

A PYTHON-BASED EDUCATIONAL FRAMEWORK FOR NATURAL LANGUAGE-DRIVEN OFFLINE PROGRAMMING OF INDUSTRIAL ROBOTS IN ROBODK

Cozmin Cristoiu^{1,3}[0000-0001-9716-1908], Roxana-Mariana Nechita^{1,2} [0009-0004-7556-5572], Bogdan Stoian¹ [0009-0003-4946-2409],
Bogdan Marian Verdet^{1*} [0009-0001-1856-9786]

¹National University of Science and Technology POLITEHNICA Bucharest, 313 Splaiul Independentei, Bucharest, Romania

²National Institute of Research and Development in Mechatronics and Measurement Technique, 6-8 Pantelimon Road, Bucharest, Romania

³Robodictive SRL, Gheorghe Doja Street, Pitești, Romania

E-mail: bogdan.verdete@upb.ro

Abstract - This paper presents a Python-based educational framework for offline programming of industrial robots in the RoboDK environment through natural language commands. The proposed system aims to reduce the entry barrier for novice users by allowing pick-and-place tasks to be specified in intuitive textual form, without the use of conventional robot programming languages. The application relies on a modular architecture that includes natural language interpretation, semantic extraction of action-object-destination elements, intermediate task representation, and automatic generation of RoboDK-compatible robot routines. The system was implemented and validated in a virtual RoboDK station containing an industrial robot and multiple manipulable objects. For experimental evaluation, a set of 20 English-language commands with different linguistic formulations was used in order to assess the correctness of object identification, action detection, destination recognition, and program execution. The results showed high performance for well-defined commands, with correct action identification in 19 out of 20 cases, correct destination recognition in 19 out of 20 cases, correct object identification in 16 out of 20 cases, and successful execution in simulation in 19 out of 20 cases. The experimental analysis also highlighted limitations related to ambiguous commands, incomplete object specification, and restricted vocabulary coverage. The results confirm the feasibility of using natural language as an educational interface for offline robot programming in controlled RoboDK scenarios. At the same time, the study outlines several directions for further development, including improved ambiguity handling, broader semantic coverage, support for more complex task sequences, and the future integration of advanced artificial intelligence methods.

Keywords: AI, Offline-programming, Industrial robots, Python, Natural language.

1. Introduction

Modern industrial systems have progressed in tandem with the rise of "Industry 4.0," an approach to manufacturing and material handling that prioritizes digitalization, automation, and AI to maximize efficiency [1,2]. Because of their widespread usage in tasks including processing, material handling, assembly, and inspection, industrial robots have emerged as crucial elements of adaptable production environments. Industrial robots are ubiquitous, but there is still a significant learning curve associated with controlling them,

requiring expertise in kinematics, robotics, and specific programming environments [3].

Both online and offline programming tools, including RoboDK and teach pendant programming, are widely used to program industrial robots. While these methods provide a lot of leeway and make precise programming and simulation possible, they could be somewhat intimidating to novices and those without much experience, particularly in classrooms. Therefore, there is a growing need for more natural ways of interacting with robots that may streamline programming without sacrificing the rational link between human purpose and machine reaction [1,3].

The use of natural language to the interaction of human and robotic systems is an encouraging trend in this field. By letting users express tasks in a familiar textual manner instead than via typical robot programming vocabulary, natural language interaction might make robot programming more accessible. Past research has shown that robotic actions, semantic concepts, and environmental awareness may understand and correlate with natural language instructions [4–7]. Recent advances in artificial intelligence have expanded the range of possibilities for connecting language and action in robotics, with examples including language-guided manipulation and transformer-based robotic control frameworks [8–10].

When designing tasks for robots to do, it is crucial to consider how the system will manage trajectories, motion continuity, and movement optimization [11,12]. From an educational perspective, these specifics are important because they demonstrate to students and new users the process of understanding instructions and translating general job descriptions into useful robotic routines. At the same time, studies in engineering education and applied mechatronics have shown that smart digital tools, multimodal interaction, and AI-assisted systems are becoming more important [13–19].

This study builds on these advancements by proposing an instructional software for offline programming industrial robots in the RoboDK environment using natural language instructions, which is based on Python. The aim of the system is to facilitate the translation of user-defined, naturally-spoken English sentences into structured task representations and executable RoboDK routines for basic pick-and-place activities. With its modular design, the suggested method offers a transparent foundation for both current and future educational applications by separating natural language processing, semantic mapping, representation of intermediate tasks, and robot program creation.

To demonstrate the feasibility of developing an educational interface for controlled offline robot programming that is driven by plain language, rather than to provide a general solution for industrial programming, is the main objective of this work. This study lays the groundwork for industrial robot programming tools that are easier for humans to use and supports the argument that robotics courses should include current paradigms of AI-oriented interfaces. The modular nature of the proposed system enables more complex robotic tasks to be executed, greater semantic flexibility, and more advanced language interpretation [20].

2. Methodology

This research shows an attempt to develop a software system that can translate plain-language commands given to industrial robots in the RoboDK environment into commands that the robots can actually execute. The system's goal is to make learning a standard robot programming language easier for novices by providing them with a natural way to explain basic robotic activities in text form. This will make the system more useful for educational purposes. Accessing, testing, and expanding upon the processing stages is now much simpler because of the application's modular architecture and Python implementation [1–3].

Once the user inputs a common instruction, such as "place the yellow cylinder on the table" or "transfer the green cube to the red box," the procedure begins as normal. As soon as the system grasps the command, it initiates a series of operations, including robot programming and RoboDK simulations. You can see the whole process in Figure 1.

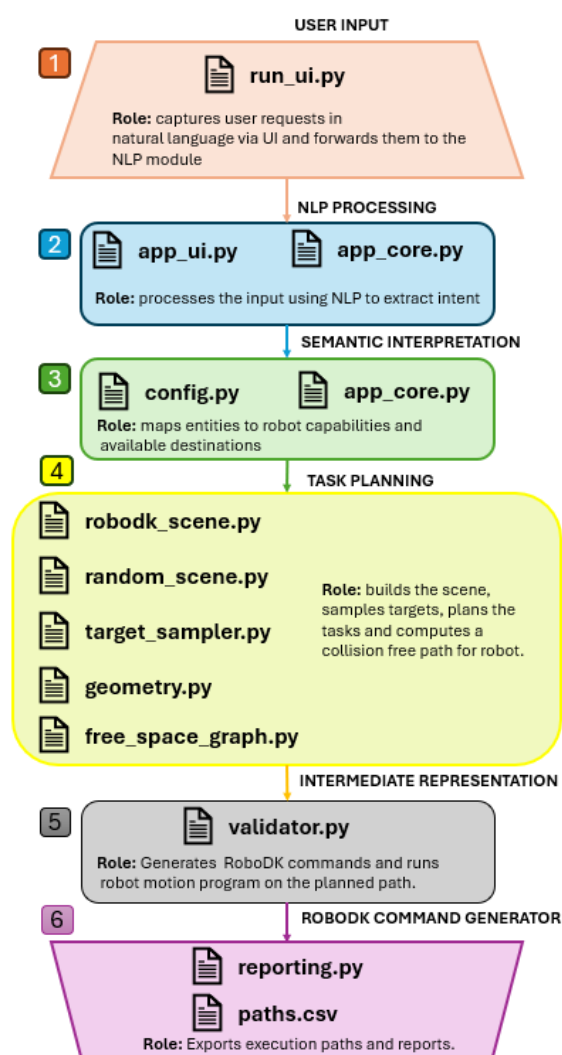


Figure 1: Application Workflow

The first step in finishing the assignment is to extract the most important semantic components from the given text. Just the task at hand and a suitable spot to set it down are necessary, apart from the item to be handled. The system must be able to distinguish the action "move," the object "green cube," and the destination name "red box" in order to carry out the instruction "Move the green cube to the red box." Language interpretation at this level is rule-based and relies on lexical structures and patterns found in dictionaries. In supervised learning contexts with a limited task vocabulary, this approach is effective when demonstrating the impact of human speech on robot behavior is the primary goal [5,6].

After retrieving the semantic elements, the next step is to convert them to an intermediate task-friendly format. Any platform may benefit from this format's organized internal description of the command. By doing so, the system is able to differentiate between the stages of language interpretation and robot execution. Parts of the job that are normalized in the intermediate model include the requested action, the identified source item, and the intended destination. Not only does this partition make things more clear, but it also makes room for future developments toward more advanced language understanding systems [9,10].

Step two involves translating the intermediate task representation into instructions that are compatible with RoboDK. To carry out the pick-and-place process, the app uses the RoboDK API to produce the necessary sequence of activities. Among these tasks is the creation of the necessary robot motion steps, the definition of the target points linked with grabbing and placement, and the identification of the matching items in the virtual station. This eliminates the need for human intervention in programming the robot's trajectory by having the system construct an exhaustive offline programming process in response to a natural language command.

The produced procedure, from an implementation perspective, is logically similar to a regular pick-and-place operation. Reaching the source item is the first step in guiding the robot. After that, it's either picked up or connected to the item, then moved to the designated place, and lastly released there. To make the motion sequence within the RoboDK station more clearly organized, it is also possible to produce additional intermediate positions. This structured approach is especially helpful in the field of education because it allows

students to see the transformation of user-supplied textual instructions into a sequence of robotic activities.

A key component of the method is the software's modular design. There is a distinct function for each processing stage, including semantic mapping, developing robot programs, representing intermediate tasks, and natural language interpretation. This architecture simplifies the application's readability as well as its extension and maintenance. Moreover, it paves the way for future system revisions to include more advanced AI methods, such as massive language models or machine learning models, in place of the current rule-based interpretation module [9,10].

In order to validate the system, we tested it in a simulated laboratory setting using a RoboDK controller, an industrial robot, and a number of manipulable objects. The objective of the evaluation was to validate the system's comprehension and execution of simple English pick-and-place commands inside the virtual setting, in addition to its competence in recognizing actions, objects, and destinations. Therefore, the suggested method is not primarily concerned with general industrial programming but rather with the instructional framework for offline robot programming using natural language.

3. Results

A suite of test cases was used to assess the built application. These scenarios were created to ensure that the natural language interpretation module and its connection with the RoboDK environment were functioning correctly. We conducted the validation in a controlled virtual station to see whether we could successfully translate basic English instructions into executable pick-and-place procedures. Proper object recognition, action identification, destination identification, and RoboDK program execution success were the four primary focuses of the examination.

One industrial robot and several items were placed in a simulated RoboDK station as part of the testing setup. There were three boxes that served as target locations, three cylinders that were various colors, and three multi-colored cubes throughout the scene. At launch, the app establishes a connection to RoboDK, finds the accessible robot, and recognizes the virtual station's configured items. Figure 2 depicts the testing virtual environment, while Figure 3 shows the produced application's graphical user interface.

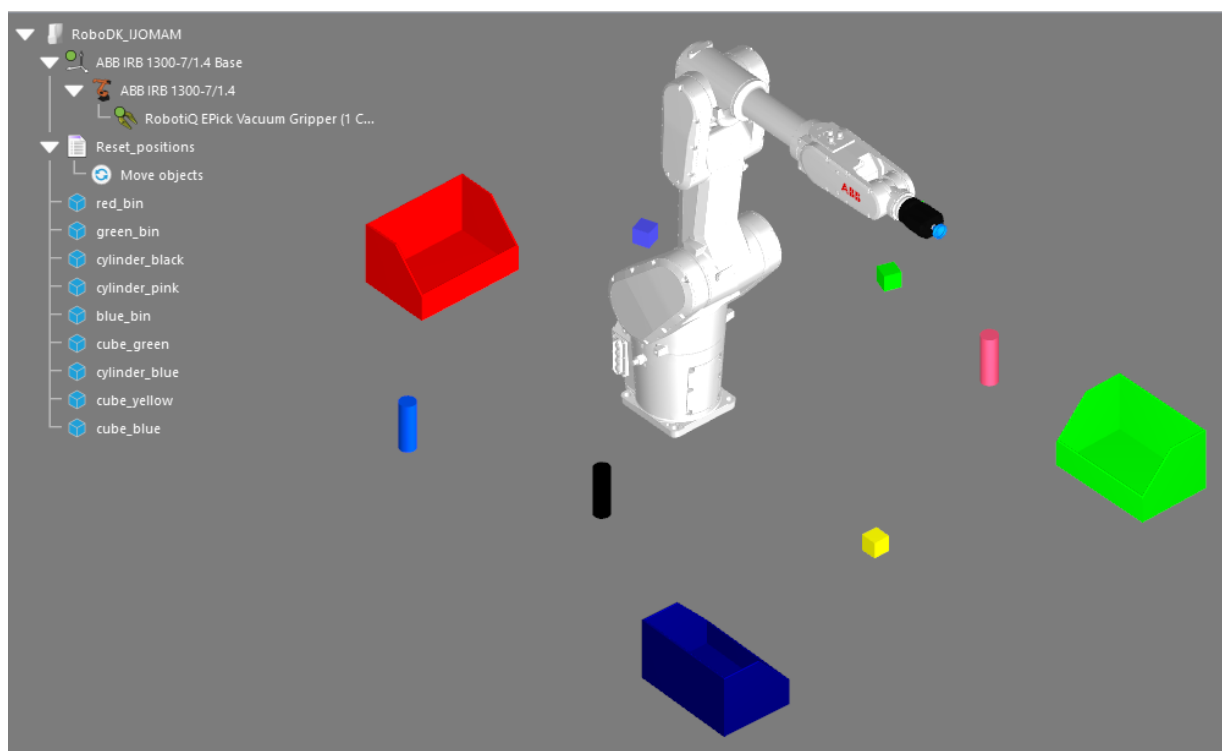


Figure 2: RoboDK virtual station

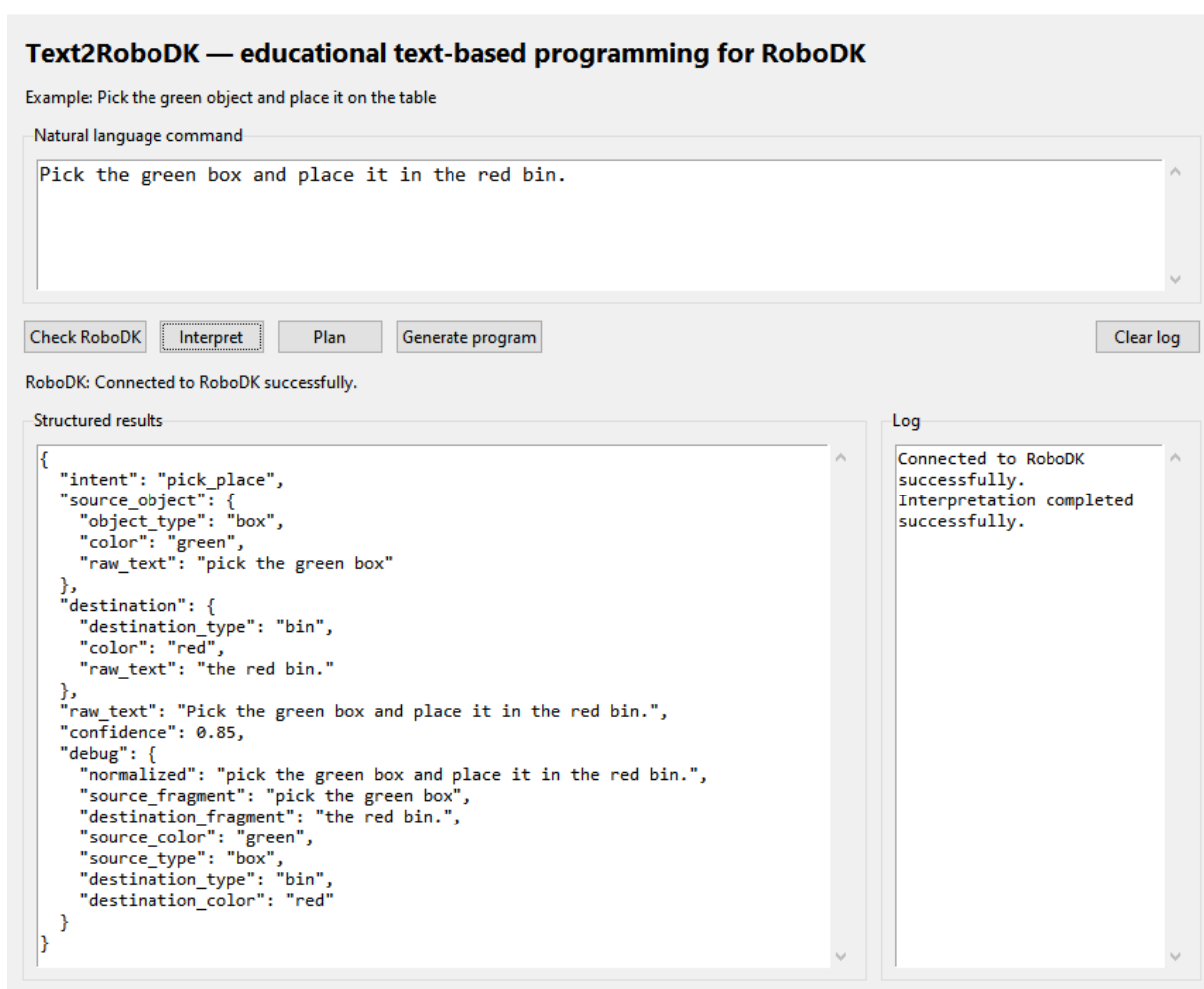


Figure 3: Application user interface

The RoboDK virtual station used for experimental validation is shown in Figure 2. The robot, items, and placement locations are specified in the execution environment, which is provided by this station. We chose this setup so that we could test object manipulation tasks under controlled settings several times. A battery of twenty natural language commands was used for functional validation. These instructions were from the English language and had subtle word changes, new wording, and altered sentence patterns. The goal of this series of tests was to see whether the system can correctly comprehend many methods of expressing the same pick-and-place operation.

"Transfer the green cube to the red box," "Receive the blue cylinder and set it in the red bin," "Set the green cube into the red bin," and "Put the blue cylinder in the green box" are all instances of instructions that have been tested.

The following steps were followed for every command: analyzing the input phrase for semantic meaning, extracting the major task parts, creating an internal representation of the task, and finally, creating the RoboDK program.

The first test command's full processing sequence is depicted to help explain the workflow more clearly. From the RoboDK station's starting point to its interpretation of sentences, planning of tasks, and automated program development, the following figures demonstrate the essential steps.

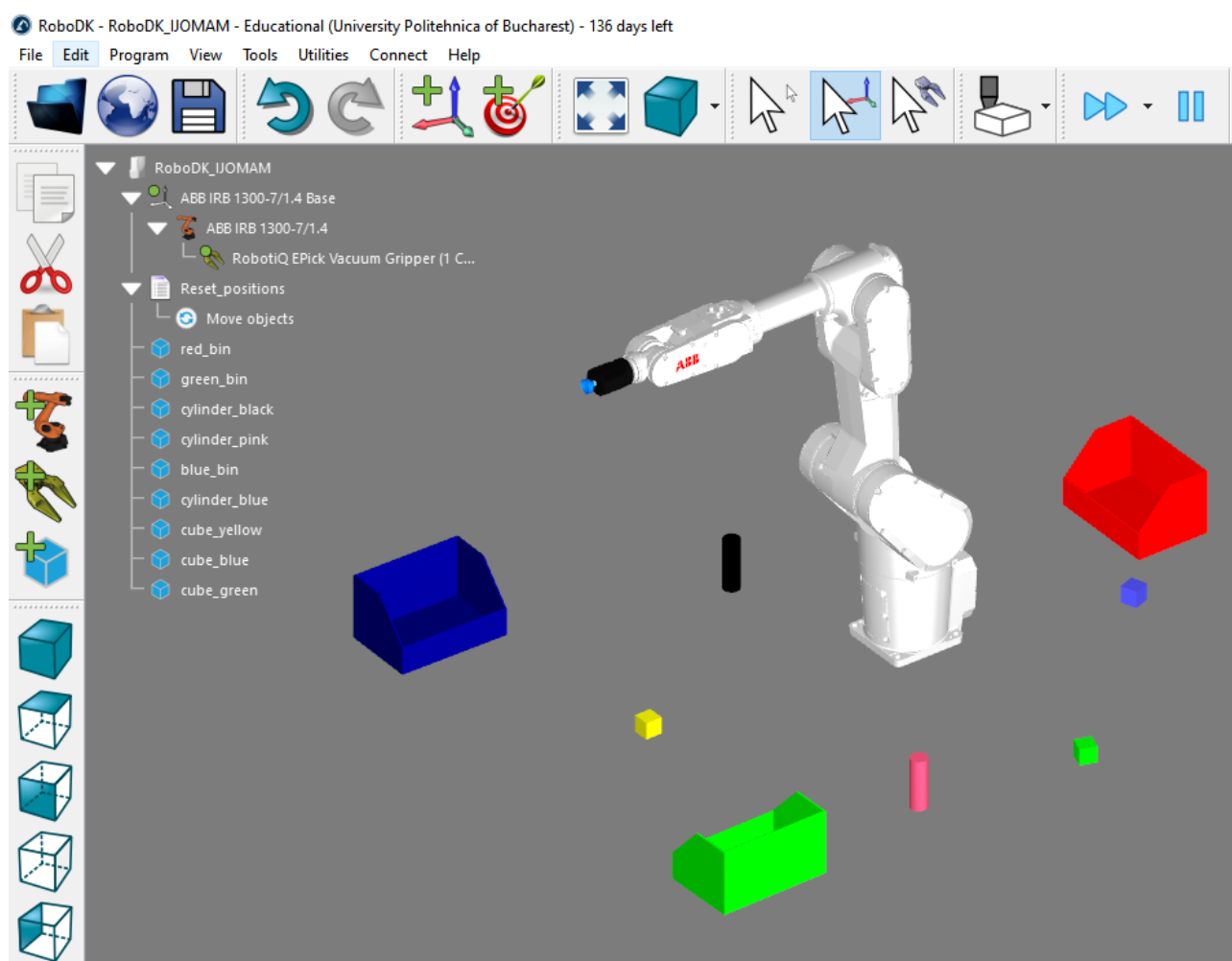


Figure 4: Station and elements in RoboDK

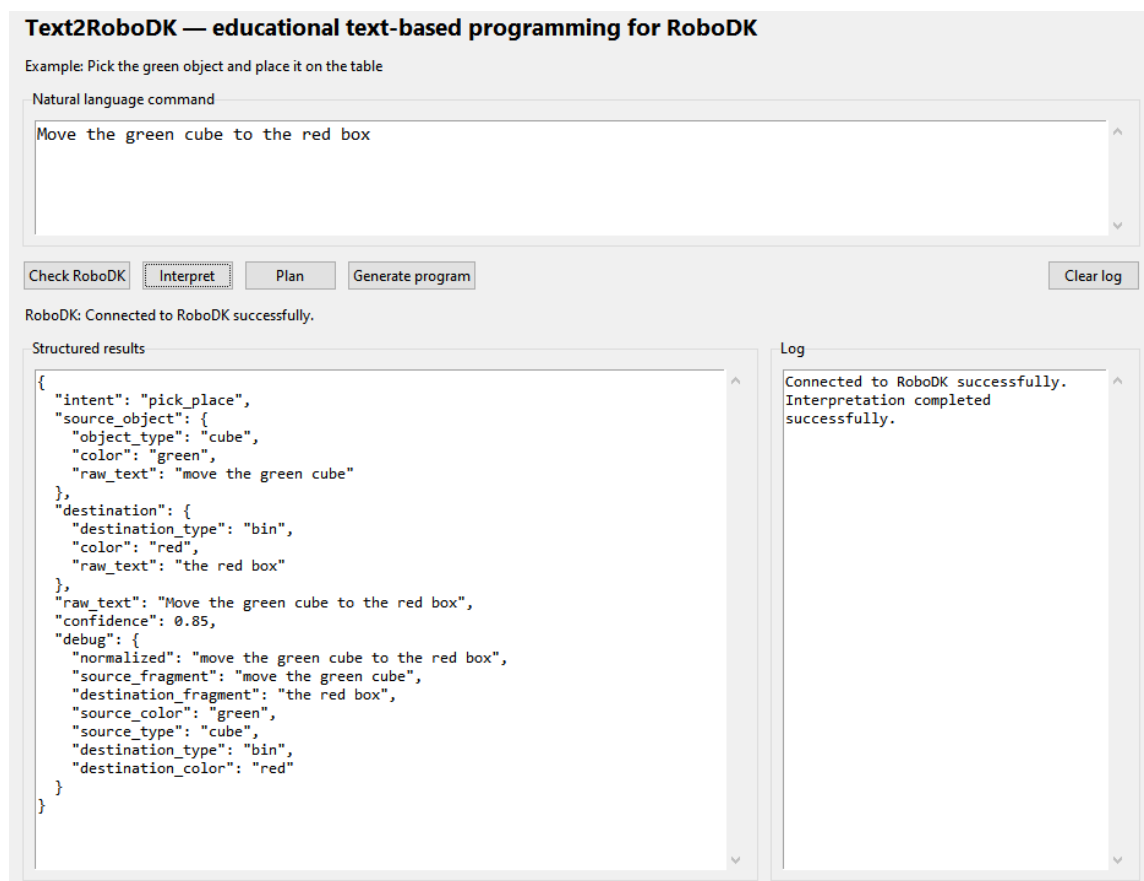


Figure 5: Interpretation of the sentence

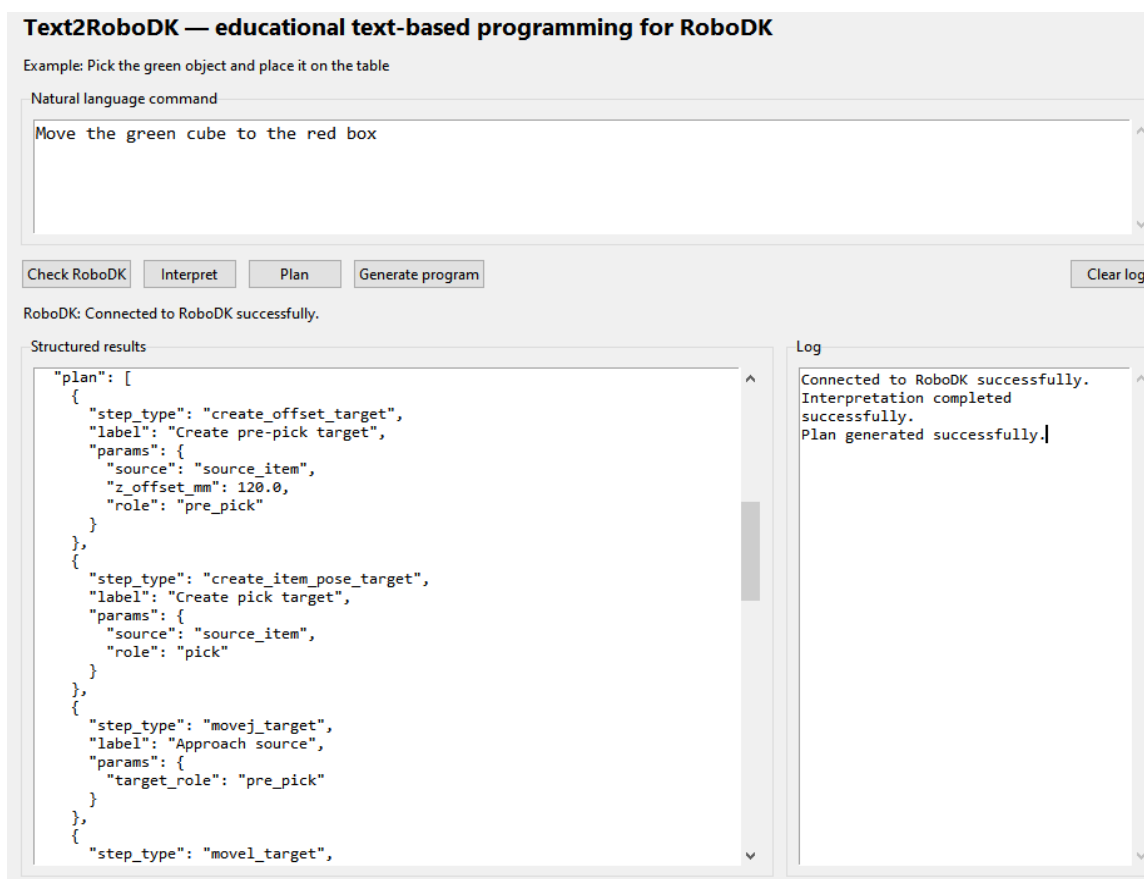


Figure 6: Program planning

In Figure 4, we can see the RoboDK station in its initial condition just before a new command is executed. Aside from the accessible items, the scenario now also includes the industrial robot, the gripper, and the reset program. Prior to interpreting the natural language command and generating any motion targets or robot programming, this graphic depicts the simulation environment.

Figure 5 shows the application's interpretation of the first test sentence that was input. By determining the primary action, manipulable object, and intended destination, the system determines the command's semantic structure at this point. Important for subsequent robot programming, this stage establishes a bridge between the user's natural language input and an internal representation of the job.

Based on the translated language, Figure 6 shows an intermediate step of the job planning process. Here, the command is seen as a structured robotic job that may be transformed into motion-related actions, rather than just words.

The chosen item will be prepared for pickup, transportation, and installation in the planning stage.

An example of a whole internal structure of the created work plan is shown at the end of the paper in Appendix 1. From initializing target locations to carrying out motion and object-handling operations, this model encompasses all the necessary stages for the pick-and-place procedure in a sequential fashion. The produced framework is an accurate representation of the natural language instruction's logical breakdown into a set of robotic operations usable inside the RoboDK framework.

There are two supplementary ways in which the program generation step occurs. To begin, the application's graphical user interface shows the created routine in textual form. Additionally, the RoboDK station automatically generates the relevant target locations, motion programs, and procedures for attaching or detaching objects. Figure 8 shows the RoboDK station after the automated development of targets and robot motion programs, whereas Figure 7 shows just a portion of the created text-based program due to its length.

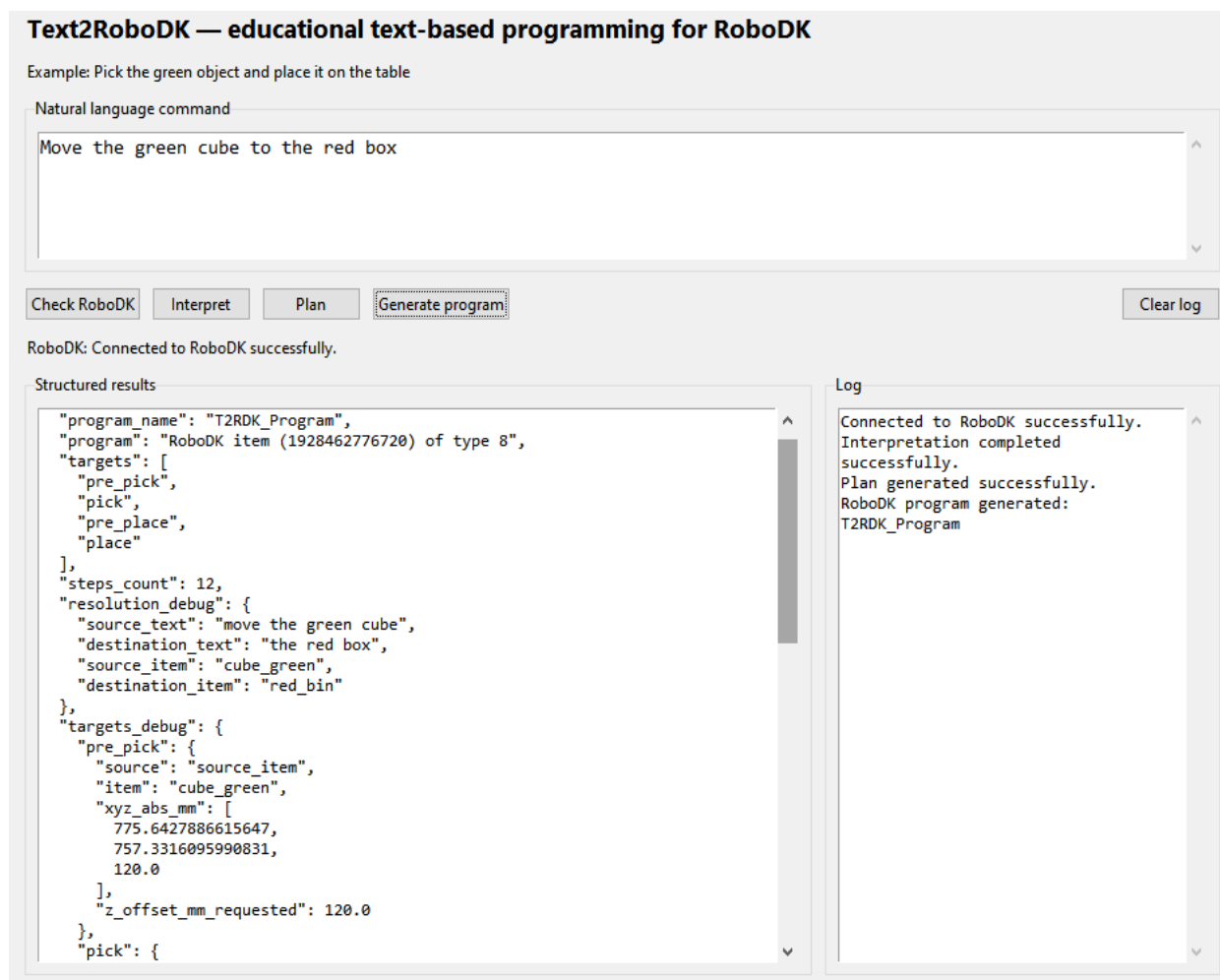


Figure 7: Textual program generation stage for off-line programming in RoboDK.

The translated natural language command produces the produced textual program, which is partially shown in Figure 8.

The output provided sheds light on the process of translating the mental representation of the job

into concrete steps for execution. This is a crucial step from an instructional standpoint as it makes the user-friendly transition from human-oriented instructions to robot-oriented instructions very clear.

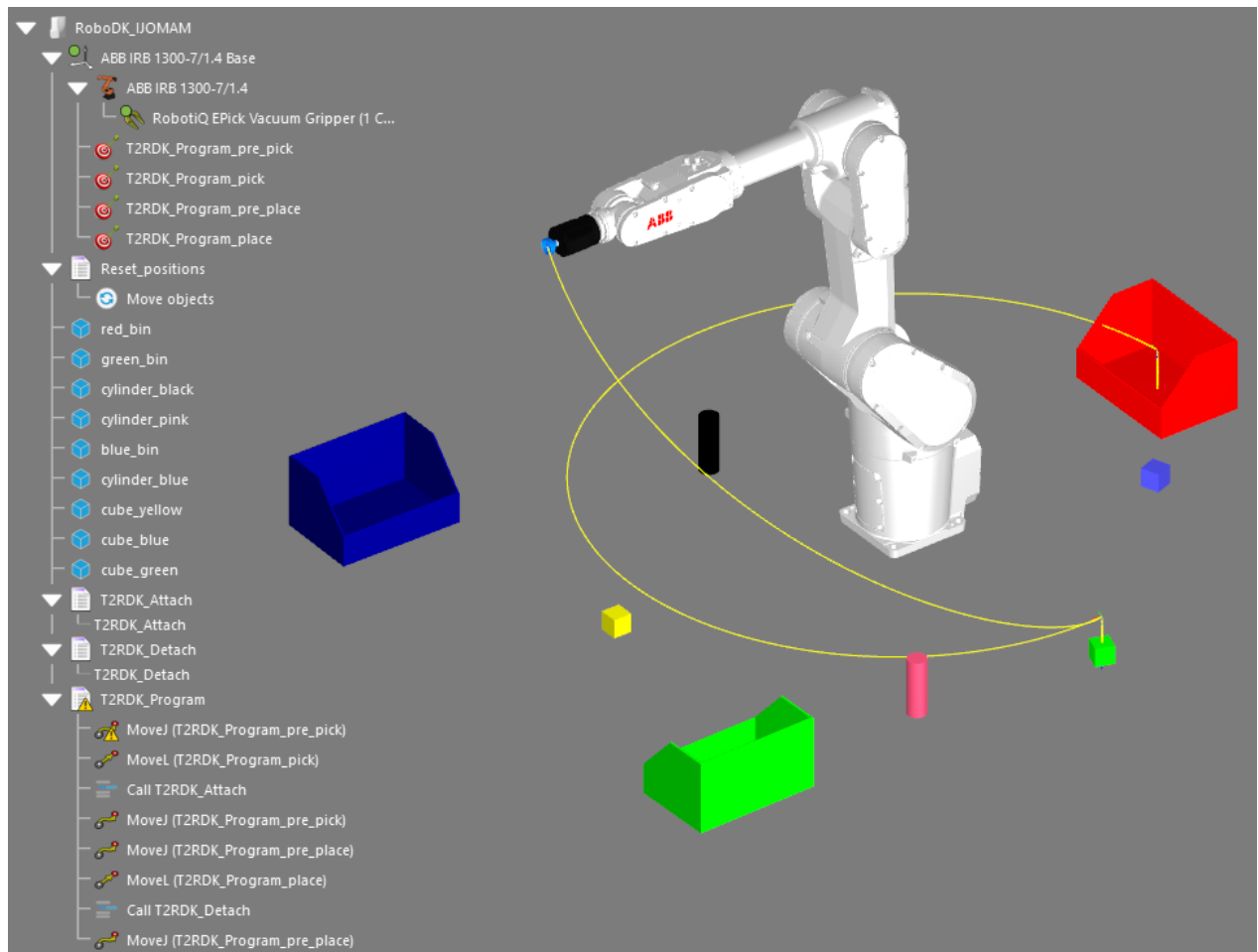


Figure 8: RoboDK station after generating targets and motion programs.

Each of the twenty test instructions followed the same processing methodology. An assessment was conducted for each example that was tested to ensure that the semantic interpretation and simulation execution were accurate. Table 1 summarizes the findings using five indicators: execution in RoboDK, object identification,

interpretation confidence (IC), action identification, and destination identification. Here, a successful execution means that RoboDK was able to produce and execute a full simulation routine, even when the initial instruction included some vagueness or missing object specifications.

Table 1. Centralized results regarding the correctness of interpretation, object and action identification, and program execution in RoboDK

NO.	NATURAL LANGUAGE COMMAND	IC	OBJECT	ACTION	DESTINATION	EXECUTION
1	Move the green cube to the red box	0.8	OK	OK	OK	OK
2	Take the blue cylinder and place it in the red bin	0.8	OK	OK	OK	OK
3	Place the green cube into the red bin	0.8	OK	OK	OK	OK
4	Put the blue cylinder in the green box	0.7	OK	OK	OK	OK
5	Pick up the red cube and move it to the blue bin	0.9	OK	OK	OK	OK

6	Grab the green cube and drop it into the red box	0.8	OK	OK	OK	OK
7	Transfer the blue cylinder to the green bin	0.2	OK	FAIL	OK	FAIL
8	Move the cube to the red container	0.6	PARTIALLY OK	OK	OK	OK
9	Take the cylinder and place it on the support	0.7	PARTIALLY OK	OK	FAIL	OK
10	Put the yellow cube inside the red box	0.9	OK	OK	OK	OK
11	Move the blue cube to the green bin	0.9	OK	OK	OK	OK
12	Take the yellow cube and place it in the blue bin	0.8	OK	OK	OK	OK
13	Put the black cylinder in the red box	0.8	OK	OK	OK	OK
14	Move the pink cylinder to the green box	0.9	OK	OK	OK	OK
15	Place the blue cube into the red bin	0.8	OK	OK	OK	OK
16	Pick up the green cube and put it in the blue box	0.9	OK	OK	OK	OK
17	Grab the blue cylinder and move it to the red bin	0.8	OK	OK	OK	OK
18	Take the yellow cube and drop it into the green bin	0.8	OK	OK	OK	OK
19	Move the cube to the blue box	0.7	PARTIALLY OK	OK	OK	OK
20	Take the cylinder and place it in the red bin	0.8	PARTIALLY OK	OK	OK	OK

Table 1 shows that the system performed very well when given specific instructions that identified the item and its final location. Nineteen out of twenty instances passed the RoboDK execution test, with nineteen correctly identifying actions and destinations and sixteen effectively identifying objects. Out of twenty instructions examined, fifteen were correctly executed when we only look at completely accurate end-to-end examples. These examples include objects, actions, destinations, and execution. These results demonstrate, under simulated situations, that the suggested software performs as expected and appropriately handles common command formats.

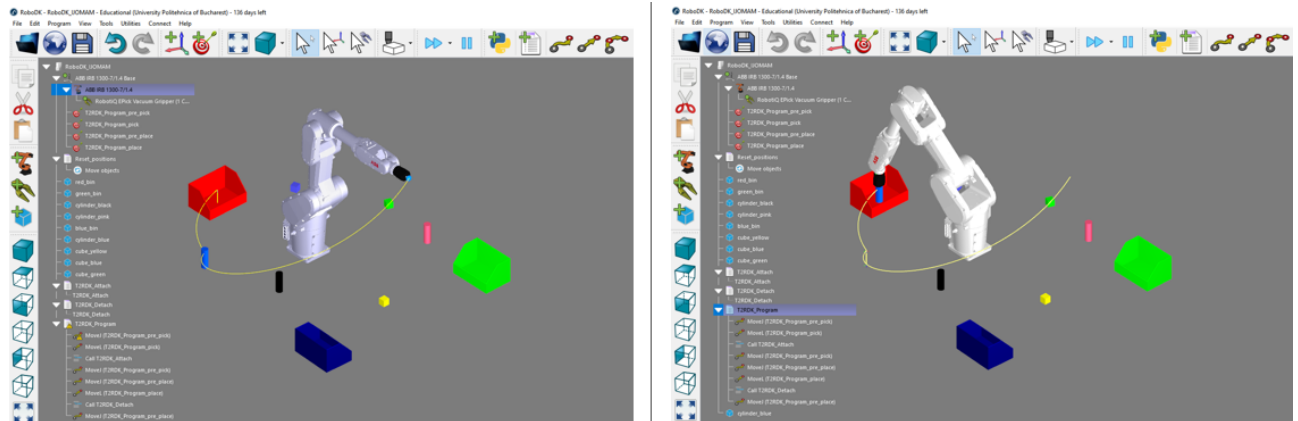
Improved interpretation confidence scores (typically between 0.8 and 0.9) were the consequence of clearly specified objects and instructions with straightforward language. Uncertain language, lacking item features, or unfamiliar terminology were linked to lower ratings. You can see the system's level of conviction in the input command's structure and completeness in the confidence score.

This examination also uncovered the existing systemic flaws. Proper result display requires the object definition to be completed. For instance, the outcomes could not be entirely precise if the words "cube" or "cylinder" were provided without any color information.

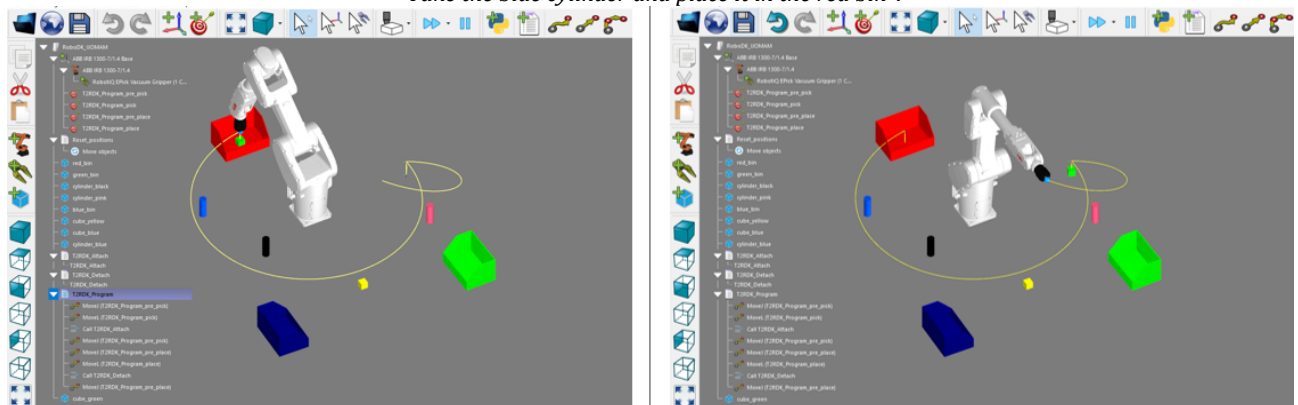
This is an example of an algorithm picking the first item from a given station that meets the requirements. On other situations, the site simply did not exist, or the synonym of the word did not adhere to the preexisting semantic frameworks. These were the causes of the reported failures or partly accurate outputs from seven, eight, nine, twentieth, and twenty commands.

Specifically, since no activity was detected, instruction 7's execution was suspended. Partial object identification was only performed by commands 8 and 19 since full cube definition was not provided. In addition to providing incomplete object descriptions in commands 9 and 20, the former also mentioned a support element that was missing from the virtual station in command 9. The system still generated an unusual trajectory for command 14, but the function itself remained executable. While this software version does a good job with controlled vocabulary inputs, it still has certain weaknesses when it comes to handling ambiguity, missing scene components, and expression modifications.

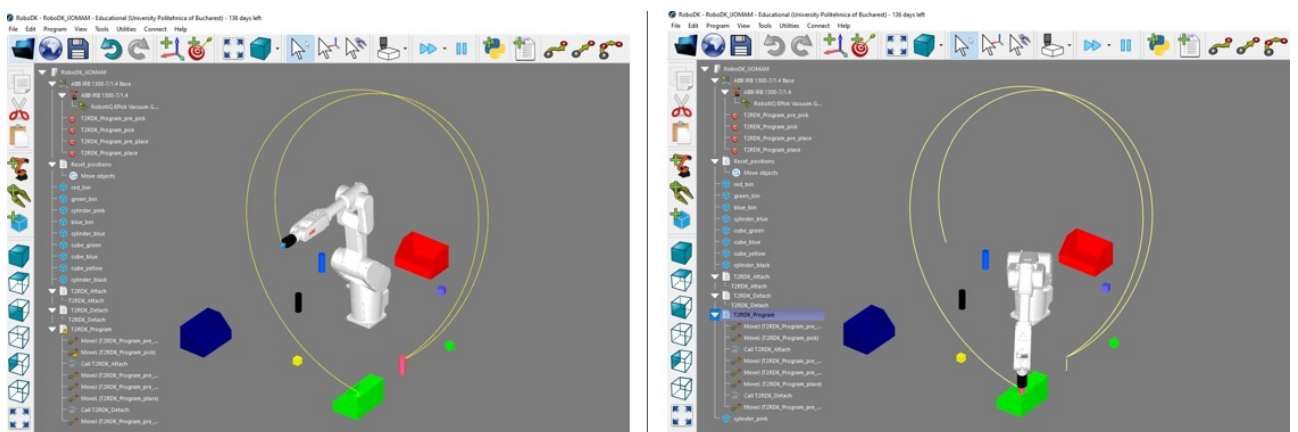
Automatically generated motion programs are shown in Figures 9–11 as examples after the interpretation of different test instructions. Presented here are the stages of the robot's setup, progress to the grasping point, transfer, and placement jobs.



*Figure 9: Automatically generated program for sentence 2
“Take the blue cylinder and place it in the red bin”.*



*Figure 10: Automatically generated program for sentence 3
“Place the green cube into the red bin”.*



*Figure 11: Automatically generated program for sentence 14
“Move the pink cylinder to the green box”.*

Figure 9 presents a representative execution sequence for sentence 2, “Take the blue cylinder and place it in the red bin”. The generated motion program shows that the system was able to identify the required object, generate the corresponding motion sequence, and execute the transfer in the RoboDK simulation environment. This example illustrates a successful case of complete interpretation and execution.

Figure 10 presents the automatically generated execution sequence for sentence 3, “Place the green

cube into the red bin”. Similar to the previous example, the image illustrates the transformation of a natural language command into a complete pick-and-place routine. The resulting simulation confirms the consistency of the proposed workflow for standard commands with clearly defined semantic elements.

Figure 11 presents the generated execution sequence for sentence 14, “Move the pink cylinder to the green box”. Although this command was executed successfully, the resulting robot path was

less regular than in the other representative cases. The resulting routines are simulation-valid, but they aren't optimized for trajectory quality, route smoothness, or motion efficiency and this is a major weakness of the system that this finding brings to light.

Experiments show that the program can understand and carry out basic pick-and-place activities with specific terminology in controlled RoboDK circumstances, as well as other natural language instructions. Concurrently, the results demonstrate that the current system can only operate with a small set of instructions and a restricted vocabulary. We conclude that the findings show promise for implementing the suggested strategy in schools. In addition, they lay out specific steps on how to improve robust motion planning, semantic flexibility, and ambiguity management in the future.

4. Discussion

Offline natural language programming of industrial robots is now a reality, thanks to the findings of the suggested method's effectiveness in the RoboDK environment. This capability is particularly useful in supervised learning environments. Students and novice users who aren't acquainted with conventional robot programming approaches will find the software's description of basic pick-and-place tasks to be easy to understand and apply [1,3]. This is one way the system may be seen as a helpful teaching tool; it establishes an obvious and direct link between the input of spoken language and the execution of robot activities.

The experimental validation revealed an important finding: the system's performance is greatly affected by the clarity of the orders. The accuracy and reliability of the results were enhanced when the object, action, and destination were clearly stated in the commands. Missing information or less common wording in the commands caused them to be either partially or inaccurately executed. In a controlled vocabulary context with an evident and visible task structure, the present software version works best, according to these data [4-7].

The results also show that the application's modular structure is an asset. To help users better comprehend how to translate textual commands into robotic operations, the system provides a domain-specific framework for natural language processing, semantic mapping, intermediate task representation, and RoboDK program generation. This is very beneficial in academic settings, where comprehending the process may be just as crucial as having a completed output. Furthermore, the modular architecture lays the groundwork for future enhancements by including more adaptable

semantic processing and more sophisticated ways of language interpretation [8-10].

The produced motion of the robot is another critical component. Using the appropriate coordinate systems of the RoboDK scene pieces, this method automatically creates the target points. This technique doesn't have any technical ways to optimize trajectories, smooth motion, or avoid obstructions, but it works well for instructional presentations and proof-of-concept validation. This means the produced routines are good enough for simulations, but they aren't optimized for things like motion quality or industrial efficiency in the actual world [11,12].

Presently, the system can only do basic pick-and-place tasks and can only execute instructions sequentially. There is currently no way to execute complicated task sequences, plan several tasks at once, or do sophisticated reasoning that requires process coordination. Similarly, since the semantic interpretation step relies on preset lexical structures, problems like as undefined destination names, underspecified objects, and unknown synonyms may significantly affect the program's performance. While these caveats do reduce the paper's use, they are reasonable given that the purpose is not to provide a comprehensive solution to industrial programming but rather to show that a natural language framework for instruction may function [13-19].

As a result, two key things are clarified by the conversation. In the beginning, the software shows that natural language may be a good interface for offline robot programming inside the RoboDK environment. The results also demonstrate that there is space for improvement in terms of resilience, execution quality, and semantic flexibility. Additional language support, better handling of ambiguity, stronger support for longer command sequences, and the incorporation of more sophisticated AI methods for command interpretation and robotic job generation may be the focus of future development [20].

5. Conclusions

A Python-based instructional tool that allows students to program industrial robots in the RoboDK environment using just natural language instructions was featured in this article. With this method, developers could take English-language descriptions of basic pick-and-place jobs and turn them into robot programming. With its simplified programming interface, this software is perfect for educational institutions or any other setting where students or new users could benefit from a more direct correlation between task requirements and robot performance.

The suggested system comprehended and carried out the majority of typical instructions during experimental validation in a controlled RoboDK virtual station when given explicit criteria for objects, actions, and destinations. In addition to demonstrating that the method is feasible, our findings also suggest that natural language might serve as an effective interface for controlled offline robot programming. On the other hand, the study did point out a few present problems with the system, such how it doesn't handle cases when there are unsupported words, missing scene pieces, or objects with unclear or incomplete specifications.

Rather than being an all-inclusive solution for industrial programming, the present application version serves more as a teaching framework and proof of concept. Now, it can only do basic tasks like pick-and-place, processing with a single command, and finding meaning based on rules. There are problems with the derived trajectories' smoothness of motion, route planning, and obstacle avoidance as well. Even in more complex robotic programming situations, the system's modular design offers a robust foundation for development and refinement [11,12].

Possible future approaches for the semantic interpretation module include a wider vocabulary, improved support for more complicated action sequences, and more resilience to fluctuations in spoken language. Improving adaptability and decreasing reliance on predetermined rules may be achieved by using strong AI technologies, including NLP models based on machine learning or huge language models. However, the system's potential for more sophisticated applications may be enhanced by integrating trajectory planning and obstacle avoidance techniques. Improving tools for human-robot interaction and showing the potential of robot programming driven by natural language are two of the study's aims [20].

Acknowledgement

This work has been supported by (1) the CERMISO Center (Project Contract No. 159/2017, Program POC-A.1-A.1.1.1-F); (2) Research Program Nucleu within the National Research Development and Innovation Plan 2022–2027, carried out with the support of MCID (project No. PN 23 43 04 01); (3) the Support Center for International RDI Projects in Mechatronics and Cyber Mix-Mechatronics (Contract No. 323/22.09.2020; the project co-financed by the European Regional Development Fund through the Competitiveness Operational Program (POC) and the national budget); and (4) ERASMUS-EDU-2023-EUR-UNIV, Project 101124676 — EELISA, funded by the European Union, <https://eelisa.eu/>.

Disclaimer

Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

References

- [1] Buzducea, C.-A.; Drăgoi, M.-V.; Cristoiu, C.; Puiu, R.-A.; Puiu, M.; Petrea, G.; Navligu, B.-C. Machine Learning in Education: Predicting Student Performance and Guiding Institutional Decisions. *Educ. Sci.* 2026, 16, 76.
<https://doi.org/10.3390/educsci16010076>
- [2] Drăgoi, M.-V.; Roxana P.; Gabriel P.; Catalin B.; Spiridon-Mocioaca T.-M.; Mihail P.; Suciu C.P. IMPROVING THE LEARNING PROCESS FOR STUDENTS. PERSONALIZING LEARNING THROUGH SOFTWARE APPLICATIONS. *UPB Scientific Bulletin, C Series* 1/2025.
- [3] Drăgoi, M.-V.; Nisipeanu, I.; Puiu, R.-A.; Tache, F.-G.; Spiridon-Mocioacă, T.-M.; Hank, A.; Cristoiu, C. An Inclusive Offline Learning Platform Integrating Gesture Recognition and Local AI Models. *Biomimetics* 2025, 10, 693.
<https://doi.org/10.3390/biomimetics10100693>
- [4] Tellex S.; Knepper R.A.; Li A.; Rus D.; Roy N.; Asking for Help Using Inverse Semantics. *Conference: Robotics: Science and Systems* 2014.
<https://doi.org/10.15607/RSS.2014.X.024>
- [5] Matuszek, C.; FitzGerald, N.; Zettlemoyer, L.; Bo, L.; Fox, D. A Joint Model of Language and Perception for Grounded Attribute Learning. *International Conference on Machine Learning (ICML)*, 2012.
<https://doi.org/10.48550/arXiv.1206.6423>
- [6] Chen, D.L.; Mooney, R.J. Learning to Interpret Natural Language Navigation Instructions from Observations. *AAAI Conference on Artificial Intelligence*, 2011.
<https://doi.org/10.1609/aaai.v25i1.7997>
- [7] Kollar, T.; Tellex, S.; Roy, N. Toward Understanding Natural Language Directions. *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, 2010.
<https://doi.org/10.1145/1734454.1734473>
- [8] Shridhar, M.; Manuelli, L.; Fox, D. CLIPort: What and Where Pathways for Robotic Manipulation. *Conference on Robot Learning (CoRL)*, 2022.
<https://doi.org/10.48550/arXiv.2109.12098>
- [9] Brohan, A.; Brown, N.; Carbajal, J.; et al. RT-1: Robotics Transformer for Real-World

- Control at Scale.
Science Robotics, 2023, 8(84), eadh5087.
<https://doi.org/10.48550/arXiv.2212.06817>
- [10] Ahn, M.; Brohan, A.; Brown, N. et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances.
<https://doi.org/10.48550/arXiv.2204.01691>
- [11] Cristoiu, C.A.; Drăgoi, M.-V.; Nechita, R.-M.; Stoian, B.N.; Dudici, C.L.; Puiu, R.-A.; Petrea, G. The Impact of Geometric Continuities (C1, C2, and C3) on the Trajectories of Industrial Robots. *Technologies* 2026, 14, 176.
<https://doi.org/10.3390/technologies14030176>
- [12] Cristoiu, C.A.; Drăgoi, M.-V.; Ivan, A.M.; Nechita, R.-M.; Grecu, I.; Puiu, R.-A.; Petrea, G.; Emilia, P. Optimization Method Based on the Minimum Action Principle for Trajectory Length of Articulated Manipulators. *Technologies* 2025, 13, 490.
<https://doi.org/10.3390/technologies13110490>
- [13] Talinga, A.-M.; Hadăr, A.; Drăgoi, M.-V.; Nisipeanu, I.; Haider, A.A.; Suciu C.P. YOLO-v8 IN CAPTURING IMPERFECTIONS GENERATED BY CHANGING 3D PRINTER PARAMETERS. *UPB Scientific Bulletin, C Series* 4/2024.
- [14] Drăgoi, M.-V.; Nisipeanu, I.; Frimu, A.; Tălîngă, A.-M.; Hadăr, A.; Dobrescu, T.G.; Suciu, C.P.; Manea, A.R. Real-Time Home Automation System Using BCI Technology. *Biomimetics* 2024, 9, 594.
<https://doi.org/10.3390/biomimetics9100594>
- [15] Drăgoi, M.-V.; Hadăr, A.; Goga, I.; Grigore, L.S.; Amado, S.; Haider, A.A.; Design and implementation of an EEG-based BCI prosthetic lower limb using Raspberry Pi 4. *UPB Scientific Bulletin, C Series* 3/2023.
- [16] Drăgoi, M.-V.; Hadăr, A.; Goga, N.; Baci, F.; Ștefan, A.; Grigore, L.Ș.; Gorgoteanu, D.; Molder, C.; Oncioiu, I. Contributions to the Dynamic Regime Behavior of a Bionic Leg Prosthesis. *Biomimetics* 2023, 8, 414.
<https://doi.org/10.3390/biomimetics8050414>
- [17] Haider A.A.; Goga N.; Vasilateanu A.; Muslim M.A.; Khalid A.A.; Drăgoi, M.-V.; English and Romanian Brain-to-Text Brain-Computer Interface Word Prediction System. *International Journal of Advanced Computer Science and Applications (IJACSA)* 13.10 (2022).
<http://dx.doi.org/10.14569/IJACSA.2022.0131003>
- [18] Drăgoi, M.-V.; Maher R.; Alaa A.A.M.H.A.; Jiga G.; Piu R.; Petrea G.; Hank A. COMPARATIVE ANALYSIS OF INTERPOLATION AND EXTRAPOLATION METHODS IN REGRESSION MODELS. *UPB Scientific Bulletin, C Series* 1/2026.
- [19] Drăgoi, M.-V.; Frimu, A.-V.; Postelnicu, A.; Puiu, R.-A.; Petrea, G.; Hank, A. Interactive Teleoperation of an Articulated Robotic Arm Using Vision-Based Human Hand Tracking. *Biomimetics* 2026, 11, 151.
<https://doi.org/10.3390/biomimetics11020151>
- [20] Varghese, J.R.; Berthet-Rayne, P.; Giataganas P.; Vitiello, V.; Yang, G.Z. A framework for sensorless and autonomous probe-tissue contact management in robotic endoscopic scanning. *IEEE Transactions on Instrumentation and Measurement*
<https://doi.org/10.1109/ICRA.2017.7989205>

Appendix 1 Example of entire work plan code structure.

```
{
  "parsed": {
    "intent": "pick_place",
    "source_object": {
      "object_type": "cube",
      "color": "green",
      "raw_text": "move the green cube"
    },
    "destination": {
      "destination_type": "bin",
      "color": "red",
      "raw_text": "the red box"
    },
    "raw_text": "Move the green cube to the red box",
    "confidence": 0.85,
    "debug": {
      "normalized": "move the green cube to the red box",
      "source_fragment": "move the green cube",
      "destination_fragment": "the red box",
      "source_color": "green",
      "source_type": "cube",
      "destination_type": "bin",
      "destination_color": "red"
    }
  },
  "resolution": {
    "source_text": "move the green cube",
    "destination_text": "the red box",
    "source_item": "cube_green",
    "destination_item": "red_bin"
  },
  "plan": [
    {
      "step_type": "create_offset_target",
      "label": "Create pre-pick target",
      "params": {
        "source": "source_item",
        "z_offset_mm": 120.0,
        "role": "pre_pick"
      }
    },
    {
      "step_type": "create_item_pose_target",
      "label": "Create pick target",
```

```

"params": {
  "source": "source_item",
  "role": "pick"
},
{
  "step_type": "movej_target",
  "label": "Approach source",
  "params": {
    "target_role": "pre_pick"
  }
},
{
  "step_type": "movel_target",
  "label": "Move to pick",
  "params": {
    "target_role": "pick"
  }
},
{
  "step_type": "attach",
  "label": "Attach object",
  "params": {}
},
{
  "step_type": "movej_target",
  "label": "Retreat from source",
  "params": {
    "target_role": "pre_pick"
  }
},
{
  "step_type": "create_offset_target",
  "label": "Create pre-place target",
  "params": {
    "source": "destination_item",
    "z_offset_mm": 120.0,
    "role": "pre_place"
  }
},
{
  "step_type": "create_item_pose_target",
  "label": "Create place target",
  "params": {
    "source": "destination_item",
    "role": "place"
  }
},
{
  "step_type": "movej_target",
  "label": "Approach destination",
  "params": {
    "target_role": "pre_place"
  }
},
{
  "step_type": "movel_target",
  "label": "Move to place",
  "params": {
    "target_role": "place"
  }
},
{
  "step_type": "detach",
  "label": "Release object",
  "params": {}
},
{
  "step_type": "movej_target",
  "label": "Retreat from destination",
  "params": {
    "target_role": "pre_place"
  }
},
],
"metadata": {
  "intent": "pick_place",
  "source": "cube_green",
  "destination": "red_bin"
}

```