Plug-and-Play Software Architecture for Coordinating Multiple Industrial Robots and Sensors from Multiple Vendors

Honglu He, Burak Aksoy, Glenn Saunders, John Wason, John T. Wen

Abstract— **Integrating robots from multiple vendors into an automation ecosystem is a challenge that hinders the deployment of industrial automation. The COVID-19 pandemic has added urgency to the need for automation to maintain productivity while observing physical distancing. To address this challenge, we propose a plug-and-play software architecture using Robot Raconteur (RR). Our approach allows quick and seamless integration of disparate sensors and robots into a production system, enabling complex tasks to be performed efficiently. This technology has the potential to transform the robotics revolution in the industry.**

In this paper, we present a case study that demonstrates the effectiveness of our plug-and-play software architecture. The demonstration includes a plug-and-play smart teachpendant and a mock production line that performs pick-and-place tasks. Using Robot Raconteur enables ready integration of robots from multiple vendors, leading to improved productivity and cost savings.

I. INTRODUCTION

Robots have been playing a key role in advancing industrial automation in the past 60 years. While robots have significantly increased the efficiency and productivity in industry with decreasing cost of industrial robotic systems, its adoption is far from universal, particularly for small and medium manufacturers. A key deterrent is the considerable amount of time and effort required to set up and integrate robots and sensors into an automation system. Further, once a system is setup, it is mostly locked to particular robot vendors preventing expansion and reconfiguration to meet varying needs. Plug-and-play capability and interoperability refer to the ability of users to seamlessly use devices, such as mice and keyboards, without the need for reconfiguration or adjustment. This concept extends to the field of robotics by enabling the diverse usage of sensors and manipulators from a range of vendors. This versatility allows manufacturers to cater to unique requirements and efficiently redistribute robots across different product lines. However, the lack of standards has thwarted the emergence of the plug-and-play capability in the industrial robot industry [1].

The emergence of Industry 4.0 is transforming the automation and robotics from the traditional stand-alone model to highly interconnected services. Standardization is a key factor in the evolution and adoption [2]. OPC Foundation introduced Open Platform Communications Unified Architecture (OPC/UA) for Industry 4.0, including robotics [3], but there is a lack of communication library and software proposed for robotics to incorporate such standards. Robot Operating System (ROS) [4] and ROS 2 [5] aim to develop reusable robotics code in research and development, but the plug-and-play capability remains elusive. It also has steep learning curve, with limited platform support and security support. ROS-Industrial has led a consortium effort to apply ROS and ROS 2 to industrial robots [6]. It has increased the adoption of ROS and ROS 2 but does not address plug-and-play and interoperability among different vendors. A Plug-and-play robotic system using LabView was proposed in 1998 for parts assembly [7]. In [8], Nguyen introduces the idea of smart home with plug-and-play robots and sensors using ROS. Companies like Ready Robotics has also offered the solution of easy programming platform (Forge OS) for different robots, but is still tied to the programming language of the underlying industrial robots [9].

However, the seamless plug-and-play capability for multiple interoperable robots with minimal or no setup from users is still absent.

This paper presents a solution to this gap. Our approach is based on the open source middleware Robot Raconteur (RR) developed by Wason Technology to interface with different robots and sensors [10], [11], and its use cases have been deployed in different robotics systems [12]–[14]. As a demonstration, we have developed a testbed with robots from three different vendors (ABB, Universal Robotics, and Sawyer) sharing the same workspace with multiple types of end-effectors and sensors. We have developed a smart RR-enabled teachpendant that can connect to and program different robots. Partnering with a local manufacturing business, we developed a mock production line with 3 robots picking small objects from bins and filling into package boxes on a moving conveyor belt. The testbed has a virtual simulated version developed during the pandemic with exactly the same interface, and this capability allows users to simulate tests with real-time position streaming control.

Contribution: We incorporate RR standardized interface with robots from different vendors, and demonstrate the plug-and-play capability using an RR-enabled smart teachpendant and a mock robotic production line with interchangeability.

This paper is organized as follows. Section II briefly summarizes the key attributes of the Robot Raconteur Mid-

Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, heh6@rpi.edu aksoyb2@rpi.edu

Manufacturing Innovations Center, Rensselaer Polytechnic Institute, saundg@rpi.edu

Wason Technology, LLC, wason@wasontech.com

Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, wenj@rpi.edu

dleware system and describes RR standardized interface for robots and sensors. Section III describes the smart teachpendant implementation and its use cases. Section IV presents the demonstration testbed. Section V lists the possible alternative solutions for plug-and-play capability and RR's advantage. Section VI presents the conclusion and future work.

A directory of links to available Robot Raconteur software and projects has been created [15]. The teach pendant software can be found in the "pyri-project" GitHub organization [16]. Refer to these locations for URL links to the repositories listed in this paper.

II. Robot Raconteur Overview

A. RR Interface Standards and Drivers

Robots from different vendors have different API and standards, and that could potentially pose a big challenge if a system needs to incorporate different robots. RR is an object-oriented communication library for robotics and automation systems, and it is compatible with most major platforms and programming languages [17]. Supported platforms include Windows, Linux, MacOS, iOS, Android, and FreeBSD. Supported programming languages include C++, C#, Python, MATLAB, Java, and LabView. RR is adding support for more platforms and programming languages over time.

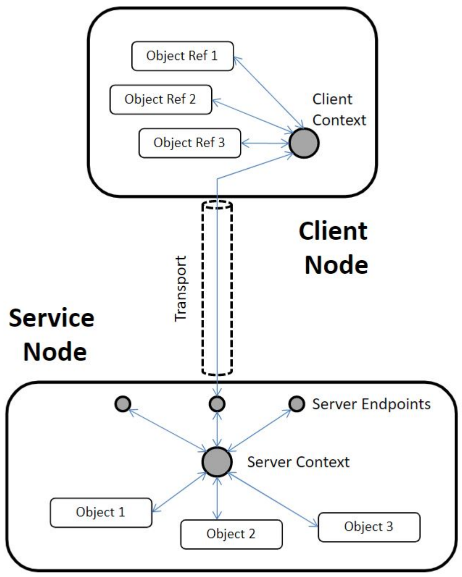


Fig. 1: Robot Raconteur Client-Service Architecture [10]

RR uses a service-client structure as its basic architecture, shown in Fig. 1, enabling easy integration of disparate systems and seamless communication between different robots, sensors, and other devices. Service nodes typically contain one or more service objects, each with attributes defined in the RR service definition. By registering an RR service with a service name and declared object, those objects are exposed to the local area network (LAN), allowing clients to connect by specifying the service's URL. This provides access to the service's attributes and functions, allowing for quick and easy communication and control of the hardware.

RR offers several advantages over other frameworks, including a short learning curve, minimal setup time, multi-platform and multi-language support, and competitive latency performance compared to ROS and ROS 2 [17]. Drivers are generally developed as RR services, with client nodes as the user interface. RR provides plug-and-play behavior using two strategies [17].

- Dynamic runtime client proxy and type generation for scripting languages such as Python, MATLAB, and LabView
- Standardized service types for classes of devices

The dynamic runtime capability is typically used during development. The standardized service types are used to provide interoperability between different devices. Clients are designed to understand service types that correspond to classes of devices. Devices and drivers that implement the standard service type can be interchangeably used by the client without modification. This plug-and-play capability allows for easy integration of new devices into existing systems without the need for extensive development effort.

As part of the Advanced Robotics for Manufacturing Institute (ARM) project "Robot Raconteur (RR): An Interoperable Middleware for Robotics" [18], standard service definitions were created that represented multiple classes of devices and allowed for interoperability between different devices. The developed service definitions are available on GitHub. Currently the published "Group 1" has 45 service definitions, 41 object types, 162 structures, 71 namedarrays, and 5 pods defined. The standard definitions [19] and different RR directories [15] are available on Github.

B. Robot Drivers

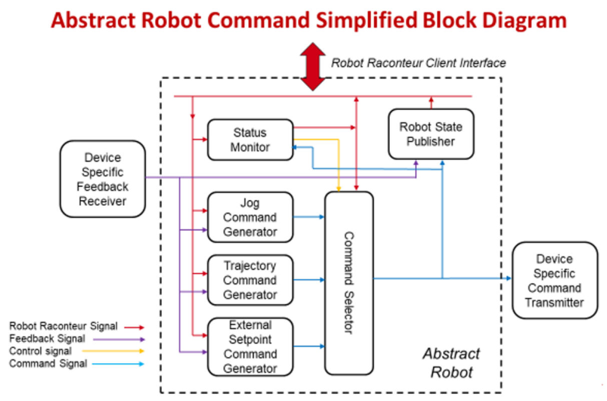

Abstract Robot Command Simplified Block Diagram
Robot Raconteur Client Interface
Status Monitor
Robot State Publisher
Device Specific Feedback Receiver
Jog Command Generator
Trajectory Command Generator
External Setpoint Command Generator
Command Selector
Device Specific Command Transmitter
Abstract Robot
Robot Raconteur Signal
Feedback Signal
Control signal
Command Signal

Fig. 2: Robot Raconteur Abstract Robot Class Structure

RR is a comprehensive solution for standardizing communication interfaces for robots and sensors. Each robot driver is implemented as an RR service node, communicating with a single robot and exposing the standard robot object over the LAN, with optional password security. To connect to the service, initializing an RR client connection will return the object containing all the functions and properties defined in the RR description. This allows users to control different robots with the same client script, using the standard RR robot definition.

Example 1 in [17] shows the standard robot definition `com.robotraconteur.robotics.robot.Robot`. It is the standardized RR definition that is adaptable to all industrial robots with RR drivers.

Each robot has a distinct "Robot Info" `yaml` description file that contains basic robot metadata such as name, joint velocity/acceleration limits, kinematics, and tool information. The driver loads this file automatically and provides this information to the client as metadata. In our setup, the robot driver runs on a standalone hardware, specifically the BeagleBone x15 board, loaded automatically as a boot-and-run system.

Figure 2 shows a block diagram for "Abstract Robot". Abstract Robot is a software library that provides common functionality for standard robots that implement `com.robotraconteur.robotics.robot.Robot`. Robots can support multiple command modes, including jog, trajectory, real-time joint position, real-time joint velocity, and home. The metadata informs clients what modes are supported. Abstract Robot allows for the rapid development of drivers, since for most robots only the robot-specific interface needs to be developed.

C. Sensor Drivers

Similar to robot drivers, each sensor possesses its own hardware running the standard RR service. For object detection devices, RR service shares the same service definition, enabling clients within the system to execute identical code to verify detected objects. The Cognex sensor, an industrial machine vision camera, operates as a black box with onboard template matching [20] It is configured through a proprietary client software application on Windows, and at runtime directly outputs detected object pose and confidence score via the TCP/IP protocol. Consequently, the RR driver only reads the output from Cognex and configures the data in a standard object detection definition. For Kinect Azure or a webcam, the RR driver processes the RGB image, runs the Yolo-v5 Object Detector [21], and configures the data in the same standard object detection definition. Utilizing cameras with open-source object detection libraries demonstrates the implementation of Artificial Intelligence (AI) with plug-and-play capabilities. In the developed testbed, object detection with Kinect Azure operates on the runtime computer due to computational requirements.

D. Tool Drivers and Configuration

Robot end-effector tools vary in form, thus we use a more general standard definition. The robot `yaml` file includes tool information, such as mass and tool center point (TCP) transformation from the robot flange. Tools operating on separate controllers or devices are considered individual plug-and-play nodes. To swap tools, users simply update the tool information in the robot `yaml` file. In instances where the tool operates on the robot controller I/O module, the tool service can run within the same robot driver node or independently. For plug-and-play purposes, we have configured robot tools as separate nodes to easily interchange different tools.

E. Plug and Play Capability

Each robot or sensor includes designated hardware running RR drivers to interface with the robot or sensor. This approach emulates an RR-enabled device, with RR drivers interfacing with robots or sensors and exposing standard services over the LAN. Hardware used includes BeagleBone X15 boards and Raspberry Pi 3B+. On each hardware, the RR drivers transform into a `systemd` service, enabling automatic startup upon booting without manual setup. Each RR driver contains a node name and a service name (or multiple, if running multiple services within a node), with an optional password. As each service possesses a unique URL, a client can easily connect to the service directly with the URL. To enhance plug-and-play capabilities, we incorporate RR service auto-discovery functionality to browse all running RR services within the LAN and search based on user-specified service and node names.

In general, users can control a robot by simply powering up the robot and the hardware running the RR service. Clients can then connect to the service by providing the service name in the LAN.

III. Smart Teachpendant

The ARM project "Open Source Teachpendant Programming Environment" [22] referred as "Smart Teachpendant" aims to create a vendor-independent programming platform by deploying the standardized RR drivers and their plug-and-play property. It provides an easy-to-use, high-level user interface and programming environment that makes advanced open-source robotics technologies accessible to a wider range of users. It simplifies programming and enhances the versatility of robotics applications for those without extensive programming experience. This section demonstrates the effectiveness of the plug-and-play architecture by summarizing the approach and key components of the smart teachpendant system in the following subsections.

A. Approach and Architecture

The overall architecture of the smart teachpendant system is shown in Fig. 3. The open-source system consists of a hand-held teachpendant with touch screen interface. The teachpendant has an enable switch and E-stop button that connect directly to the robot controller. A Space Mouse [23] joystick allows the user to jog the robot in either joint or Cartesian mode. User definable auxiliary buttons provide additional customizable functionalities. The teachpendant also allows the user to edit or execute robot programs and monitor status of robots and state of various signals. The runtime computer provides the heavy lifting behind the scene. It stores the user programs (plugins) and allows execution of the specified program. User programs are executed inside

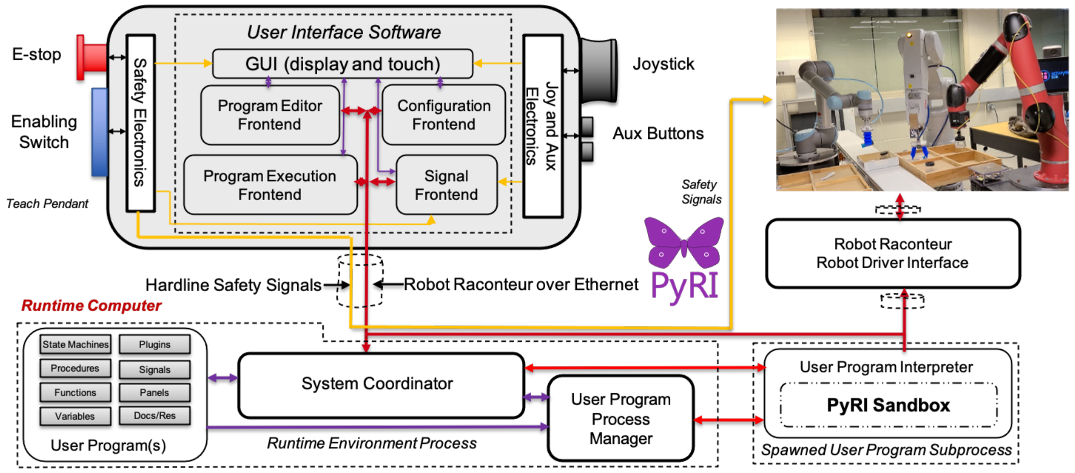


Fig. 3: Overall Smart Teachpendant Architecture: Demonstrates the approach of connecting the standard RR drivers to the user interface through the runtime computer.

the sandbox using the Python Restricted Industrial (PyRI) [24] dialect that optimizes Python language for use in industrial programming environments. To create and edit the user programs, Blockly Visual Programming [25] is used as an alternative to text-based Python to ease adoption and implementation. All the communication is performed by RR. The runtime computer uses the standardized RR drivers to interface to the physical robots and sensors. For ROS-enabled devices, an RR-to-ROS bridge may be used.

B. *Runtime Environment*

Runtime Environment is the core software that communicates with hardware, executes user programs, and interacts with the various devices that make up the system. The Runtime Environment is executed on the Runtime Computer, which is a permanent part of the automation system. The Runtime Environment communicates with the teachpendant WebUI Interface, which a technician can use to operate the automation system and develop user programs. The Runtime Environment is mostly developed in Python, using a micro-services architecture loosely based on serverless computing architecture [26]. Functionality is broken up into multiple small processes, with each process implementing part of the total functionality and communicating with other services such as device discovery and management, user program execution, variable storage database, WebUI server, jog and motion services, and vision related services (e.g., camera calibration, fiducial marker detection etc.). The Runtime Environment uses RR for communication between services within the runtime, and to communicate with other devices in the automation system. The RR standard service types and drivers discussed in Section II-A are used to interact with devices.

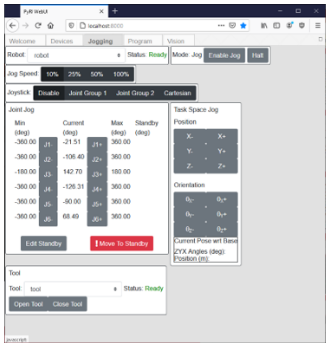
C. *WebUI and Programming*

The user interface of the teachpendant, WebUI, is designed to run inside a modern web browser. It uses web browser-based implementation without the need of any internet connection. Hence, it can be used on touch screen panels or standard laptop computers easily. The default user interface provides standard equipment management (e.g., robot jogging, waypoint teaching and playback, camera calibration, etc. (Fig. 4)). WebUI is also customizable to tailor to different plugins beyond the default components for the integration of other types of tools and functionalities (e.g., integration of force/torque sensors or deep learning trained object detection modules).

To create a user-friendly programming environment for novice users, the WebUI leverages the Blockly library. This is accomplished by integrating the Blockly editor workspace into the teachpendant WebUI and adding new blocks for robotics functionality. Blockly blocks and categories are added to the teachpendant using plugins. Each added Blockly block has a corresponding "generator" function that generates the appropriate Python code for the block based on what the user has selected. This function is called during the Python generation process for the block diagram. Typically each block will correspond to a single sandbox function. For more advanced users, scripting with the PyRI procedure editor is also available in the WebUI.

D. *Hardware*

For physical hardware (Fig. 5), a runtime computer cabinet and two prototype teachpendants with touch screen interface are implemented. The first prototype uses a touch screen tablet computer, Microsoft Surface Go [27]. The second prototype uses Raspberry Pi 4 [28] connected to a touch screen display. We adopt an open source design philosophy using easily sourced components, requiring minimal machining, while carefully consider the component cost. The CAD files for the design as well as the complete bill of materials are all open source, and may be readily replicated. The teachpendant may be adapted for either right-handed or left-handed users. The hardware prototypes may be considered as a ruggedized DIY design that considers both industrial and DIY users. The implementation is industrial or near-industrial. The emphasis is on 3D printed parts, and uses metals as needed. The



Robot: robot Status: Ready Mode: Jog

Jog Speed: 10% 25% 50% 100%

Joystick: Disable

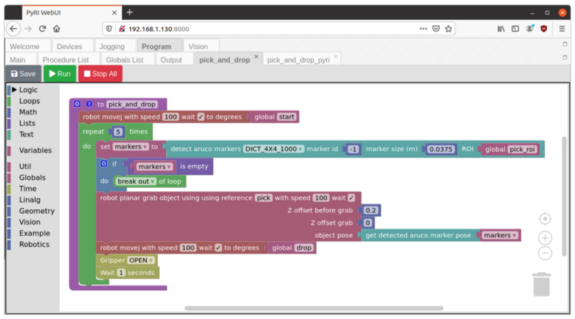
Joint Jog

Min (deg)		Current (deg)		Max (deg)	Standby (deg)
-360.00	J1-	-21.51	J1+	360.00	
-360.00	J2-	-106.40	J2+	360.00	
-180.00	J3-	142.70	J3+	180.00	
-360.00	J4-	-126.31	J4+	360.00	
-360.00	J5-	-90.00	J5+	360.00	
-360.00	J6-	68.49	J6+	360.00	

Current Pose wrt Base
ZYX Angles (deg):
Position (m):

Tool
Tool: tool Status: Ready

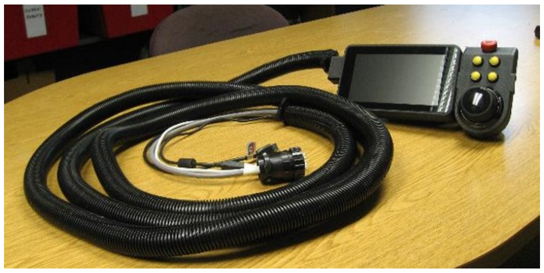
(a) Jog Panel



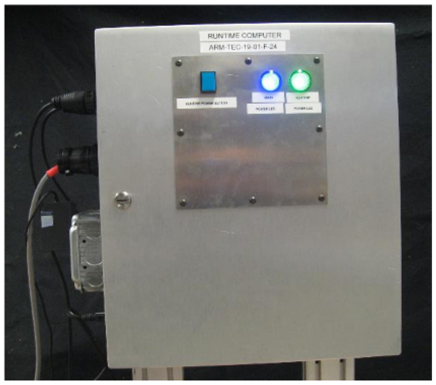
```
to pick_and_drop
  robot movej with speed 100 wait ✓ to degrees global start
  repeat 5 times
  do set markers to detect aruco markers DICT_4X4_1000 marker id -1 marker size (m) 0.0375 ROI global pick_roi
     if markers is empty
     do break out of loop
     robot planar grab object using using reference pick with speed 100 wait ✓
                                               Z offset before grab 0.2
                                                       Z offset grab 0
                                 object pose get detected aruco marker pose markers
     robot movej with speed 100 wait ✓ to degrees global drop
     Gripper OPEN
     Wait 1 seconds
```

(b) Blockly Programming Panel

Fig. 4: Example Teachpendant User Interface Panels



(a) Open-Source Teachpendant Raspberry Pi 4 Prototype



(b) Runtime Computer Cabinet

Fig. 5: Smart Teachpendant Hardware

design is modular for the I/O module interface and runtime computer cabinet entries. The cables have sufficient length (15') to allow user mobility. They have cable covers and strain reliefs for better protection and robustness.

E. Demonstrations

The project demonstrated the jogging, teach-and-playback, adding a new robot, execution of a pick and place collaboration tasks with ABB IRB1200, Sawyer, UR5, and Tormach ZA-6 robots from the same teachpendant system thanks to the implemented plug-and-play structure of robots. In addition, vision functionalities such as camera intrinsic and extrinsic calibrations, robot pose origin calibration, and vision-guided motion has been demonstrated. The overall process for a user to set up a vision-guided pick and place task given RR-enabled robots and a camera is demonstrated in the videos at [29], [30], and [31].

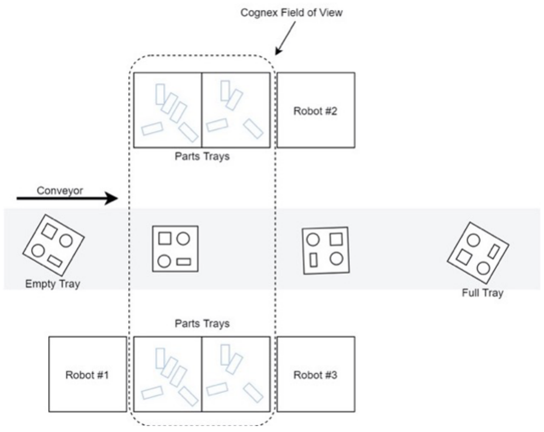


Fig. 6: Pick-and-Place Schematic: Robots take position #1, #2 and #3 to pick up objects from bins and fill in empty trays on the conveyor belt.

IV. Demonstration Testbed

Pick and place tasks with conveyor belts are among the most common scenarios in the industry. A local manufacturing business has provided us with a sample box containing soap, toothpaste, perfume, and a round bottle. In this paper, we use Sawyer, UR5, and ABB IRB1200 robots from three different vendors to accomplish pick and place tasks on a moving conveyor belt, demonstrating the plug-and-play capability with RR.

The testbed is a smart and re-configurable system, containing three robots, a moving conveyor belt, and two overhead cameras: Cognex and Kinect Azure, served as object localization sensors. Four bins holding the objects are placed on each side of the conveyor belt. The conveyor belt is moving at a preset constant speed. Our testbed schematic is shown in Fig. 6

Robots are mounted on mobile pedestals, and each robot has an OpenCV Aruco tag on its end-effector, allowing the overhead camera to detect and calibrate robot pose in camera frame. The system architecture is depicted as Fig. 9. Each robot/sensor is connected to the designated hardware running an RR service. And the hardware exposes the RR service over the LAN. There are also other RR services running on the run-time computer, including Yolo-v5 detection service and collision checking service which require more computa-

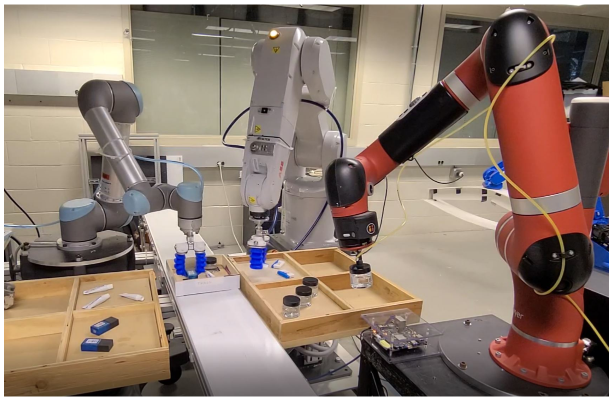
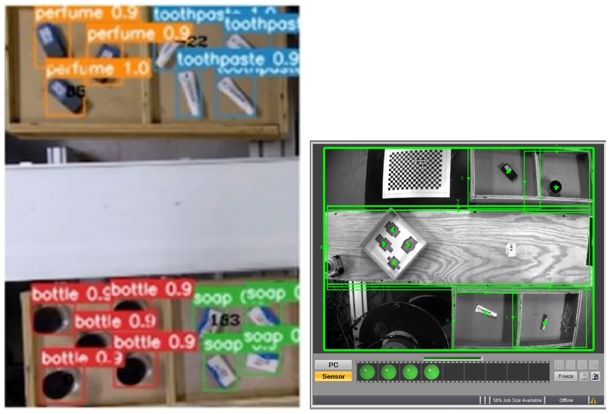


Fig. 7: Testbed Setup: Physical testbed for UR5, Sawyer and ABB IRB1200 collaboration to fill an empty box with four different objects from the bins.



(a) Yolo-v5 Object Detection (b) Cognex Object Detection

Fig. 8: Object Detection: A snippet of Kinect Azure running Yolo-v5 object detection and Cognex machine camera on four different objects.

tional power. A universal client accepts robot (Sawyer, UR, or ABB) and object arguments from users to assign specific robots to pick and place objects through the RR interface.

A. Reconfigurability

This testbed is easily reconfigurable in ways like swapping robots, sensors, objects and grippers. Standard RR drivers simplify the interchangeability of existing robots/sensors, while the auto-calibration process updates the robot relative pose within the testbed. The motion planning service generates collision-free trajectories for all robots. On the run-time client computer, there are local `yaml` files containing the testbed calibration information, which can be modified through a human interface introduced in Section IV-D.

B. Auto-calibration

To introduce a new robot to the system, it is crucial to obtain an accurate relative pose to the world frame. By providing initial and final robot end-effector poses, in which the OpenCV Aruco tag is visible to the overhead camera, the robot moves with a fixed tool orientation. Simultaneously,

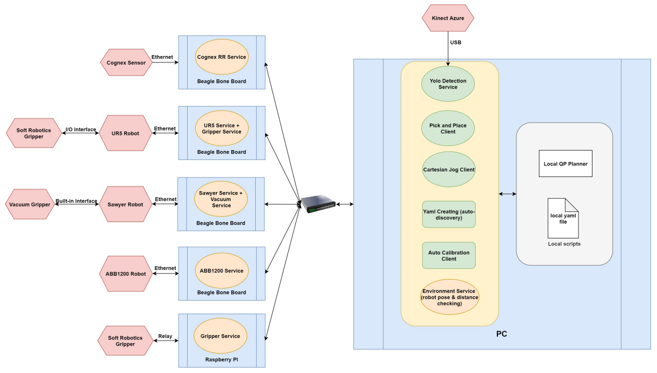


Fig. 9: Testbed Architecture: Each robot or sensor has its designated hardware running RR service, and the hardware is connected to the LAN, exposing the running service. Users can connect to all different services under the LAN using a client on a PC.

the overhead camera detection service reports the tag pose. Regression is performed between two sets of points to find the transformation matrix: points in the robot base frame and points in the world frame.

C. Motion Planning

For safety, tt is essential for robots to avoid collisions with the environment and other robots. In this paper, we use the Tesseract library [32] developed by ROS-Industrial for collision checking. Tesseract collision checking inputs a (`.urdf`) file containing the environment (static obstacles) and robots (rigid body chain) and outputs the closest two points and minimum distance (or penetration distance) between any two rigid bodies. We developed an RR service for Tesseract, which reads real-time robot joint positions and checks for potential collisions.

Given the desired pose to pick/place an object, a planner service commands the robot toward destination with real-time collision avoidance. Our motion planning service is a centralized node for all robots, planning trajectory with real-time collision avoidance. Define the collision vector as $\mathbf{d_c}$ pointing from closest point from robot to the environment, and collision distance $d = -n\left\|\mathbf{d_c}\right\|$, where $n \in \{-1, 1\}$ with -1 and 1 corresponding to no collision and collision respectively. A collision will result in negative collision distance as penetration distance. Denote J_0^t and J_0^i as robot Jacobian of tool and i_{th} link with respect to robot base frame respectively, and $J_{(p)}$ is the positional part of the Jacobian. For a robot, with a desired tool spatial velocity ν^* toward target pose, current joint position (q), the real-time collision avoidance is formed as

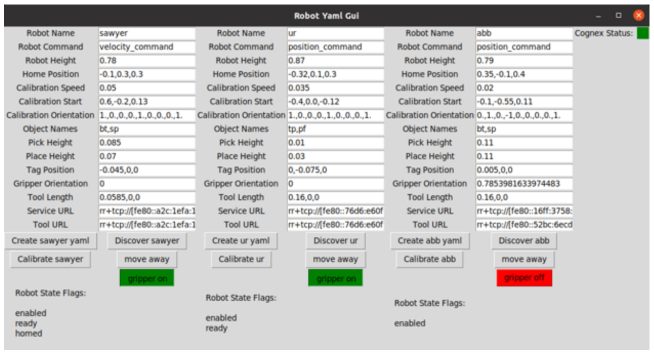
$$\begin{aligned} &\min_{\delta\mathbf{q}} \left\| J_0^t(\mathbf{q})\delta\mathbf{q} - \nu^* \right\|^2 + \left\|\delta\mathbf{q}\right\|_{W_q}^2 \\ \text{subject to} \quad & \\ &-\delta\mathbf{q}_{\max} \preccurlyeq \delta\mathbf{q} \preccurlyeq \delta\mathbf{q}_{\max}, \\ &\frac{\mathbf{d_c}}{\left\|\mathbf{d_c}\right\|} J_{0(p)}^i(\mathbf{q})\delta\mathbf{q} \le h(d). \end{aligned} \tag{1}$$

where $\delta\mathbf{q}_{\max}$ is the maximum size of a step size for joint position increments, $h(d)$ is a control barrier function depending on the collision distance d, i is the robot link number containing the closest point to environment, W_q is a weightfactor. The minimization problem is realized by quadratic programming (QP) between the current and desired pose with collision constraints [33], [34]. The real-time position command for the robot is

$$\mathbf{q}_{cmd} \leftarrow \mathbf{q} + \delta\mathbf{q} \tag{2}$$

By taking advantage of a centralized planner, each robot can utilize the same planner in real-time with synchronized updated joint positions to prevent possible collision.

D. Client Interface


Robot Yaml Gui

Robot Name	sawyer	Robot Name	ur	Robot Name	abb	Cognex Status: 🟩
Robot Command	velocity_command	Robot Command	position_command	Robot Command	position_command	
Robot Height	0.78	Robot Height	0.87	Robot Height	0.79	
Home Position	-0.1,0.3,0.3	Home Position	-0.32,0.1,0.3	Home Position	0.35,-0.1,0.4	
Calibration Speed	0.05	Calibration Speed	0.035	Calibration Speed	0.02	
Calibration Start	0.6,-0.2,0.13	Calibration Start	-0.4,0.0,-0.12	Calibration Start	-0.1,-0.55,0.11	
Calibration Orientation	1.,0.,0.,0.,1.,0.,0.,0.,1.	Calibration Orientation	1.,0.,0.,0.,1.,0.,0.,0.,1.	Calibration Orientation	0.,1.,0.,-1,0.,0.,0.,0.,1.	
Object Names	bt,sp	Object Names	tp,pf	Object Names	bt,sp	
Pick Height	0.085	Pick Height	0.01	Pick Height	0.11	
Place Height	0.07	Place Height	0.03	Place Height	0.11	
Tag Position	-0.045,0,0	Tag Position	0,-0.075,0	Tag Position	0.005,0,0	
Gripper Orientation	0	Gripper Orientation	0	Gripper Orientation	0.7853981633974483	
Tool Length	0.0585,0,0	Tool Length	0.16,0,0	Tool Length	0.16,0,0	
Service URL	rr+tcp://[fe80::a2c:1efa:1	Service URL	rr+tcp://[fe80::76d6:e60f	Service URL	rr+tcp://[fe80::16ff:3758:	
Tool URL	rr+tcp://[fe80::a2c:1efa:1	Tool URL	rr+tcp://[fe80::76d6:e60f	Tool URL	rr+tcp://[fe80::52bc:6ecd	

Robot State Flags:

enabled
ready
homed

Robot State Flags:

enabled
ready

Robot State Flags:

enabled

Fig. 10: Python User Interface for Pick and Place Task: users can monitor robots/sensor status, run vision-based calibration routine or manually override testbed information.

In order to better assist users in customizing different robots for the smart testbed, a simple Python-Tk interface is designed, allowing users to input testbed and robot information as shown in Fig. 10. This dedicated interface enables users to specify details such as robot height, calibration configuration, and RR service URL. The interface also includes service auto-discovery based on robot names, automatically filling in the URL section. In general, this interface serves the role as the testbed data storage and manipulation in a local `yaml` file, enabling the client to pick up configurable data and adapt to the smart testbed when executed.

E. Gazebo Simulation Interface

Robot simulation is often needed in both research and industrial areas, with Gazebo being a popular choice [35]. Typically, Gazebo is used together with ROS, but it is a standalone simulation software. In this project, we have developed an RR Gazebo plugin such that a standard RR robot driver service is able to interface with Gazebo directly (on both Ubuntu and Windows). As shown in Fig. 11, robot geometry definition files (`.sdf`) are used in the scene to spawn robot models, and RR simulation robot drivers expose the standard robot service definition over the LAN. In this case, by using the same client for the actual testbed task can users achieve the behavior in Gazebo simulation.

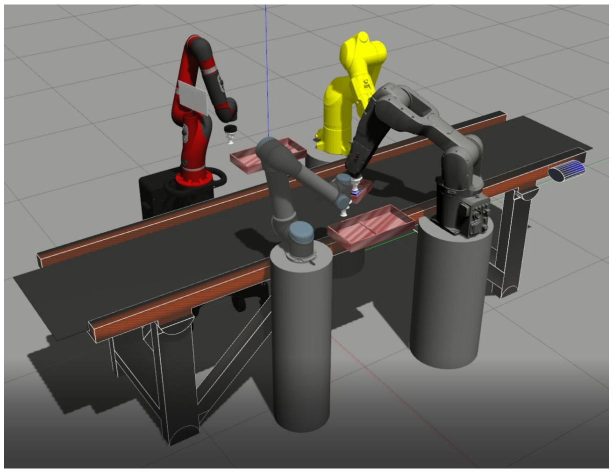


Fig. 11: Gazebo Simulation: robot models and environment objects are spawned with user-defined (`.urdf`) files, and RR Gazebo plugin controls the simulation joint angles while exposing them as a standard RR robot interface.

V. Alternative Solutions

This paper has presented a method to develop plug-and-play capability and interoperability with robots and sensors using RR, though implementing with ROS is also an alternative. ROS is widely used in the field of robotics, and robots equipped with ROS may be more accessible for existing ROS users. Some ROS-enabled robots like Sawyer and Tormach robots are running ROS in their internal controller, and `rostopic` and `rosservice` will be exposed to the LAN after configuring ROS `IP` and `MASTER_URI`. However, each ROS node is configured differently for different robots and may contain customized messages or service types, and users still need to read through the API to command the robots. If ROS-enabled robots share the same interface, then it is also possible to achieve plug-and-play capability demonstrated in this paper. Additionally, ROS does not support auto-discovery for nodes within the LAN, and the effort required for users to learn and set up ROS on a computer is significantly higher than with RR.

By standardizing robot definitions using RR, all robots sharing the standardized interface can be commanded with hassle-free plug-and-play capabilities. The effort to develop an RR driver for a new robot is also significantly lower than using ROS because RR is compatible with most platforms and programming languages. For any unique sensor or device not available in standard definitions, it is always possible to customize the definition for an RR service, allowing flexibility for researchers and users with specific needs.

VI. Conclusion and Future Directions

In this project, we have developed standard RR robot drivers for Sawyer, Baxter, UR series, ABB IRC5 series, Tormach ZA6, and Gazebo simulation robots. Sensor drivers include Webcam, Cognex, Kinect Azure, Realsense, and ATI F/T sensor. All these drivers follow standard definitions and offer complete plug-and-play capabilities for users with the same client code. Current supported devices and systems are listed in Robot Raconteur Directory [15], and more drivers will be developed in future projects.

Additionally, this paper demonstrates the plug-and-play capability using Robot Raconteur with our smart teachpendant for simple programming and a mock production line robotic system. Further development including Tesseract collision-free path planning, motion visualization, interactive debugging and mobile robot support will be key features for the smart teachpendant. In the future, we plan to add more functionalities to it, enabling users to easily set up the same mock production line system easily with the teachpendant with a motion planning plugin.

ACKNOWLEDGMENT

Funding for this research was provided by the ARM (Advanced Robotics for Manufacturing) Institute. The ARM Institute is sponsored by the Office of the Secretary of Defense under Agreement Number W911NF-17-3-0004. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of the Secretary of Defense or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] M. Jändel, "Plug-and-play robotics," in *NATO Symposium on Emerged/Emerging "Disruptive" Technologies*, 2011, pp. IST–099.

[2] N. Velásquez Villagrán, E. Estevez, P. Pesado, and J. De Juanes Marquez, "Standardization: A key factor of industry 4.0," in *2019 Sixth International Conference on eDemocracy & eGovernment (ICEDEG)*, 2019, pp. 350–354.

[3] OPC-Foundation, *Information Model*.

[4] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[6] ROS-Industrial, "Ros-industrial," 2022. [Online]. Available: https://rosindustrial.org/

[7] G. Bright and J. Potgieter, "Pc-based mechatronic robotic plug and play system for part assembly operations," in *IEEE International Symposium on Industrial Electronics. Proceedings. ISIE'98 (Cat. No.98TH8357)*, vol. 2, 1998, pp. 426–429 vol.2.

[8] S. M. Nguyen, C. Lohr, P. Tanguy, and Y. Chen, "Plug and play your robot into your smart home: Illustration of a new framework," *KI - Künstliche Intelligenz*, vol. 31, no. 3, pp. 283–289, Aug 2017. [Online]. Available: https://doi.org/10.1007/s13218-017-0494-8

[9] Ready Robotics , "Forgeos." [Online]. Available: https://www.ready-robotics.com/solutions/forgeos

[10] J. D. Wason and J. T. Wen, "Robot raconteur: A communication architecture and library for robotic and automation systems," in *2011 IEEE International Conference on Automation Science and Engineering*, 2011, pp. 761–766.

[11] J. D. Wason, "Robot raconteur® version 0.8: An updated communication system for robotics, automation, building control, and the internet of things," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 2016, pp. 595–602.

[12] H. He, G. Saunders, and J. T. Wen, "Robotic fabric fusing using a novel electroadhesion gripper," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, 2022, pp. 2407–2414.

[13] D. Kruse, R. J. Radke, and J. T. Wen, "A sensor-based dual-arm telerobotic manipulation platform," in *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, 2013, pp. 350–355.

[14] ——, "Collaborative human-robot manipulation of highly deformable materials," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3782–3787.

[15] Robot raconteur directory. [Online]. Available: https://github.com/robotraconteur/robotraconteur-directory

[16] Pyri open source teach pendant github organization. [Online]. Available: https://github.com/pyri-tech

[17] J. Wason and J. T. Wen, "Robot raconteur®: Updates on an open source interoperable middleware for robotics," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, 2023.

[18] Robot raconteur (RR): An interoperable middleware for robotics. [Online]. Available: https://arminstitute.org/projects/robot-raconteur-rr-an-interoperable-middleware-for-robotics/

[19] Robot raconteur standard service definition types. [Online]. Available: https://github.com/robotraconteur/robotraconteur_standard_robdef

[20] "Cognex introduces video camera designed specifically for industrial machine vision applications," *Sensor Review*, vol. 19, no. 3, Jan 1999. [Online]. Available: https://doi.org/10.1108/sr.1999.08719caf.003

[21] G. Jocher, A. Stoken, and et. al., "ultralytics/yolov5: v3.0," Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3983579

[22] "Open source teach pendant programming environment," (accessed Mar. 15, 2023). [Online]. Available: https://arminstitute.org/projects/open-source-teach-pendant-programming-environment/

[23] "Spacemouse compact," (Accessed on Mar. 15, 2023). [Online]. Available: https://3dconnexion.com/us/product/spacemouse-compact/

[24] "The python restricted industrial (pyri)," (Accessed on Mar. 15, 2023). [Online]. Available: https://github.com/pyri-project/pyri-core/blob/master/README.md

[25] "Blockly: A javascript library for building visual programming editors." (Accessed on Mar. 15, 2023). [Online]. Available: https://developers.google.com/blockly

[26] "Serverless on aws." (Accessed on Mar. 15, 2023). [Online]. Available: https://aws.amazon.com/serverless/

[27] "Microsoft surface go 2." (Accessed on Mar. 15, 2023). [Online]. Available: https://www.microsoft.com/en-us/surface/business/surface-go-2

[28] "Raspberry pi 4." (Accessed on Mar. 15, 2023). [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[29] "Open source teach pendant jog, save, playback." (Accessed on Mar. 15, 2023). [Online]. Available: https://www.youtube.com/watch?v=9KSYgGpG8mk

[30] "Camera calibration using open source teach pendant." (Accessed on Mar. 15, 2023). [Online]. Available: https://www.youtube.com/watch?v=0Q6a07FSsBc

[31] "Vision guided collaboration using open source teach pendant." (Accessed on Mar. 15, 2023). [Online]. Available: https://www.youtube.com/watch?v=jF_BGaFI7Qc

[32] ROS-Industrial, "Tesseract," 2022. [Online]. Available: https://github.com/tesseract-robotics/tesseract

[33] H. He, C.-l. Lu, Y. Wen, G. Saunders, P. Yang, J. Schoonover, J. Wason, A. Julius, and J. T. Wen, "High-speed high-accuracy spatial curve tracking using motion primitives in industrial robots," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 12 289–12 295.

[34] S. Seereeram and J. Wen, "A global approach to path planning for redundant manipulators," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, 1993, pp. 283–288 vol.2.

[35] C. E. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, E. Krotkov, and G. Pratt, "Inside the virtual robotics challenge: Simulating real-time robotic disaster response," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 494–506, 2015.