot should assemble that step. The prompt concludes with the instruction that the operator always initiates the process.

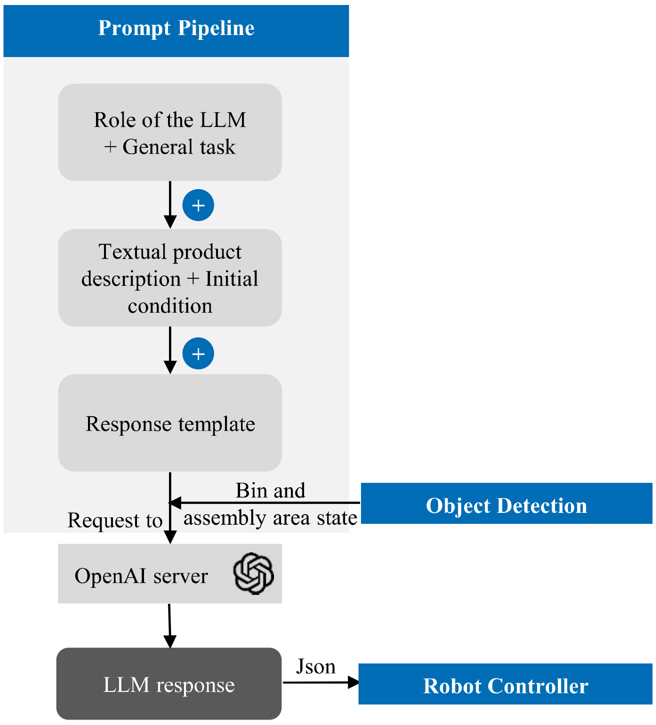

Fig. 10. Prompt template and communication with object detection module and robot controller.

The JSON-formatted action plan is communicated to the Robot Controller module via ROS topic and then stored in an action queue, ready to be executed by the robot and its gripper. At the same, all the modules are subscribing and publishing to the existing ROS topics to update the robot and human state, to generate online actions to complete the assembly of the product.

F. Robot Controller Module

The Robot Controller module receives the output from the Action Planner module in JSON format through a ROS topic. Hence, the robot knows which component to pick from the bin area but not its exact position. To determine it, the Robot Controller communicates with the Object Detection module to retrieve the coordinates, which are in the camera coordinate system. The Robot Controller then transforms these coordinates to the robot coordinate system using a homography matrix. To generate the path, waypoints are created between the calculated start position in the bin area and the predefined end position in the assembly area using a Bézier curve. With the MoveIt package from ROS, the robot executes the path and actuates the gripper to pick and place the component.

V. Evaluation and Results

To evaluate the assembly process for both products (Mini-Controller and Sensor-Box), two scenarios are tested. The first scenario, called continuous assembly, requires mutual assembly without any errors from the human operator; the LLM must return the correct (mechanically feasible) next assembly step, to be performed by the robot. The second scenario, called skipped step, involves mutual assembly where the human operator is allowed to skip one step, and the LLM is expected to identify and perform the skipped step. If the LLM proposes an incorrect step in the first scenario or fails to propose the skipped step in the second scenario, the assembly is halted, and the experiment is restarted.

The diagram in Fig. 11 shows the results of 25 experiments conducted for each product and scenario. GPT-4 achieved a 100% correctness rate for the Mini-Controller in both scenarios, as illustrated by the first two bars in Fig. 11. The third and fourth bars depict slightly lower scores, indicating the presence of some errors. The continuous assembly of the Sensor-Box has a correctness rate of approximately 85%, while the scenario where one step is skipped has a correctness rate of around 75%.

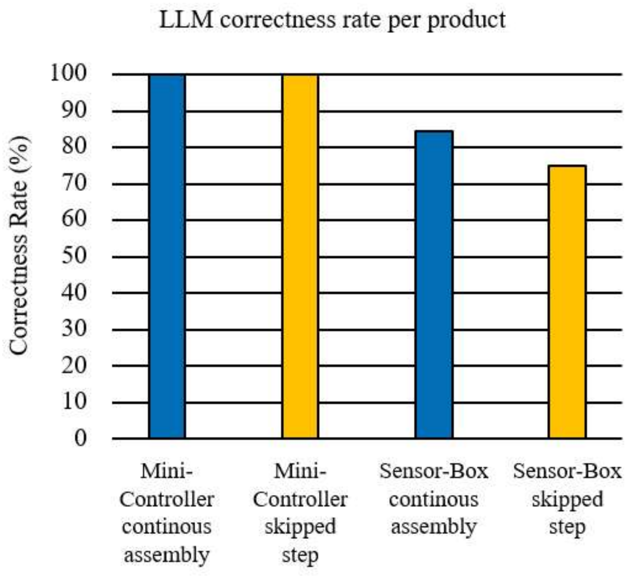

Fig. 11. Correctness rate per product

By examining the Sensor-Box, which exhibits a lower correctness rate and specifically focusing on the steps where errors initially occur, as described on the the y-axis in Fig. 12 showing the absolute number of errors, it is revealed that these errors predominantly occur within the first five steps. As highlighted in Fig. 6, assembly options diverge during these initial steps, beginning with the placement of the Bottom Case until securing the cubes. Skipping a step significantly increases the likelihood of errors, particularly in parts of the state space with multiple options. Overall, the LLM finds it more challenging to handle multiple options, whereas a sequential and continuous assembly process without skipped steps demonstrates greater robustness.

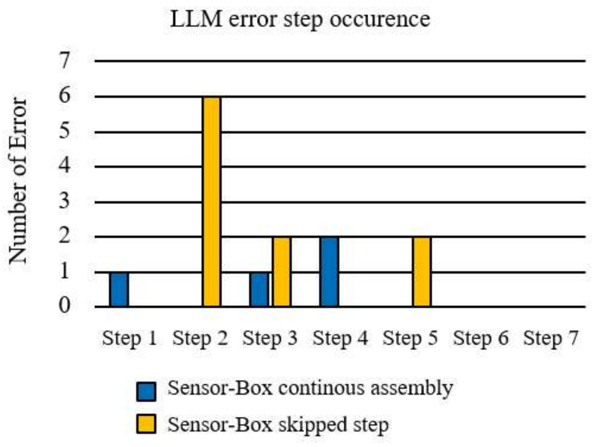

Fig. 12. Frequency of initial LLM errors after each step skipped by human

In summary, a strictly sequential state space results in a more error-resistant response from the LLM. The LLM is also capable of detecting when a step is skipped. Notably, the two products have a relatively small state space. We assume that even a larger, linear state space would maintain its error resistance as long as no more than one step is skipped.

Conversely, a state space with multiple pathways to the final state tends to be more error-prone. While our results are promising, we foresee that a larger state space with numerous assembly options per step would introduce additional challenges.

VI. Summary

This work demonstrates the use of an LLM as the reasoning module for cooperative robotics. The task involves mutually assembling two different products with a human operator. We implemented an LLM based on GPT-4 using basic instructions and few-shot learning to perform the mutual assembly task. The product structure was modeled using natural language descriptions. The LLM successfully handled the first product, which required a linear assembly sequence, without errors and rectified any skipped steps by the human operator. For the second product, which allows for multiple assembly sequences, the LLM was less reliable but still produced promising results. We note that a more complex product with multiple assembly options could challenge the current system. As a result, this approach performs better with linear-structured products, but integrating new technologies can further enhance the system's capabilities

For future work, we plan to extend this approach to handle errors caused by the human operator in real-time, evaluate the LLMs reasoning capabilities, which would enable better state space exploration.

Acknowledgment

This paper was authored within the context of the RICAIP research project (Research and Innovation Centre on Advanced Industrial Production). RICAIP receives funding from the European Union's Horizon 2020 program under Grant Agreement No. (GA: 857306).

References

[1] S. Makris, Cooperating Robots for Flexible Manufacturing, Cham: Springer International Publishing, pp. 373–380, 2021.

[2] L. Wang et al., “Symbiotic human-robot collaborative assembly,” CIRP Annals, vol. 68, no. 2, pp. 701–726, 2019.

[3] Ahmad, M. A., & Boca, P. P. (2018)., “Human-robot collaboration: A survey," IEEE Transactions on Human-Machine Systems, pp. 408–423, 2018.

[4] U. Othman and E. Yang, “Human-robot collaborations in smart manufacturing environments: Review and outlook," Sensors (Basel, Switzerland), vol. 23, no. 12, 2023.

[5] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," 2016. [Online]. Available: arXiv:1603.02199v4 [Accessed: Jul. 16, 2024].

[6] Gombolay, M., Jensen, R., Stigile, G., Son, S. H., & Shah, J., “Autonomous robots: Task allocation and scheduling for human-robot collaborative teams," in pp. 299–312.

[7] A. Hoffmann, L. Nagele, and W. Reif, "How to find assembly plans (fast): Hierarchical state space partitioning for efficient multi-robot assembly," 2020 Fourth IEEE International Conference on Robotic Computing (IRC), Taichung, Taiwan, 2020, pp. 172-177.

[8] D. McDermott, M. Ghallab, et al., "PDDL - The Planning Domain Definition Language," The AIPS-98 Planning Competition Committee, 1998.

[9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, 2018.

[10] "Ultralytics YOLOv5 models," PyTorch Hub. [Online]. Available: https://pytorch.org/hub/ultralytics_yolov5/. [Accessed: Jul. 16, 2024].

[11] L. S. Homem de Mello and A. C. Sanderson, "AND/OR graph representation of assembly plans," in IEEE Trans. Robot. Automat., vol. 6, no. 2, pp. 188-199, Apr. 1990.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numer. Math., vol. 1, no. 1, pp. 269-271, 1959.

[13] R. Tarjan, "Depth-First search and linear graph algorithms," SIAM J. Comput., vol. 1, no. 2, pp. 146–160, 1972.

[14] D. C. Kozen, "Depth-First and Breadth-First search," in The Design and Analysis of Algorithms. Texts and Monographs in Computer Science, New York: Springer, pp. 19–24.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Feermination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, July 1968.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484-489, Jan. 2016.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, et al., "Attention Is all you need," in Advances in Neural Information Processing Systems 30 (NIPS 2017), Dec. 2017. [Online]. Available: https://papers.nips.cc/paper/7181-attention-is-all-you-need. [Accessed: Jul. 16, 2024].

[18] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," 2020. [Online]. Available: arXiv:2010.11929 . [Accessed: Jul. 16, 2024].

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: arXiv:1512.03385v1 . [Accessed: Jul. 16, 2024].

[20] GPT-4. [Online]. Available: https://openai.com/gpt-4 [accessed: Jul. 16 2024].

[21] R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, and R. Soricut, "Gemini: A family of highly capable multimodal models," 2023. [Online]. [Accessed: Jul. 16, 2024].

[22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, et al., "Chain-of-Thought prompting elicits reasoning in large language models," Jan. 2022. [Online]. Available: http://arxiv.org/pdf/2201.11903. [Accessed: Jul. 16, 2024].

[23] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, “Large language models are Zero-Shot reasoners,” 2022. [Online]. Available: https://arxiv.org/abs/2205.11916. [Accessed: Jul. 16, 2024].

[24] "ROS," Robot Operating System. [Online]. Available: https://www.ros.org/. [Accessed: Jul. 16, 2024].