Integrating ChatGPT with Blockly for End-User Development of Robot Tasks

Luigi Gargioni
luigi.gargioni@unibs.it
Department of Information Engineering
University of Brescia
Brescia, Italy

Daniela Fogli
daniela.fogli@unibs.it
Department of Information Engineering
University of Brescia
Brescia, Italy

ABSTRACT

This paper presents an End-User Development environment for collaborative robot programming, which integrates Open AI ChatGPT with Google Blockly. Within this environment, a user, who is neither expert in robotics nor in computer programming, can define the items characterizing the application domain (e.g., objects, actions, and locations) and define pick-and-place tasks involving these items. Task definition can be achieved with a combination of natural language and block-based interaction, which exploits the computational capabilities of ChatGPT and the graphical interaction features offered by Blockly, to check the correctness of generated robot programs and modify them through direct manipulation.

CCS CONCEPTS

• **Human-centered computing** → *Collaborative and social computing devices*; • **Computer systems organization** → *External interfaces for robotics.*

KEYWORDS

Human-Machine Interaction, End-User Development, Human-Robot Collaboration, Collaborative Robots

ACM Reference Format:
Luigi Gargioni and Daniela Fogli. 2024. Integrating ChatGPT with Blockly for End-User Development of Robot Tasks. In *Companion of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24 Companion), March 11–14, 2024, Boulder, CO, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3610978.3640653

1 INTRODUCTION

The growing interest in the use of collaborative robots (cobots) to assist human operators in different work scenarios, such as manufacturing, healthcare, and logistics, opens up an important research question about the easiness of robot task definition. Cobots are flexible automation technologies, thus suitable to small-batch productions [15]; some models are also lightweight and can therefore be moved in different places within a work environment to support location-based tasks [25]. For these reasons, cobots must be quickly and easily programmed by end users, who may be expert in their work domain, but not necessarily expert in computer and/or robot programming.

The recent survey [1] highlights how end-user robot programming can be much more complex than traditional end-user programming: the created robot programs must in fact refer to physical objects and locations; in addition, the robot must interact with the surrounding environment, including humans, by moving its arm(s) in the space and performing actions on the physical objects. Therefore, an important objective of the research on end-user robot programming is defining methods that allow end users without expertise in robotics and programming to cope with this complexity. The literature presents methods based on programming by demonstration [14, 21], visual programming languages, and natural language programming. Programming by demonstration usually requires many trials, thus making the programming task inefficient. Visual programming languages, in turn, exploit flowchart notation [2], hierarchical trees [20] or block composition [9, 26]; in all these cases, programming concepts, such as variables, conditionals or loops, are visually expressed in the programming environment, but their meaning must be known to the user; in other terms, the user must possess some programming knowledge, especially because the user must conceive the whole program from scratch. With the skill-based visual programming language, [22] adopts a higher-level approach than the previous ones: the user can create a sequence of *skills*, namely robot actions (e.g., "pick object", "navigate to location", "rotate object"), to define the desired robot behavior. However, the approach appears as less flexible than those based on the other visual programming languages. Natural language programming has been proposed for instance in [4, 16, 17], even though the recent spread of Large Language Models (LLMs) has revived this approach. In [24], the authors explore OpenAI ChatGPT [18] as a potential versatile tool for robot programming; in their approach, a high-level function library is first created and then ChatGPT can parse user requests and convert them to a logical sequence of function calls. However, the user must be able to evaluate the code generated by ChatGPT and provide feedback on its quality and safety. The assumption is that the user is able to understand the generated code and suggest modifications, if needed. Other studies explore the use of LLMs in robotics for task planning [10, 23] and control [11], but do not consider the users as the recipient of generated code.

In this work, we do not simply propose a new, user-oriented, programming method for cobots, but an End-User Development (EUD) approach that combines different interaction modalities and can lead the user to generate robot programs in an unwitting manner [5]. EUD subsumes end-user programming [3, 12], since it is not simply focused on facilitating programming by end users. EUD

HRI '24 Companion, March 11–14, 2024, Boulder, CO, USA

ACM ISBN 979-8-4007-0323-2/24/03.
https://doi.org/10.1145/3610978.3640653

has a wider perspective in that it aims to empower end users in modifying, extending, and creating digital artifacts with software environments designed around their own domain concepts and work practices.

In the approach presented in this paper, the user goal is defining pick-and-place robot tasks. The idea is that the user can describe desired robot tasks to an assistant using natural language (as in the real world), but that what the assistant understands is not visualized as program code, like in [24], but through a more intuitive representation. In other words, we propose a EUD environment where the user interacts with a chat interface based on ChatGPT to describe pick-and-place robot tasks, and then verifies and possibly corrects the generated program through an intermediate visualization of the program based on Google Blockly [8]. The selection of ChatGPT is based on a thorough assessment of its computational capabilities [13], ease of adaptation in training, and facility of integration and customization. The use of Blockly for the graphical representation and programming of robot tasks arises from its inherent benefits in fostering an intuitive programming environment, as shown in [9], even though in our case the user does not need to define a Blockly program from scratch. In addition, pick-and-place tasks can be created based on the items (objects, actions, and locations) defined by the users themselves within the EUD environment, thus making it tailorable to different application domains.

The rest of the paper presents the architecture and implementation of the EUD environment. The generated programs can be executed on the DENSO COBOTTA collaborative robot.

2 ARCHITECTURE

The developed EUD environment is a web application implemented in adherence to a client-server paradigm, including a front-end developed in JavaScript/Typescript React and a back-end developed in Python Django; an SQLite database is used as data management system. The application architecture is shown in Figure 1. It encompasses four layers:

(1) The *User Interface Layer*: through this layer the user can interact either with a chat or a graphic interface to define pick-and-place tasks through natural language interaction or block-based interaction respectively;
(2) The *Task Definition Layer*: this layer is the application logic of the EUD environment, where ChatGPT or Blockly are exploited to process users' requests coming from the User Interface layer.
(3) The *Data Layer*: in this layer robot programs are stored in the database using a JSON format compatible with Blockly; parsing functions transform the output of ChatGPT into JSON data that the Blockly engine may recognize.
(4) The *Execution Layer*: here, the task execution by the robot is managed by means of a program template.

Going more in detail in the user workflow, let us assume that the definition of a new task for the robot begins with the interaction with the Chat interface (*Step 1* in Figure 1). User messages, conveyed via the chat, are directed to an Adapter, which exploits the ChatGPT API to interpret the user's request *(Step 2)*. This module provides ChatGPT not only with the user's request, but also with specific constraints and instructions, to ensure precise and non-deterministic interpretation of the robot task *(Step 3)*. After the task definition is complete and the interaction with the Chat ends, a custom JSON representation of the robot program is generated *(Step 4)*. Following this, parsing functions are employed to convert the custom JSON format used by the Chat into a JSON format that can be interpreted by Blockly *(Step 5)*, enabling the program to be visualised in a graphical format *(Step 6)*. At this stage, the user can check the correctness of the program in the Graphic interface (*Step 7*), and can either confirm or modify it (*Step 8*). Finally, the Blockly program representation is used by a program template *(Step 9)*; this program will execute the elementary robot actions needed to accomplish the task *(Step 10)*, like moving the arm to a specific position or searching for an object in the working area.

Advanced users of the application can directly define a new robot task through the Graphic interface starting from *Step 7*. While this strategy guarantees faster execution, it necessitates more cognitive effort than using the Chat interface, and then simply check and correct what AI has generated.

3 IMPLEMENTATION

As an example of the robot task definition, useful for explaining the workflow in detail, let us present an interaction scenario:

The user aims to organize their desk using a collaborative robot. They will require to specify one or more items that ought to be picked up, such as the highlighter item, along with the position to drop it, in this case a box. The user wants to execute a shaking action before disposing the highlighter to prevent its tip from drying. Once the items are defined, they can request through the chat to execute the operation and repeat it five times, since five highlighters are present on the desk. Subsequently, the user can check the task created in the graphic interface and run it on the robot.

3.1 Item definition

To create a task for the robot, the first step is to define the items, namely the object to pick *(highlighter)*, the location where to place the object *(box)* and the optional action *(shaking)* to be executed between the pick and place steps. These items will then be saved in the corresponding libraries to be re-used for future tasks. An object is a tangible entity that a robot can manipulate. To create a new object, the user should capture a picture of it using the robot camera and set all relevant data required for proper manipulation. For identification purposes, users must provide an object name and optionally a set of synonyms to be used by the natural language processing function of the chat. Technical data must also be provided, pertaining to the required force for object grasping and the approach to the object, i.e. the Z-axis distance the robotic arm needs to attain in order to pick up the object. After having acquired the photo of the object, the image is processed through segmentation, binarization, filtering, and contour searching. At the time of execution, the robot can efficiently search for items in the designated area, retrieve their position and alignment, and successfully grab them. With this procedure, the user can define the "highlighter" object of our running example. A location refers to a point in space that serves as the target for a "place" operation. Therefore, its definition remains straightforward: upon providing the identification information for the location, the robot arm must be manoeuvred

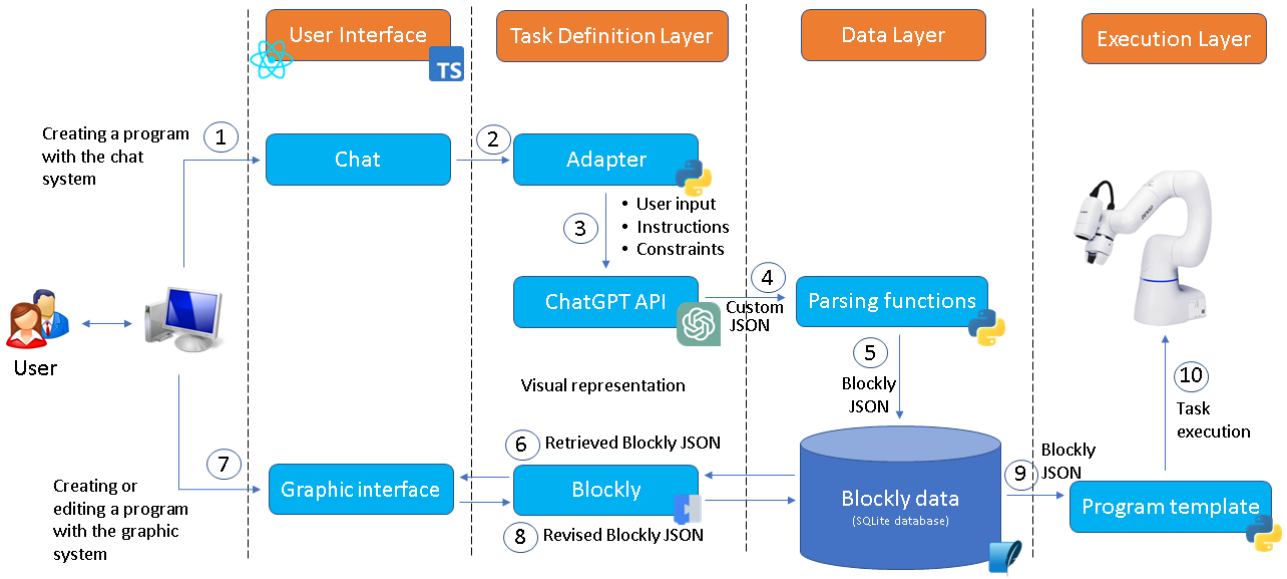

Figure 1: The prototype architecture and the collaborative robot COBOTTA by DENSO WAVE Ltd.

to the intended point before acquiring the specific position. Here, the user can define the "box" location for our running example. An action enables the robot to move along a specified path between a "pick" and a "place" step. It consists of a sequence of points (a cloud of points) that the robot arm must reach to follow a defined path. Recording a cloud of points is achieved by guiding the robot arm along the desired path and repeatedly pressing a specific button in the software application to capture significant points of the trajectory. As for objects and locations, identification data related to the action must also be provided. In this way, the user can define the "shaking" action for our running example.

3.2 Task definition and execution

The chat-based interface is used to guide the user through the programming of the robot tasks. Figure 2 presents the interaction between the user and our application integrating ChatGPT, considering the scenario introduced above. We specifically chose to use the *gpt-3.5-turbo* model, which is recognized as the most powerful model currently available for a free trial.

To tailor the behavior of ChatGPT, it must be provided with clear instructions about its goals and context. As mentioned above, the *Adapter* is responsible for handling the calls to the ChatGPT APIs and instructing the model about its goals. All instructions are given as prompts at the beginning of the conversation and are transparent to the user. This specification allows the model to recognize the details required to define a complete robot task. In addition to these instructions, the request from the *Adapter* includes the JSON data format expected as the output of the model. The desired output is defined by passing a function to ChatGPT *Chat completions API*, which expects as parameter a representation of the custom JSON format. In this way, the API will respond with a JSON object compliant with the expected format. As to the *temperature* parameter of the model, the value 0.2 was chosen to obtain a quite deterministic reasoning and prevent creative interpretation of the user's intentions. However, a small degree of creativity was retained to allow the chat to better adapt to user's requests and exhibit a human-like behavior.

After the dialogue is concluded, ChatGPT produces a task description in a custom JSON format (see the top right-hand corner of Figure 2). This format was introduced since ChatGPT was not able to always provide an output compatible with Blockly. The custom JSON data is passed to Python functions, which parse the data and transform them into a Blockly-compatible JSON format.

Blockly is a library of puzzle-like visual blocks, organized in categories, which allows creating intuitive programming interfaces, and thus empowering users to conceptualise and articulate program logic in a highly comprehensive manner. For the sake of integration in our EUD environment, Blockly categories and blocks have been properly tailored. The *Logic* category has been created for logic constructs, such as "When" and "Repeat". The *Events* category encompasses conditions like "Sensor" and "Find" (the former is used to check whether a sensor signal is arrived, while the latter is used to check if a specific object is recognized). *Steps* is the category containing basic robot skills, like "Pick", "Place", and "Processing". Finally, the *Objects*, *Locations*, and *Actions* categories serve as user-defined libraries of items, which can be used like variables, without that the user must know the variable concept.

After the generation of the Blockly JSON data, the parsing functions query the database to retrieve the identifiers of objects, locations, and actions involved in the task defined through the chat, and integrate them in the data structure as meta-data. Item identifiers are needed for the subsequent task execution by the robot, to obtain the other technical data (e.g., object photo, location position, etc.). Item searching in the database is performed also using synonyms specified during the item definition. Once the retrieval of the item

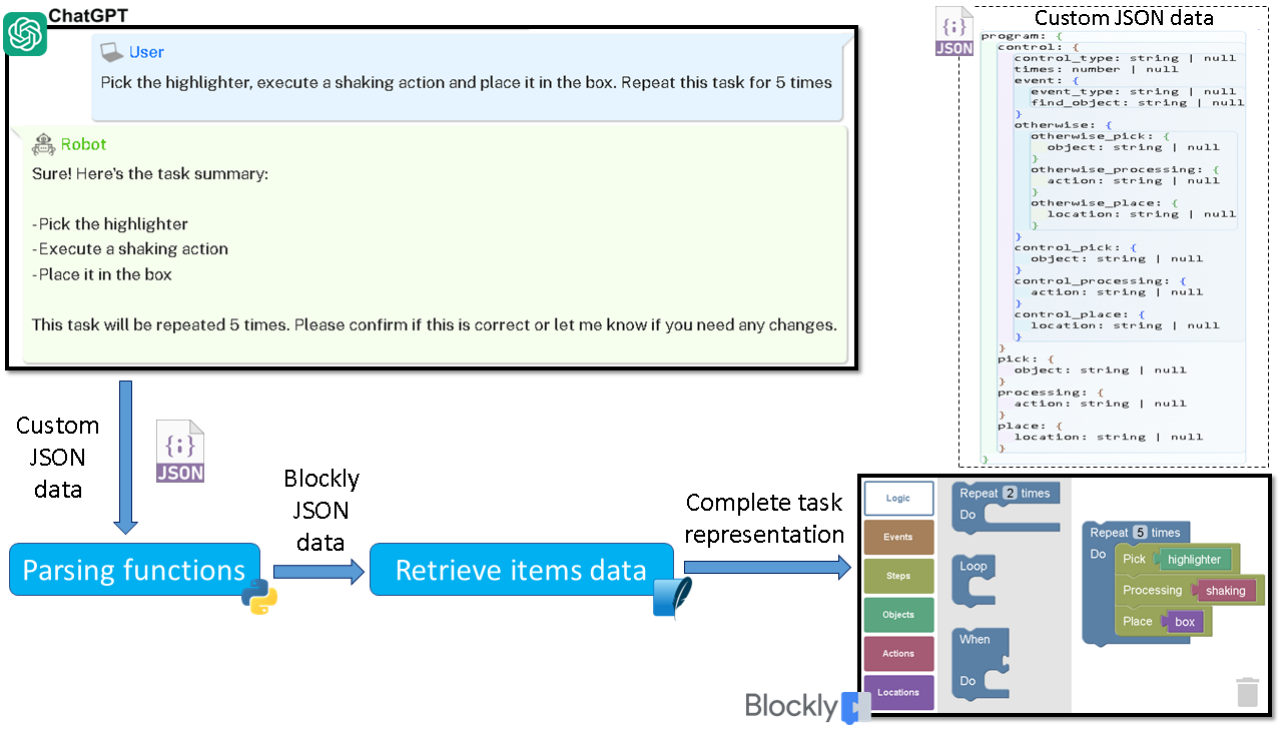

Figure 2: The ChatGPT-Blockly workflow

details is finished, a data format that includes all the required information for execution is available and compatible with Blockly. Subsequently, the task is displayed in the graphical interface (see the bottom right-hand corner of Figure 2).

The task can now be executed on the COBOTTA robot. This robot is designed to be lightweight, easily portable, and highly adaptable. It is a single articulated arm with six axes, and its standard configuration includes a camera and a gripper suited for pick-and-place operations. To execute the task, a Python program template is activated, which analyzes the Blockly JSON code and determines the elements that are essential for generating an executable sequence of robot instructions. Beyond including calls to proprietary DENSO APIs, these ones also include the objects to pick up, the actions to execute, the locations where the objects should be placed, and the conditions and cycles to apply. During execution, the robot camera searches for the objects to be picked up, which are then manipulated by the robot arm and finally placed in the desired location.

4 DISCUSSION AND CONCLUSION

The system here described is inspired to the hybrid approach to user-oriented programming presented in [6]. In this paper, we experimented ChatGPT capabilities for user intent recognition to create domain-independent robot programs. Our objective was not to generate the final code for the robot based on the user's input, but rather to obtain a task representation that could be processed in our EUD environment, enabling users to evaluate the program correctness via Blockly, here tailored to represent pick-and-place robot tasks. In this way, non-expert users can visually check robot programs and modify them through drag-and-drop of the blocks, or direct manipulation of the variables corresponding to users' defined items.

We plan in the future to evaluate system usability with the participation of university students not enrolled in computer science courses, who must acquire basic programming skills in their curriculum. In this way, we aim also to assess whether our hybrid approach could be employed for educational goals, to improve the learning process of imperative programming languages. We are experimenting the same approach to support pharmacists in the definition of robot tasks for the preparation of personalized medicines [7]; in this case, a simpler block-based visualization (more suitable to the target users) than Blockly was implemented. Positive qualitative feedback has been collected till now.

As to the limitations of the approach, a first one concerns the complexity of the tasks that can be defined. In the chat, only the generation of tasks without nested logic has been implemented. To define more complex tasks, users must possess adequate computational thinking skills. It is essential to further investigate ways to enhance task complexity, while minimizing the required end-user technical expertise. A second limitation concerns the image processing algorithms used for object recognition. These algorithms are quite simple and basic; it could be interesting to explore the new approaches to computer vision that have recently emerged along with LLMs, such as GPT-Vision [19].

REFERENCES

[1] Gopika Ajaykumar, Maureen Steele, and Chien-Ming Huang. 2021. A Survey on End-User Robot Programming. 54, 8, Article 164 (oct 2021), 36 pages. https://doi.org/10.1145/3466819

[2] Sonya Alexandrova, Zachary Tatlock, and Maya Cakmak. 2015. RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 5537–5544. https://doi.org/10.1109/ICRA.2015.7139973

[3] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. 2019. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software* 149 (2019), 101–137. https://doi.org/10.1016/j.jss.2018.11.041

[4] Nina Buchina, Sherin Kamel, and Emilia Barakova. 2016. Design and evaluation of an end-user friendly tool for robot programming. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 185–191. https://doi.org/10.1109/ROMAN.2016.7745109

[5] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. 2008. End Users as Unwitting Software Developers. In *Proceedings of the 4th International Workshop on End-User Software Engineering* (Leipzig, Germany) *(WEUSE '08)*. Association for Computing Machinery, New York, NY, USA, 6–10. https://doi.org/10.1145/1370847.1370849

[6] Daniela Fogli, Luigi Gargioni, Giovanni Guida, and Fabio Tampalini. 2022. A hybrid approach to user-oriented programming of collaborative robots. *Robotics and Computer-Integrated Manufacturing* 73 (2022), 102234.

[7] Luigi Gargioni, Daniela Fogli, and Pietro Baroni. 2023. Designing Human-Robot Collaboration for the Preparation of Personalized Medicines. In *Proceedings of the 2023 ACM Conference on Information Technology for Social Good* (Lisbon, Portugal) *(GoodIT '23)*. Association for Computing Machinery, New York, NY, USA, 135–140. https://doi.org/10.1145/3582515.3609527

[8] Google. 2012. Blockly. https://developers.google.com/blockly

[9] Justin Huang and Maya Cakmak. 2017. Code3: A System for End-to-End Programming of Mobile Manipulator Robots for Novices and Experts. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction* (Vienna, Austria) *(HRI '17)*. Association for Computing Machinery, New York, NY, USA, 453–462. https://doi.org/10.1145/2909824.3020215

[10] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand (Proceedings of Machine Learning Research, Vol. 205)*, Karen Liu, Dana Kulic, and Jeffrey Ichnowski (Eds.). PMLR, 1769–1782.

[11] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as Policies: Language Model Programs for Embodied Control. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*. IEEE, 9493–9500. https://doi.org/10.1109/ICRA48891.2023.10160591

[12] Henry Lieberman, Fabio Paternò, and Volker Wulf. 2006. *End User Development (Human-Computer Interaction Series)*. Springer-Verlag, Berlin, Heidelberg.

[13] Zhi Wei Lim, Krithi Pushpanathan, Samantha Min Er Yew, Yien Lai, Chen-Hsin Sun, Janice Sing Harn Lam, David Ziyou Chen, Jocelyn Hui Lin Goh, Marcus Chun Jin Tan, Bin Sheng, et al. 2023. Benchmarking large language models' performances for myopia care: a comparative analysis of ChatGPT-3.5, ChatGPT-4.0, and Google Bard. *EBioMedicine* 95 (2023).

[14] Yueyue Liu, Zhijun Li, Huaping Liu, and Zhen Kan. 2020. Skill transfer learning for autonomous robots and human–robot cooperation: A survey. *Robotics and Autonomous Systems* 128 (2020), 103515. https://doi.org/10.1016/j.robot.2020.103515

[15] Malin Löfving, Peter Almström, Caroline Jarebrant, Boel Wadman, and Magnus Widfeldt. 2018. Evaluation of flexible automation for small batch production. *Procedia Manufacturing* 25 (2018), 177–184. https://doi.org/10.1016/j.promfg.2018.06.072 Proceedings of the 8th Swedish Production Symposium (SPS 2018).

[16] Dipendra K. Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. 2016. Tell Me Dave: Context-Sensitive Grounding of Natural Language to Manipulation Instructions. *Int. J. Rob. Res.* 35, 1–3 (jan 2016), 281–300. https://doi.org/10.1177/0278364915602060

[17] Dipendra K. Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. 2016. Tell me Dave: Context-sensitive grounding of natural language to manipulationinstructions. *The International Journal of Robotics Research* 35, 1-3 (2016), 281–300. https://doi.org/10.1177/0278364915602060

[18] OpenAI. Accessed: November 20th, 2023. ChatGPT. https://chat.openai.com/

[19] OpenAI. Accessed: November 24th, 2023. GPT-Vision API Documentation. https://platform.openai.com/docs/guides/vision

[20] Chris Paxton, Andrew Hundt, Felix Jonathan, Kelleher Guerin, and Gregory D. Hager. 2017. CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (Singapore, Singapore). IEEE Press, 564–571. https://doi.org/10.1109/ICRA.2017.7989070

[21] Leonel Rozo, Sylvain Calinon, Darwin G Caldwell, Pablo Jimenez, and Carme Torras. 2016. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics* 32, 3 (2016), 513–527.

[22] Casper Schou, Rasmus Skovgaard Andersen, Dimitrios Chrysostomou, Simon Bøgh, and Ole Madsen. 2018. Skill-based instruction of collaborative robots in industrial settings. *Robotics and Computer-Integrated Manufacturing* 53 (2018), 72–80. https://doi.org/10.1016/j.rcim.2018.03.008

[23] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*. IEEE, 11523–11530. https://doi.org/10.1109/ICRA48891.2023.10161317

[24] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. *ChatGPT for Robotics: Design Principles and Model Abilities*. Technical Report. Microsoft. https://www.microsoft.com/en-us/research/uploads/prod/\2023/02/ChatGPT_Robotics.pdf

[25] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. 2018. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics* 55 (2018), 248–266. https://doi.org/10.1016/j.mechatronics.2018.02.009

[26] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C. Shepherd, and Diana Franklin. 2018. Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173940