2024 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)
July 15-19, 2024. Boston, MA, USA

Optimizing Small-Scale Commercial Automation: Introducing WOS, a Low-Code Solution for Robotic Arms Integration

Yue Feng, Weicheng Huang, I-Ming Chen, Fellow, IEEE/ASME

Abstract - **Amidst escalating labor costs and the imperative for workplace safety, automation has become a crucial trend. Yet, the substantial expenses and technical complexities of robotic systems, demanding significant time and expertise for design and deployment, limit their adoption in small enterprises. To tackle this, we present the WinGs Operating Studio (WOS) – a novel low-code platform for robotic arm operations. WOS stands out by effortlessly integrating with a wide range of robotic arms and accessories, including various sensors and end effectors, through flowchart programming and versatile APIs. This facilitates straightforward implementation of advanced features like multi-robot cooperation and external system interactions. The paper delves into WOS's design, capabilities, and architecture, highlighting its role in lowering technical barriers and operational costs. Performance evaluations on ARM-based Single-Board Computers and real-world scenarios, such as automated coffee making with dual robotic arms and VR controlled spray painting, underscore WOS's potential to empower small businesses with robotic automation.**

I. Introduction

As collaborative robotics technology advances, a growing array of robotic arm hardware has emerged in recent years [1-3], offers a wide range of options and price points. This progress supports the integration of collaborative robots into various sectors, including catering, agriculture, and construction, with many startups and research teams applying these technologies commercially [4-7]. However, robotics remains an interdisciplinary field with high entry barriers, necessitating expertise in computer science, robotics, and electronics. This complexity hinders the rapid adoption of robots in smaller commercial projects.

Robot manufacturers commonly create specialized software, complemented by Software Development Kits, to facilitate system integrators to perform secondary development. Some products feature intuitive user interfaces, allowing end-users to quickly become acquainted with them. However, the proprietary nature of this software development leads to limited code reusability; software tailored for one product line or brand cannot be easily adapted for use with another, which significantly complicates hardware updates or iterations for users once development is complete.

Numerous advanced general robotic software solutions like ROS 1 and ROS 2 (Robot Operating System) [8-9], YARP (Yet Another Robot Platform) [10], OROCOS (Open Robot Control Software) [11], OPC UA (Open Platform Communications Unified Architecture) [12], and LCM (Lightweight Communications and Marshalling) [13] have significantly contributed to the robotics field, enhancing code reusability, and reducing workload. These platforms offer specialized tools and libraries to support the development of robust robotic applications, focusing on various aspects such as communication, control, and industrial automation.

However, these solutions often target experts and come with steep learning curves. Excluding ROS, many frameworks address specific aspects of robotics, requiring additional effort for comprehensive functionality. ROS stands out for its system-wide openness and scalability but introduces complexity in version management and dependency conflicts. Effective use of ROS also requires substantial programming and robotics knowledge, underlining the interdisciplinary challenge in robotics system design. An anecdotal account from a doctoral student in mechanical engineering and robotics highlights this challenge; without any previous experience with Linux, the student described a demanding period of self-directed learning, which lasted several months before they could successfully operate a laboratory robot using ROS for the first time.

This scenario underscores the necessity for more accessible, versatile, and user-friendly robotic software platforms that can lower the entry barrier for interdisciplinary research and development in robotics, fostering innovation and accelerating the deployment of robotic solutions across various sectors. Addressing these challenges, this paper introduces the WinGs Operating Studio (WOS), a novel low-code platform designed for the operation of robotic applications.

The Key Features of WOS include:

- *Hardware Abstraction and Standardized APIs:* It abstracts robotics hardware into standardized resources, offering APIs across various protocols. This enables automatic feature availability based on component compliance.
- *Broad Compatibility & Build-Free Deployment:* WOS ensures wide-ranging compatibility across computing platforms, from PCs to mobile devices, by distributing as a binary executable.
- *Web-Based GUI and Applications:* It offers low-code robot programming applications via a web UI, catering to various scenarios. For a specific scenario, it offers a third-party app store for custom human-machine interactions, ensuring optimal user experience.

This paper is structured as follows: Section 2 details WOS's design philosophy and technical specifics. Section 3 showcases the system's capabilities and reliability through experiments and real-world case studies. The conclusion in Section 4 summarizes the research findings and future works.

This project is supported by A*STAR under "RIE2025 IAF-PP Advanced ROS2-native Platform Technologies for Cross sectorial Robotics Adoption (M21K1a0104)" programme.

I-Ming Chen and Yue Feng are with Robotics Research Center, Nanyang Technological University, 639798 Singapore; Emails: michen@ntu.edu.sg, yue011@e.ntu.edu.sg; Weicheng Huang is with WinGs Robotics LLC, NY, 10314 USA; Email: Info@WingsRobotics.com

II. METHODOLOGY

A. Scope

Different from the well-known robot meta-operating system ROS, WinGs Operating Studio (WOS) conceptualized as a Robot as a Service (RaaS) platform [14], focuses on rapid prototyping and deployment of robotics projects to meet end-user needs. It supports a wide range of computing infrastructures, including standard PCs, embedded systems, edge devices, and Android mobile devices, with the goal of broadening accessibility to robotics development. WOS is offered as a binary executable, simplifying integration, and enhancing security. Its architecture separates the transport layer from the customer application, enabling the use of any programming language that supports HTTP, WebSocket, or other network protocols like MQTT. In addition, we developed a comprehensive web-based GUI through the WOS APIs, enabling users without coding experience to visually program robots. Table I provides a comparison between WOS and ROS, highlighting their key differences.

TABLE I. COMPARISON BETWEEN WOS AND ROS

Category	Platform	
	WOS	***ROS***
Deliverable Format	Executable Binary	Source Code
API Intractability	No WOS Dependencies Required	ROS Dependencies Required
User Interface	Web-based GUI	Command Line Interface

B. Design

Fig. 1 depicts the node-based architecture of WOS, designed to streamline robotic application development and deployment. At its heart, WOS Core performs communication forwarding and Node lifecycle management. Components, serving as specialized Nodes, manage hardware, whereas Application Nodes handle non-hardware logic, such as algorithms. Functional Nodes connect to WOS Core via APIs to register Services, making them available to front-end GUIs or custom programs. This architecture ensures a scalable and modular approach to robotics, highlighting the efficient integration and management capabilities of WOS.

1) Node

Node serves as the basic executable component within WOS, with its lifecycle, data production, and exception handling governed by the system. To accommodate varied operational needs, four distinct node types are delineated:

- *CMD Node:* A standard host process. Primarily for debugging purposes.
- *Container Node:* A Docker [15] container process, facilitating runtime environment encapsulation.
- *Internal Node:* Built-in WOS utilities, not open for customization.
- *Macro Node:* Idempotent functions for executing simple operations.

Node can have inputs and outputs. Nodes receive inputs via environment variables or command-line parameters, and output data in stdout or through the WOS API.

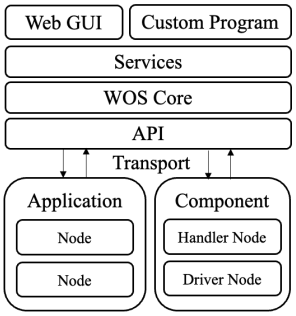

Fig. 1: WOS Design Architecture

The node lifecycle includes states of starting, running, and ending, with capabilities for pausing and resuming provided by WOS. Node definitions are required in YAML or JSON format or can be dynamically registered via the WOS API, with at least one executable action specified.

2) Service

WOS services are the control interface for node. Node can register itself as a service provider and other node will be able to control the node using standardized API. WOS Services underpin the system's communication and operational framework, focusing on Topics, Requests, and Actions:

- *Topic:* Utilizes a publish/subscribe model for efficient, event-driven communication across the system, allowing services to broadcast updates and enable other nodes to subscribe for real-time information.
- *Request:* Enables direct interactions between components for specific operations or data retrieval, following a request-response pattern for synchronous message exchange.
- *Action:* Allows the definition and execution of long-running task through simple triggers, automating tasks within the WOS ecosystem. Facilitating feedback and cancel mechanism.

3) Component

Robotic components are virtual representations of robotic hardware managed by WOS, facilitating simplified interaction with hardware by abstracting functionality. These components are divided into two interconnected parts:

- *Handler:* The control layer for the Component, executing algorithms and managing Driver communication through nodes and services.
- *Driver:* Provides the hardware connection, handling essential low-level communications. Developed as nodes and services.

Components are structured to potentially include child Components, allowing for complex hierarchical relationships within the robotic system. For instance, an end effector attached to a robotic arm would be considered a child Component, with WOS managing their interaction and necessary cartesian transformations.

4) Application

Application enables the ability to dynamically extend the capabilities of WOS. Applications are sophisticated assemblies of Nodes and other static assets. This architecture enables the creation of versatile and complex applications tailored to

specific tasks. Crucially, all applications can be presented through a Web UI, which serves as the interactive front end.

The system also encompasses a set of system applications, offering critical functionalities:

- *Graph:* A visual tool for orchestrating node execution., providing the ability to create sequential and conditional node execution logic. simplifying complex task design.
- *Trigger:* Enables event-driven interaction, allowing WOS to react when certain event occurred.
- *Real-time Control*: Empowers users with the capability to manage robotic components via a diverse array of interfaces, including Web UI, gamepads, VR, and more.

C. Communication Patterns

1) WOS API

The core of WOS's communication infrastructure is formed by its API, which exposes a suite of services including Topics, Requests, and Actions through Service registration. It enables the publishing, subscribing, and unsubscribing of Topics, the sending of Requests and receiving of responses, as well as the running of Actions, providing feedback, and canceling Actions, which facilitates communication among robotics components.

2) Transport

WOS supports diverse communication needs through two main transport mechanisms:

- *Internal:* This method enables direct interactions within the WOS system, allowing components and services to efficiently execute and communicate through internal API calls.
- *External:* WOS decouples its transport layer and allows varies of transport protocol to access WOS resource. Such as HTTP, WebSocket, WebRTC, MQTT or pure TCP connection.

D. Extensibility

WOS is designed with extensibility at its core, enabling users and developers to customize, extend, and integrate a wide range of functionalities and resources. Here's how WOS supports extensibility across various aspects:

1) Customized Node

WOS allows for the creation of customized nodes, and application providing developers with the tools to tailor functionalities specific to their needs while WOS manages their life cycle. For example, an ArUco [16] marker detection node can be developed to add pose estimation capabilities to a robotics project, leveraging WOS's framework for lifecycle management, including starting, running, and ending processes, as well as handling outputs and exceptions.

2) External API Request

WOS supports integration with external APIs, allowing scripts to make requests from outside the WOS ecosystem. Developers has the flexibility to choose their own programming languages and build process to build their program as long as it has the ability to communicate to WOS core.

3) Third Parties' Resources

The platform is open to incorporating third-party resources, WOS has an adapter layer that talks to ROS and other popular communication patterns. This allows painless migration of existing functionalities to WOS platform. such as the Franka Emika ROS driver [17], WOS utilize ROS driver to control Franka Robots [2] and this pattern can be extended to all other ROS ready robotics component. This openness ensures that WOS can serve as a comprehensive ecosystem for robotics projects, supporting a vast array of components and algorithms from different vendors and communities. By allowing third-party integrations, WOS enhances its utility and adaptability, making it a versatile platform for robotics development and deployment.

III. Experiments

This chapter focuses on validating WOS's performance through stress tests in varied conditions and showcasing its application in specific cases to illustrate its efficiency and low entry barriers. We also highlight two application cases: dual-arm coffee preparation using WOS's graph low code module and VR-assisted remote toy spray teaching via WOS API. These examples demonstrate WOS's effectiveness in streamlining robotic operations and its potential to enhance automation in small businesses.

A. Performance Test

The performance evaluation was conducted on a Raspberry Pi 3 Model B, an Arm-based single-board computer equipped with a Quad Core 1.2GHz CPU. The assessment focused on the performance of WOS's binary core across different communication modes and message sizes.

The tests covered three modes of communication:

- *Internal Process:* Message passing within the WOS program.
- *Localhost Process:* Message passing from another program to WOS on the same machine.
- *Remote through Ethernet:* Message passing from another computer to WOS via onboard Ethernet through a router.

Each communication mode was tested with varying message sizes, ranging from an empty payload to trajectory data of

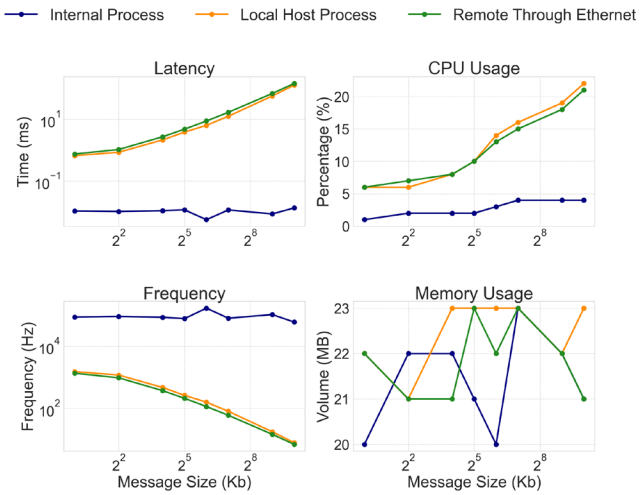

Fig. 2: WOS Performance Test Results

different sizes for a 6-DOF robotic arm, between 1KB and 1MB. The messages in the performance test were formatted in text-based JSON, undergoing message decoding, routing, and encoding.

As shown in Fig. 2, internal process communication exhibited the lowest latency and highest transmission rate, making it the most efficient mode. In contrast, remote communication via localhost and Ethernet significantly increased latency and reduced transmission rates, particularly with larger message sizes. However, no data loss was observed due to the advantage of TCP, even though larger message sizes did result in a drop in processing frequency.

Internal process communication, using direct WOS API calls, remains the most efficient, whereas external API messaging relies on text-based JSON, leading to high CPU usage due to the overhead of encoding and decoding. Despite this, memory usage remained stable across different communication modes, indicating consistent memory management.

To improve message encoding and decoding efficiency, WOS is adopting Protocol Buffers [18] while exploring the use of UDP for scenarios that require higher frequency stability, such as joint state publishing.

B. *Application cases*

1) Dual-arm coffee preparation

This section highlights the innovation of WOS's Graph application, demonstrating the use of visual flowcharts for managing dual robotic arms in complex tasks. This method greatly simplifies the learning curve for robotic arm manipulation, allowing those without robotics engineering expertise to easily engage in the creation and design of robotic applications. We showcase the setup and calibration of two different brand robotic arms of various sizes and costs, using WOS. Within just a few hours, these arms were ready to successfully prepare a capsule coffee, illustrating the system's efficiency and user-friendly approach.

The coffee brewing experiment featured a capsule coffee machine and two distinct robotic arms: the Franka Emika Research 3 (FR3) [2], a 7-degree-of-freedom robot with a gripper, and the STR400 [3], a 6-degree-of-freedom compact robot with a 3D-printed shovel as its end effector. The FR3 uses

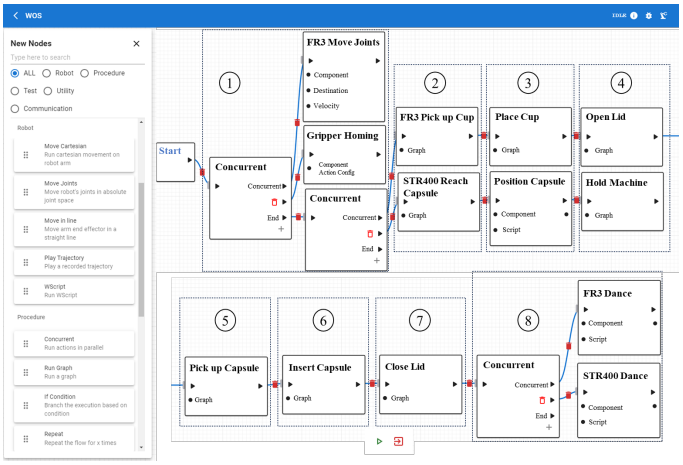

Fig. 3: WOS Graph Interface for Dual Robotic Arms Coffee Preparation

Docker [15] encapsulation for its operation in FCI (Franka Control Interface) mode [17], enabling precise PD (Proportional-Derivative) position and velocity control via ROS Control Packages. In contrast, the STR400 is natively supported by WOS, directly accepting low-level motion commands. WOS's Graph allows for the invocation of internal motion planning algorithms, facilitating varied movements for both robotic arms in the brewing process.

Fig. 3 presents screenshots stitching of the WOS Graph's interactive interface. The left side lists available Graph Nodes, including various trajectory planning modules for robots, as well as logic modules required for low-code programming. These nodes can be dragged from the left into the central workspace, where parameters are set and logics linked, enabling choreography of robotic actions. After arranging the actions, initiating the program is done by clicking the green play button at the bottom, with the Start Graph Node serving as the entry point.

The WOS Graph in Fig. 3 displays the main program for the Dual-arm coffee preparation task, with eight outlined areas corresponding to the eight stages shown in Fig. 4. The diagram utilizes several functional nodes to achieve the task, specifically Concurrent, Run Graph, WScript, and Component Action, most of which are assigned nicknames for ease of debugging. The

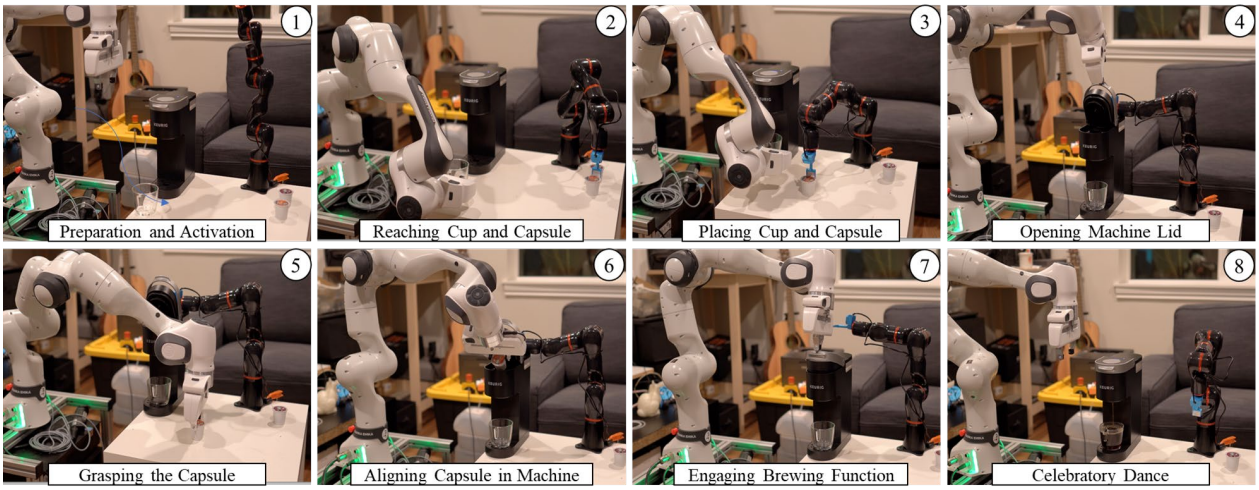

Fig. 4: Sequential Collaboration of Dual Robotic Arms in Capsule Coffee Preparation

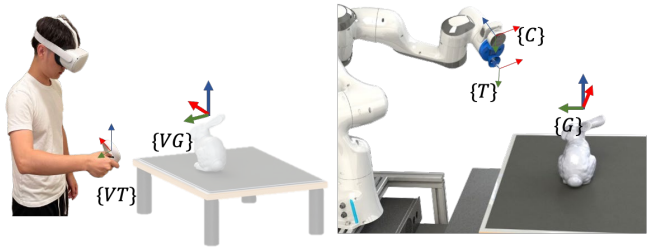

Fig. 5: Experimental Design for Remote VR Teaching in Spray Painting

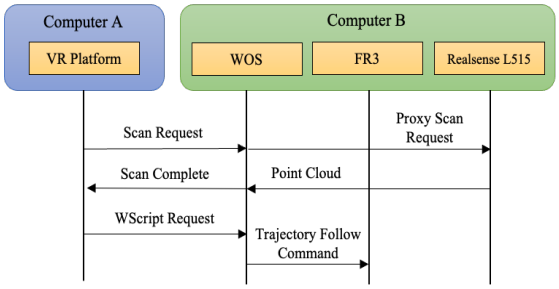

Fig. 6: Process Implementation of the VR Teaching System

Concurrent node used in the first group of nodes initializes FR3's pose with the Move Joints node while homing the gripper simultaneously with the Component Action node. Upon completion of these actions, it proceeds to the next stage with another Concurrent node, whose branches orchestrate the actions of FR3 and STR400, enabling them to operate simultaneously in a pre-arranged sequence. Apart from "Position Capsule" node in group 3, which employs WScript (a WOS-defined script for editing a series of robot arm actions), other nodes in groups 2, 3, 4, 5, 6, and 7 are Run Graph, calling subgraphs during the main program execution. This segmentation allows complex tasks to be divided into smaller, manageable tasks. At the end of program, the last Concurrent node is used to execute pre-programmed dancing actions using WScripts for both robots.

Following this, Fig. 4 illustrates the specific implementation steps of these processes: Initially, the environment is prepared, and both robotic arms are activated, setting the stage for the task at hand (Fig. 4-1). The FR3 takes the initiative by picking up a cup, while the STR400 arm approaches a predetermined position where the coffee capsule is placed (Fig. 4-2). Subsequently, FR3 carefully places the cup into the coffee machine's holder, and STR400 adeptly positions the coffee capsule at the specified spot (Fig. 4-3). In a coordinated effort, FR3 proceeds to open the coffee machine's lid, with STR400 applying pressure on the machine to ensure stability and prevent any displacement (Fig. 4-4). Following this, FR3 skillfully grasps the coffee capsule (Fig. 4-5), and then positions it within the machine, making necessary adjustments for optimal alignment (Fig. 4-6). Once the capsule is correctly placed, FR3 closes the lid and engages the brewing function (Fig. 4-7). The culmination of this intricate process is marked by both robotic arms performing a synchronized dance, signifying the successful completion of a freshly brewed cup of coffee (Fig. 4-8).

The results were highly positive, showcasing the system's efficiency. While areas like improved path planning and sensor integration for flexible item placement were noted for enhancement, researchers successfully implemented the process in just two hours. This highlights WOS's efficiency, adaptability, and the practicality of robotic collaboration in daily tasks.

2) VR Teleoperation Toy Spraying

The design of the WOS, featuring a separation between the frontend and backend, enables the majority of functionalities available in the web-based interactive interface to be accessible via API calls. This section demonstrates how a VR application leverages the WOS to manage robotic hardware and sensors, conveniently facilitating a simple VR application for remotely teaching a robotic arm to spray paint toys in actual use cases.

Fig. 5 shows the setup for VR-based remote teaching of spray painting, highlighting key coordinate notations. An FR3 robotic arm, equipped with a RealSense L515 camera [19], with its position denoted as {C}. An LED light marks the spray gun nozzle's position as {T}. The global coordinate system {G} is set 0.2 meters above the center table toy rabbit, with the Y-axis of {G} pointing to one table side and the Z-axis vertically up. On the left, an operator wearing VR gear scans the operation table, viewing a point cloud image from the scene. The VR's global coordinate, {VG}, matches the real-world global coordinate setup, placed 0.2 meters above the table center, with identical orientation. The VR controller, representing the spray gun nozzle, is marked as {VT}, mirroring the real nozzle's position and orientation. This setup ensures that {VT}'s movement relative to {VG} directly replicates {T}'s movement relative to {G}. Recording the controller's movements allows the robotic arm to replay these actions, effectively teaching it the spray-painting task.

Fig. 6 outlines the system's implementation, utilizing two computers: Computer A runs a VR teaching demo using Unreal Engine 5, while Computer B runs WOS, managing the FR3 robotic arm and a RealSense 3D camera. Initially, a button in the VR demo triggers a scan of the environment using the RealSense camera, controlled via WOS API. The resulting point cloud image and camera position are sent back to Unreal Engine 5, where coordinate transformations align the virtual {VG} with the real {G}. Operators can then see the 3D point cloud and start recording the controller's trajectory at 60Hz. Due to hardware noise and hand tremors, the raw data undergoes downsampling to 1Hz before being sent to WOS. WOS then interpolates the trajectory using internal WScript node, making it suitable for playback by the robotic arm. Moving from the system's technical details to its application, Fig. 7 shows the virtual reality setup on the left, where the operator, with a spray-painting device, aims at a point cloud

Fig. 7: Virtual Reality Scene and Corresponding Physical Setup

Fig. 8: Trajectory Data Across Cartesian Spaces in the Experiment

table to record movements. The right side shows the actual experimental setup.

Fig. 8 captures the trajectory data in three distinct Cartesian spaces throughout the experiment. The raw data of hand movements collected in VR are represented by small blue dots. These points, after being down sampled and smoothed by the motion planning algorithm, are depicted as solid yellow lines, serving directly as references for the trajectory tracking PD controller. Finally, the green dashed lines record the trajectory that the robot arm actually follows during its operation. All three types of data consist of 1200 steps at a frequency of 60Hz, indicating the entire spray teaching lasts for 20 seconds. The standard pose description in the WOS utilizes the most intuitive Cartesian coordinates x, y, z, along with Roll, Pitch, Yaw, XYZ Euler angles for rotation. Consequently, the six sub-figures visualize data for these dimensions respectively.

The VR teaching experiment demonstrated encouraging outcomes. When comparing the raw data recorded from VR with the actual trajectory followed by the robotic arm during execution, it achieves a mean error of -0.00225 meters and a standard deviation of 0.01408 in position data, along with a mean error of -0.00003 radians and a standard deviation of 0.04973 in rotation data. Using the WOS's straightforward API interface and its built-in motion planning algorithm, the robot's movements are made uniformly smooth without significantly distorting the original recorded data. Importantly, such system design does not depend on specific hardware devices, meaning engineers working on the VR side do not need to concern themselves with the specific models of robotic arms and cameras used on-site.

IV. Conclusion

In conclusion, this paper has introduced the WinGs Operating Studio (WOS), a low-code integration platform for robotic arms designed to simplify automation in small enterprises. Leveraging its hardware abstraction and standardized APIs design philosophy, WOS provides a solution that eases the management of varied robotic hardware through a single binary executable. Its efficiency and compatibility ensure smooth performance on budget-friendly PC hardware. The platform's web-based GUI and application framework, enabling intuitive flowchart programming and flexible APIs, have demonstrated effectiveness in scenarios such as dual-arm coffee preparation and VR-controlled spray painting. WOS stands as a significant tool for reducing technical hurdles in robotic automation, highlighting its capacity to drive innovation.

For future work, we aim to explore communication technologies that maintain compatibility with current systems while achieving higher efficiency. Additionally, we plan to expand our application domains to include more industrial sectors, understanding and catering to their specific needs. To enrich the functionality of modules, we intend to devise strategies to encourage broader participation in the development of the WOS ecosystem, further enhancing its versatility and impact across various industries.

References

[1] "Universal robots." https://www.universal-robots.com. Accessed: 05-02-2024.

[2] "Franka emika, panda." https://www.franka.de. Accessed: 05-02-2024.

[3] "WinGs Robotics, STR400" https://www.wingsrobotics.com/str400. Accessed: 05-02-2024.

[4] "Artly Coffee" https://artly.coffee/. Accessed: 05-02-2024.

[5] "Zordi" https://www.zordi.com/. Accessed: 05-02-2024.

[6] Zhang, K., Yang, L., Wang, L., Zhang, L., & Zhang, T. (2012). Design and experiment of elevated substrate culture strawberry picking robot. Nongye Jixie Xuebao= Transactions of the Chinese Society for Agricultural Machinery, 43(9), 165-172.

[7] Asadi, E., Li, B., & Chen, I. M. (2018). Pictobot: A cooperative painting robot for interior finishing of industrial developments. IEEE Robotics & Automation Magazine, 25(2), 82-94.

[8] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[9] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics, 7(66), eabm6074.

[10] Metta, G., Fitzpatrick, P., & Natale, L. (2006). YARP: yet another robot platform. International Journal of Advanced Robotic Systems, 3(1), 8.

[11] Bruyninckx, H. (2001, May). Open robot control software: the OROCOS project. In Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164) (Vol. 3, pp. 2523-2528). IEEE.

[12] Hansch, G., Schneider, P., Fischer, K., & Böttinger, K. (2019, September). A unified architecture for industrial IoT security requirements in open platform communications. In 2019 24th IEEE international conference on emerging technologies and factory automation (etfa) (pp. 325-332). IEEE.

[13] Albert S. Huang, Edwin Olson, and David C. Moore. LCM: Lightweight Communications and Marshalling. In IEEE/RSJ Inter- national Conference on Intelligent Robots and Systems, pages 4057– 4062, 2010.

[14] Chen, Y., & Hu, H. (2013). Internet of intelligent things and robot as a service. Simulation Modelling Practice and Theory, 34, 159-171.

[15] Boettiger, C. (2015). An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review, 49(1), 71-79.

[16] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., & Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition, 47(6), 2280-2292.

[17] Franka Emika GmbH. https://frankaemika.github.io/docs/. Accessed: 05-02- 2024.

[18] Currier, C. (2022). Protocol buffers. In Mobile Forensics–The File Format Handbook: Common File Formats and File Systems Used in Mobile Devices (pp. 223-260). Cham: Springer International Publishing.

[19] "Intel RealSense" https://www.intelrealsense.com/lidar-camera-l515/. Accessed: 05-02-2024.