A Smart IDE for Robotics Research

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India.
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Vice-Chancellor, Srinivas University, Mangalore, India.
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area/Section: Computer Science.
Type of the Paper: Simulation-based Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed in: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.6781577
Google Scholar Citation: IJMTS

How to Cite this Paper:
Chakraborty, Sudip, & Aithal, P. S. (2022). A Smart IDE for Robotics Research. *International Journal of Management, Technology, and Social Sciences (IJMTS)*, *7*(1), 513-519. DOI: https://doi.org/10.5281/zenodo.6781577

International Journal of Management, Technology, and Social Sciences (IJMTS)
A Refereed International Journal of Srinivas University, India.

CrossRef DOI: https://doi.org/10.47992/IJMTS.2581.6012.0205

Received on: 08/06/2022
Published on: 30/06/2022

© With Authors.

This work is licensed under a Creative Commons Attribution-Non-Commercial 4.0 International License subject to proper citation to the publication source of the work.
Disclaimer: The scholarly papers as reviewed and published by the Srinivas Publications (S.P.), India are the views and opinions of their respective authors and are not the views or opinions of the SP. The SP disclaims of any harm or loss caused due to the published content to any party.

A Smart IDE for Robotics Research

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India.

OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Vice-Chancellor, Srinivas University, Mangalore, India.

OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: *We see the robot is engaged in every task, from the hazardous industrial environment to the floor cleaning. We are benefited directly or indirectly from robots in our everyday life. This is the untired efforts of many Robot researchers who play a significant role behind the scenes. They pass lots of tests before releasing the robot for humans. Most of the test procedure is to send the command from the IDE to the robot. The researcher generally feels two types of constraints. One is various IDE. They spend lots of time finding the tool's location for different robot vendors with their IDE. Another one is, sometimes the proprietary robot IDE is not too fit for robot researchers. They need some customizable IDE. But in the close source application is not possible to customization. Here we described a procedure so that our two constraints can be solved. We developed a smart robotic IDE that can operate various robots, introducing a hardware abstraction layer. This IDE can also be configured in every aspect. It is an open-source application. The robot researcher can easily customize it according to their need.*

Design/Methodology/Approach: *We create a GUI application in C# inside the visual studio community edition 2022. The main UI thread is integrated with file handling, programming interface, and various robot object. Here we created a couple of objects. In this way, we can make any new robot object. The object is exposed on the GUI element. Selecting any robot, we can send the available command through the terminal or programming interface.*

Findings/Result: *Sometimes, robot researchers must break some impenetrable barrier for their research needs. But need lots of effort to develop some custom application to interact with robot or automated device. Through this research work, we provide helpful information so they can integrate their robot easily. Compared to other available open-source or closed-source applications, this application will take less time to debug and solve the issue at the earliest.*

Originality/Value: *The smart IDE makes research work better and faster. We need complete freedom to interact with robots or automated devices. The industry standard robots have their IDE to interact with their robot. Sometimes times more flexibility makes work easy. In this scenario, our approach can provide us with better flexibility to work. By following this procedure, the researcher can get some unique benefits for their research work.*

Paper Type: *Simulation-based Research.*

Keywords: Robot IDE, Robot Simulator, Robot Programming.

1. INTRODUCTION :

We use various types of IDE to program or run the different vendor robots. Most of the time, we use vendor-provided IDE for their robot. The native IDE has excellent features and flexibility to use. The IDE needs to change when we employ another robot for our research. The new IDE needs to be learned before using it. So, the scenario is the variety of robots use multiple native IDE to understand. It takes lots of time. We must give more time to learning the IDE command tools instead of doing our research. Sometimes we defocussed on our research work. It would be great if one IDE would work for the entire automation device or robot family. i.e., Learn once, use all times model.

For this purpose, this is our research work. Here we tried to provide that effort. This is the initiation phase. And it can be significant for others' contributions. Here we keep the UI elements identical for all robots or automated devices. The robot object is different; that is code level, not user level. This

approach gives us great flexibility to learn. When we select the robot, the command list is automatically populated into the command line or combo box. There is no need to memorize the available command for the robot.

Using this IDE has lots of benefits. Due to the same IDE for all robots, its usability is straightforward. It will take just one time to learn. When we use the robot not in the list, studying the respective robot's API, we need to create the robot object and keep the same command format as we used before. Our robot object is bridged between our user interface and vendor-provided API. It gives us great flexibility. The commands are populated on the command lines. It is also a one-time effort. Nowadays, the Open source community is a great strength. We keep it Open source. Community contributions might be made as significant as possible.

2. RELATED WORKS :

Beate Jost et al. developed a robotic framework named Roberta. It has a feature to assist lectures and teachers directly with workshops and practical classes. It is a fully online, web-based alternative for visual educational robot programming. It allows the browser to program without manual installation and preparation with an easy connection with different robot hardware [1]. K.Nishiwaki et al. have developed a humanoid robot, "H6," as a platform for researching perception-action coupling in the intelligent behavior of humanoid-type robots. It has the features like enough DOFs, and each joint has enough torque for full-body motion. PC/AT compatible onboard computer which RT-Linux controls from low-level to high-level control is achieved simultaneously [2]. Stephen Hart et al. introduce the Robot Task Commander (RTC) framework for defining, developing, and deploying robot application software for different run-time contexts. An expert developer can implement a new application with a combination of scripts, called process nodes and state machines that set the robot's control mode. A non-expert developer can assemble process nodes and controller state machines into novel hierarchical applications using a visual programming language (VPL) [3]. Soohee Han, Mi-sook Kim, and Hong Seong Park have developed an IDE to program the Robotic Services efficiently. It covers controlling hardware (HW) devices to execute complicated application programs. Based on available software (SW) architecture, standardized components with design patterns, frameworks, and dedicated servers for developing robot SW applications smoothly and efficiently [4]. Dennis Stampfer et al., in their paper, argue that there is a lack of approaches addressing the overall integration challenge (i.e., systematic composition) in service robotics, and efforts for integration are underestimated. To address this, they approach one of the recent trends in service robotics: Model-Driven Software Development (MDSD) [5]. In their paper, Paulo A. F et al. present a low-cost, low-cost platform for swarm robotics, easy to assemble using off-the-shelf components and deeply integrated with the most used robotic framework available today: ROS (Robot Operating System) [6]. In their paper, Ankur M. Mehta et al. present a procedure to create a complete and easily customizable general-purpose control system for miniature robotic systems, particularly micro air vehicles. In its default configuration, hardware designs, firmware, and software are available to deliver an out-of-the-box robot control [7]. In their paper, Gregory F. Rossano et al. primarily consider path creation and programming of robotic systems' logic (i.e., decision-making) aspects. This will allow us to make robot programming easier for someone who already understands them [8]. Carlos Mateo et al., in their paper, present a novel tablet-based end-user interface for industrial robot programmers). This application makes it easier to program industrial robots for polishing, milling, or grinding [9]. In their paper, Luke Gumbley and Bruce A MacDonald present a new Integrated Development Environment (IDE) target explicitly in Robotics programming. The IDE is based on IBM's Eclipse platform and supports Python [10]. Robert S. Breznak developed a unified system to build robotics, and other cyber-physical methods are discussed. The suitable system includes a platform that integrates the processing of actuators, sensors, and other modules [11].

3. OBJECTIVES :

The objective of the research work is to provide some helpful information about smart robotic IDE. We use a different IDE for different vendors' robots. The learning curve is high in learning other robots' IDE. So here, we developed an intelligent featured robot IDE, which means we will understand only one IDE for all robots. Using a new robot will not take much more time, and migration to any other vendor's robot will be a little easier. Through our research, we demonstrated how we could do that work possible.

4. APPROACH AND METHODOLOGY :

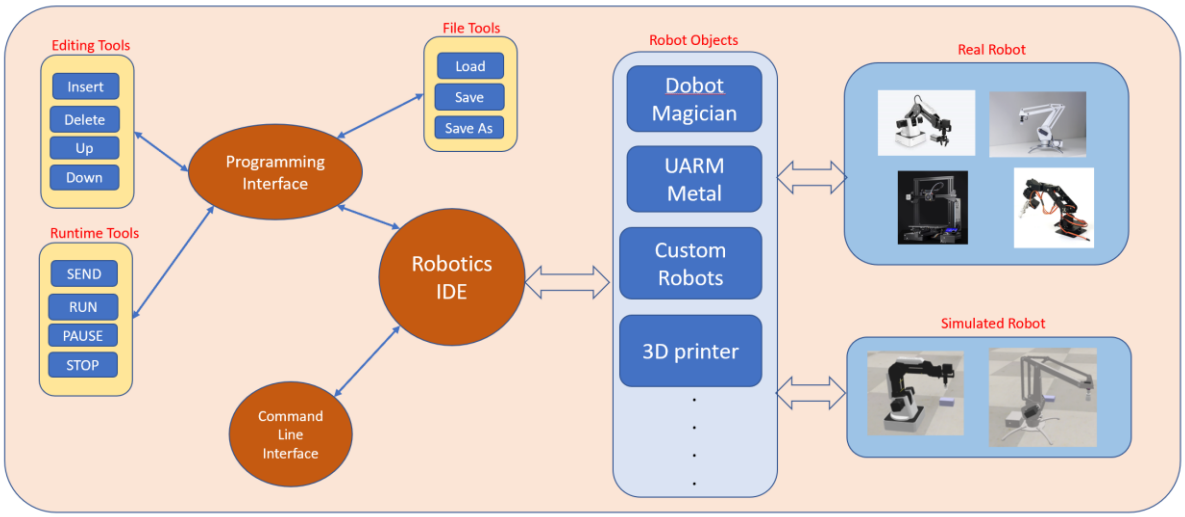

Fig. 1: Block diagram of the complete system

Figure 1 depicts the block diagram of our complete system. The C# UI thread is the heart of the entire system. It is connected with three sub-systems: programming interface, command-line interface, and robotic object. The programming interface has three extensions. Let us know a few details on these:

Editing tools: The editing tools help to edit the programming we are writing inside the programming interface. The "Insert" command inserts a blank on the selected row. The "Delete" button is used to delete the selected line. The "Up" moves the selected line one row up. And opposite, the "Down" button is used to down the selected row. These tools are disabled when the program is running. To edit the program using these tools, we need to stop and enable these buttons to edit.

Run-time tools: Using these tools, we control the program. The "SEND" command sends a command to the robot. The "RUN" command runs the available command sequence. The "Pause" is used to pause the program for a while. From pause to run does not change the execution line number. But if we press the "stop" button, the program will stop, and the running program line will reset to 0. When it starts, it will begin from the start, i.e., the top first line.

File Tools: The file tools are used to handle program files. The "load" button loads the program from the saved file on the disk. If we press the "save" button, it will keep the current changes to the disk with the same name. And "save as" is used to save the file using another name. The running file saves automatically when the application is closed. The file path will also be saved. If the file path is not found, the program space keeps empty. The file path is located in the setting.json, which can be found inside the working directory. All settings are available in this file in text format. The files are not encrypted for external modification.

Command-line interface: The command-line interface is used basically for experiment purposes. We can send particular command to the robot with different parameters to see the result. If we satisfy the result, we can send it to the programming interface from the history list. Every command we send to the robot through the command line is also saved to the history list. We can also send the command from the history list to the programming interface. Click the line, and press the enter key. The command will add to the programming area.

Robot object: In the robot object sub-system, we created each object for each robot. For the new robot, we need to add it here. In most scenarios, every branded robot has an API list. Seeing the API, we can create an object with the most familiar command. Here we created a low-cost robot, Dobot magician, UARM-like robot. The main thread is connected with the actual object or simulated object. The command can send to the simulator or the real robot through the serial port or TCP/IP, which is available for their interface. The CoppeliaSim has some robot models. We can experiment and integrate with them without purchasing them.

5. EXPERIMENT :

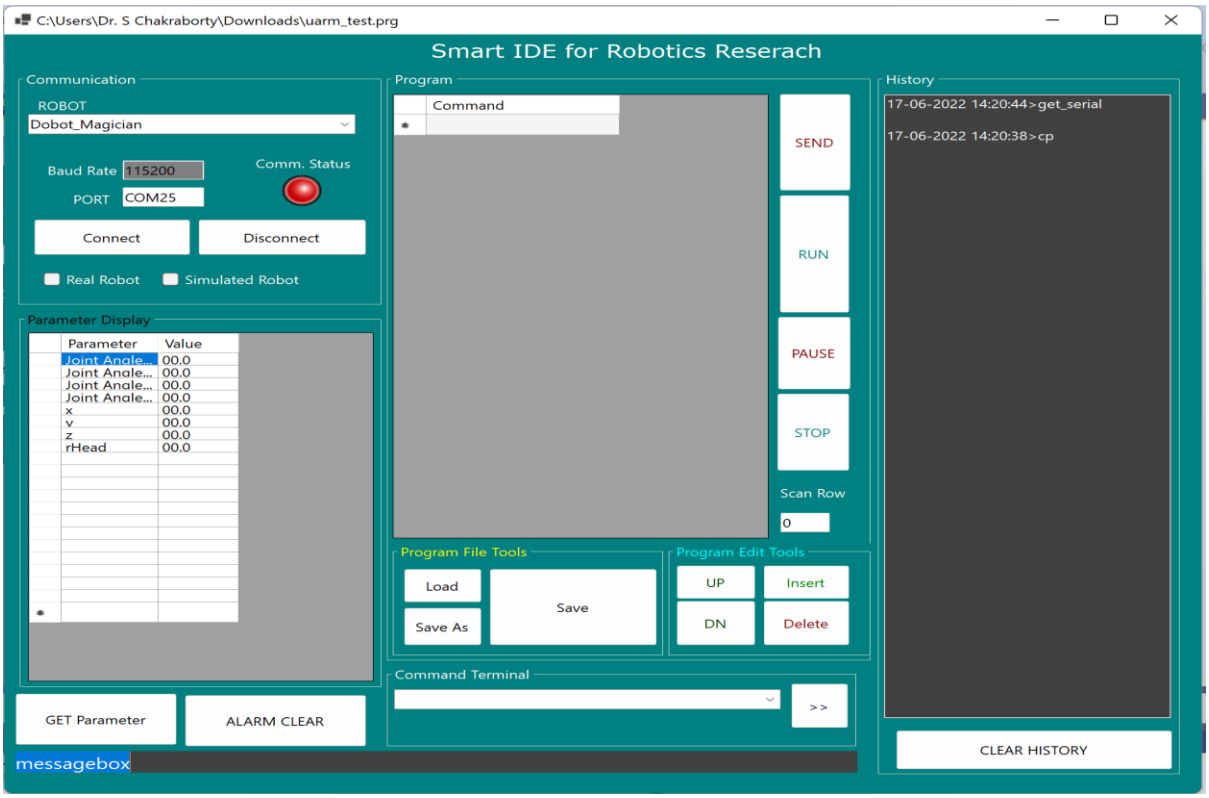

Fig. 2: The application User Interface.

Now let's do some experiments. We need to follow the below steps:
Download project: Download the project from Github. The link is available under the recommended section. Unzip the project. Go to bin\debig folder. Run the exe file. We will see an application in figure 2.

Connect the robot: If we have the robot already present inside the combo box, then it will be good. If not present, we have to create the particular robot object. Assume the robot is available on our list. Then connect with the PC/Laptop. From the windows device manager, notes the assigned serial port. Under the communication group box, add the given PORT name into the "PORT" text box.

Communication: under the communication category, select the robot. Press the "connect" button. If Green LED glows, the communication port has been created successfully. Red LED indicates the port is not available for communication. We need to check whether another opened application already uses the port or if other issues are present. The problem may be system-level or device-level.

Command Send: There are two ways to program or move the robot. The command is loaded and associated with the robot object when selecting the robot. We will see the command list if we click the combo box in the command terminal window. Click any command and set the parameter, and press the enter key. It will send to the robot. With sending to the robot, it will send to the history list.

Programming Interface: Selecting the row, we can type the command. It is also possible to bring the command from the history list. Select the line in the history list and press enter. It will come to the programming area. The last command must be the "end." Fetching the "end" command, the program pointer will go to the first line; else, it might produce an error. By modifying the source code, we can also change the terminal command.

History List: The history list is used to see the past command. Every command is displayed with a time stamp. The Last Command is pushed at the first line. The history is also saved when an application is going to close. We can clear the history by pressing the "CLEAR HISTORY" button.
That's all.

These are the main flow of the application. If not going as expected, we need to debug using breakpoint. In that scenario, some programming knowledge is required to modify or add some functionality to the application.

6. RECOMMENDATIONS :

- The project file is available from https://github.com/sudipchakraborty/Smart_Robotics_IDE
- The visual studio community edition is available from https://visualstudio.microsoft.com/vs/community/
- The entire project provides information on creating an intelligent IDE for robotics research. Lots of scopes are available for bug fixing and improvements.
- We added the robot object to our need basics. The more robots object can be added using the same procedure.
- For a new robot implementation, study the API of the respective robot. If they provide a dynamic link library (DLL), that will be great; else need to create from scratch.
- There is scope through a command-line interface like camera access, feedback sense, and trigger some output.
- We tested the interface using the Dobot Magician and UARM Metal robot. For other robots, it might create a little bit issue.
- We initiated the journey. The community can make it extensive, adding functionality and more robot objects.

7. CONCLUSION :

We demonstrated how to integrate most robot operations into a single intelligent robotic IDE. It has lots of advantages. The learning curve is a little bit. It is easy to send commands to the robot. To test and debug is made easy. We can also customize the IDE to our needs. It is also possible to add a command that is easy to understand on a specific application, leading to more readable and understandable.

REFERENCES :

[1] Beate Jost et al. (2014). Graphical Programming Environments for Educational Robots: Open Roberta - Yet another One?. 2014 IEEE International Symposium on Multimedia, 381-386. Google Scholar↗

[2] Nishiwaki, K., Sugihara, T., Kagami, S., Kanehiro, F., Inaba, M., Inoue, H. (2000). Design and Development of Research Platform for Perception-Action Integration in Humanoid Robot: H6. Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 0-7803-6348-5/, 1559-1564. Google Scholar↗

[3] Stephen Hart, Paul Dinh, John D. Yamokoski, Brian Wightman, Nicolaus Radford, (2014). Robot Task Commander: A Framework and IDE for Robot Application Development. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014) September 14-18, Chicago, IL, USA, 1547-1554. Google Scholar↗

[4] Soohee Han, Mi-sook Kim, and Hong Seong Park. Open Software Platform for Robotic Services. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, 9*(3), JULY 2012. 467-481. Google Scholar↗

[5] Dennis Stampfer, Alex Lotz, Matthias Lutz, Christian Schlegel. The Smart MDSD Toolchain: An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software. *Journal of Software Engineering for Robotics. Journal of Software Engineering for Robotics 7*(1), July 2016. 3-19. Google Scholar↗

[6] Paulo A. F., Rezeck, Hector Azpurua, Luiz Chaimowicz (2017). HeRo: An Open Platform for Robotics Research and Education. pp 1-6. 978-1-5386-0956-9/17. Google Scholar↗

[7] Ankur M. Mehta, Kristofer S. J. Pister. (2010). WARPING: A complete open-source control platform for miniature robots. The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems October 18-22, Taipei, Taiwan. 5169-5174. Google Scholar↗

[8] Gregory F. Rossano, Carlos Martinez, Mikael Hedelind, Steve Murphy, and Thomas A. Fuhlbrigge. (2013). Easy Robot Programming Concepts: An Industrial Perspective. 2013 IEEE International Conference on Automation Science and Engineering (CASE). 978-1-4799-1515-6/13. 1119-1126. Google Scholar↗

[9] Carlos Mateo, Alberto Brunete, Ernesto Gambao, Miguel Hernando. (2014). Hammer: An Android Based Application for End-User Industrial Robot Programming. 978-1-4799-2280-2(2014) IEEE. 1-6. Google Scholar↗

[10] Luke Gumbley and Bruce A MacDonald. (2005). Development of an Integrated Robotic Programming Environment. Department of Electrical and Computer Engineering University of Auckland, 1-8. Google Scholar↗

[11] Robert S. Breznak, Somerville, M. A. (US); Alexander V. Camilo, Worcester, M. A. (US); Kevin J. Harrington-Rutter, Worcester, M. A. (US). (2011). Development Platform for Robotic Systems. The United States Patent Application Publication. Pub. No.: US 2011/0224828A1. Pub. Date: Sep. 15, 2011. Google Patents↗

****