2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

Towards a lightweight ROS–CoppeliaSim simulator for the BEATRIX robotic education platform

Janaardhanan Alagarraja[1], Adrian Rubio-Solis[2] and Uriel Martinez-Hernandez[1]

***Abstract*— This work presents the design and development of BEATRIXsim, a simulation environment for integration into the educational BEATRIX robot platform. BEATRIXsim is implemented in CoppeliaSim due to its support for kinematics and low computational load. Both the physical robot (BEATRIX) and digital robot (BEATRIXsim) are interfaced using an Arduino Mega microcontroller and the Robot Operating System (ROS) using the rosserial protocol. Custom-built ROS messages, services and a bridge are developed for synchronised operation, communication, control and real-time feedback of both the physical and digital robots. This simulation environment includes a Graphical User Interface (GUI) to allow users to simultaneously control both the physical hardware and the digital robot, but also their individual control. BEATRIXsim serves as a tool for experimentation and testing processes without the need for its physical counterpart, reducing risks, costs and development time. Overall, this work shows that an affordable humanoid robot for education purposes can be enhanced by the use of digital technology, providing a robust platform for real-time control, experimentation and teaching in robotics.**

I. INTRODUCTION

Humanoid robots have been a subject of fascination and innovation in robotics, primarily due to their potential applications in fields such as healthcare, education, research and human-robot interaction [1], [2]. Humanoid robots are equipped with a variety of sensors, actuators, and control systems that allow them to replicate human movements. Even though they are designed to mimic human movements and behaviours, humanoid robots present numerous challenges in terms of their mechanical complexity, the need for precise control systems and accurate simulation environment [3]. Robots typically offer a digital version or simulator of the physical robot. This digital technology is particularly useful to perform experiments and analyse the robot capabilities without the need to have the physical counterpart [4], [5], [6].

Robotics simulators –virtual representations of physical systems that are updated in real-time based on data from the actual robot [7]. They allow for the simulation, testing, and analysis of robot operations without physical wear and tear on the hardware [8], [9], [10]. By enabling virtual experiments, robotics simulators can significantly enhance the design, control, and optimisation processes.

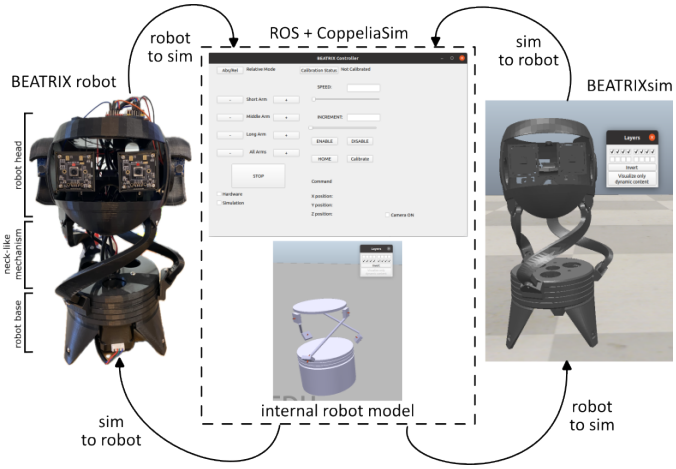


Fig. 1. BEATRIX robot and its simulator BEATRIXsim.

This work focuses on developing BEATRIXsim the first-ever digital robot or simulator for BEATRIX [11] –a platform for education in robotics (Figure 1). The aim of this robotic simulator is to enable real-time control and simulation of the physical robot. This simulation environment serves as a platform for testing and refining the robot behaviour in a safe, virtual environment before deploying it in physical scenarios.

BEATRIXsim is built in the simulation environment CoppeliaSim [12], [13]. The digital robot is interfaced with the physical robot using an Arduino Mega board and the Robot Operating System (ROS) [14], [15], [16]. Custom-built packages are implemented for communication of the robot actuators and sensors between the digital and physical robots. This allows to know the state of the physical and digital robot in real-time for precise control and feedback. A Graphical User Interface (GUI) implemented in the simulation environment allows the user to work either simultaneously or individually with the physical robot and its digital version for real-time communication, robot state and send control signals. BEATRIXsim is tested sending control signals, enabling/disabling functionalities, synchronising movements and observing the response of both the physical and digital robots. Overall, the experiments show that BEATRIXsim offers a robust and lightweight tool for experimentation without the need of the physical robot.

*This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) for the project (EP/X025470/1).

[1]Janaardhanan and Uriel are with the Multimodal inte-R-action lab, Department of Electronic and Electrical Engineering, Faculty of Engineering and Design, University of Bath, Claverton Down, Bath, UK {ja2351, u.martinez}@bath.ac.uk

[2]Adrian is with the Hamlyn Centre for Robotic Surgery, Imperial College London, London, UK a.rubiosolis@imperial.ac.uk

II. METHODS

A. BEATRIX robot platform

The physical BEATRIX humanoid robot, shown in Figure 1 is developed for educational purposes and to explore aspects of human-robot interaction. The robot is constructed using 3D printing technology and off-the-shelf components. BEATRIX features a neck mechanism with three degrees of freedom (DOF), enabling it to perform head movements in pitch, yaw, and roll orientations.

1) Mechanism overview: The robot mechanism features a set of components ensuring coordinated head motion (see Figure 2A). *Base and arm links*: The base of the robot serves as the anchor point for the neck mechanism. Attached to this base are three arm links, each equipped with internal gears. These arm links are connected to the base via spur gears, which are driven by stepper motors mounted on the base. *Connecting rods and skull mount*: Each arm link is connected to a twisted connecting rod, which is attached to the skull mount of the robot. This configuration creates a closed-loop kinematic system, ensuring that any movement in one arm link results in movement across all the connecting rods. *Gear mechanism*: Each stepper motor is connected to a spur gear, which drives the motion through an internal gear linked to the neck's arm. The driven spur gear has 26 teeth, while the driving internal gear has 68 teeth, resulting in a gear ratio of approximately 2.615:1. This mechanism ensures that both accurate and controlled head movement in any direction.

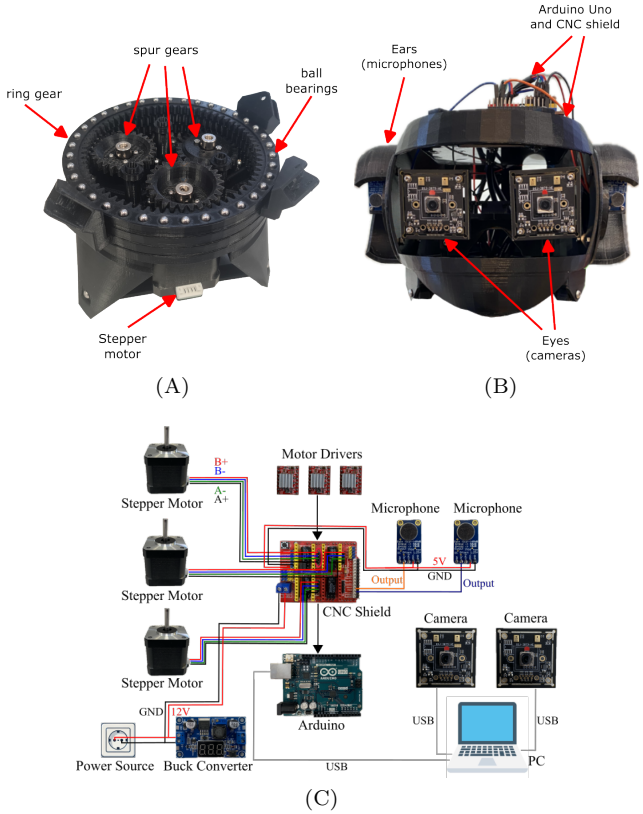


Fig. 2. BEATRIX robot platform. (A) Mechanical design. (B) Sensing components. (C) Control and communication.

2) Sensors and actuators: BEATRIX is equipped with sensors and actuators to interact with its environment (see Figure 2B). *Cameras*: Two RGB cameras are used for visual processing tasks. The cameras are configured to operate at 480 px resolution at 30 frames per second (fps). These cameras provide visual input for tasks such as object detection and facial recognition. *Microphones*: Two microphones are integrated in the robot, however, they have not yet been interfaced. Future development iterations will include software modules for auditory processing capabilities. *Stepper motors*: Three stepper motors (NEMA 17) allow the robot to perform reliable and precise movements. These motors provide a step angle of 1.8 deg per step, offering fine head movement control. The motors are connected to the robot's arm links via a CNC shield, which is mounted on the Arduino Mega, simplifying the wiring and integration process.

3) Controller and communication: The BEATRIX control system uses the Arduino Mega board with memory and processing power necessary to handle the demands of the robot control routines and integration with ROS. The stepper motors are driven via a CNC shield (A4988 driver) mounted on the Arduino Mega. Figure 2C illustrates the CNC shield and Arduino Mega, demonstrating the streamlined setup that allows an efficient stepper motors control [17]. The cameras are physically connected to the robot frame, with USB connections to the simulation computer. Communication between the computer and Arduino is done by the rosserial-arduino protocol, allowing real-time data exchange and control.

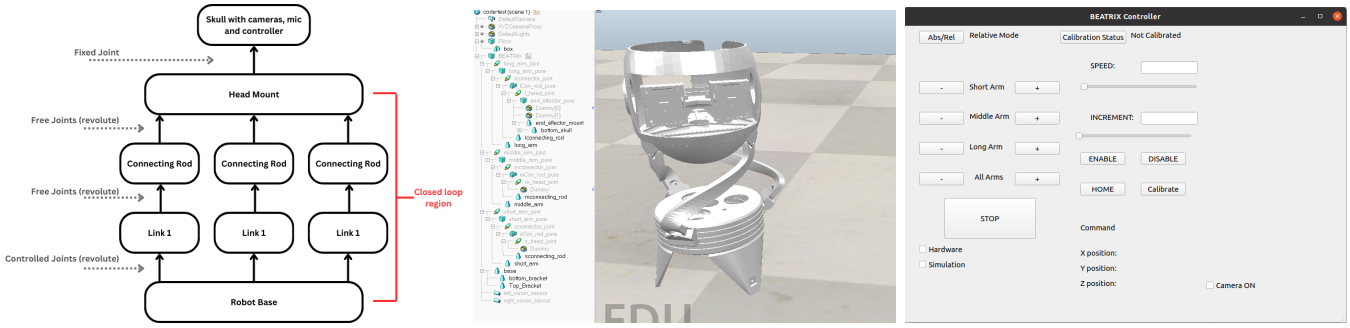
B. BEATRIXsim

1) Simulation platform: CoppeliaSim is used as the simulation platform due to its ability to handle closed-loop kinematics, a key feature of the robot's neck mechanism. Gazebo was considered initially, however, its higher computational overhead makes it less suitable for real-time simulation of BEATRIX. CoppeliaSim also provides an efficient platform for visualising and controlling the robot in real-time with reduced computational load.

2) Modelling in the simulation platform: The BEATRIXsim digital robot is modeled in CoppeliaSim by

(A) (B)

Fig. 3. BEATRIXsim 3D model in CoppeliaSim. (A) Visual model component. (B) Hidden model component.



(A) (B) (C)

Fig. 4. (A) Kinematic chain of the BEATRIX robot platform. (B) Model tree of BEATRIXsim in CoppeliaSim. (C) GUI for control of the interaction with the physical and digital robots.

importing its CAD representation and carefully constructing its mechanical relationships. The simulation model consists of two key components: 1) *the visual model* and 2) *the hidden model*, each serving a distinct purpose in the simulation process. Figure 3 shows the two models working together to create a balance between visual accuracy and computational efficiency. *Visual model*: This model represents the external appearance of BEATRIX and includes detailed geometries such as the neck, head and connecting rods. It is visible during the simulation for display purposes but does not interact with the physics engine or influence the robot's dynamics (Figure 3A). *Hidden model*: This model, used for collision detection and mechanical calculations, uses simplified cylinders and cubes geometric shapes. This reduces the computational load while maintaining the accuracy needed to simulate BEATRIX movements. The hidden model is not visible during normal operation but is essential for efficient simulation and accurate robot motion calculations (Figure 3B).

3) Joints and closed-loop kinematics: BEATRIX neck mechanism is modeled entirely with revolute joints. The base of the robot connects to three spur gears that revolve around the center of the base. Each gear is connected to an arm mounted at a 45 deg angle. The arms are placed 120 deg degrees apart from each other in the initial state. Moving all the joints together at the same speed causes the head to rotate left or right, while moving the joints at different speeds results in the head tilting in different directions.

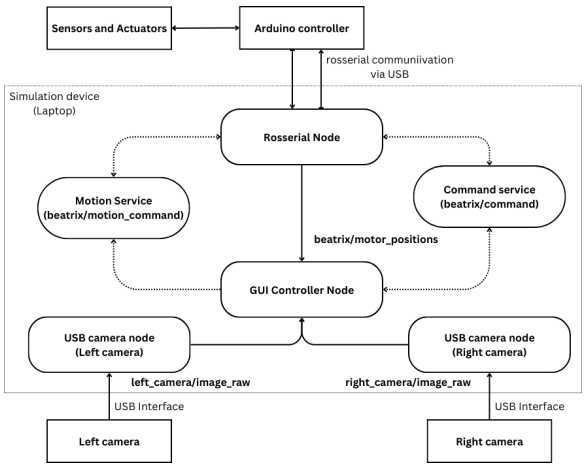
Revolute joints: The joints at the robot base are actively controlled in BEATRIXsim, driving the rotation of the arm links. These joints are controlled in position mode, meaning the a target position is provided, and the joint rotates to achieve the specified position. Figure 4A provides a schematic of the closed-loop kinematic chain, showing the interconnected joints. *Closed-loop kinematics*: The kinematics of BEATRIXsim is implemented using pairs of dummies in CoppeliaSim. Two pairs of dummies are placed at key connection points between the head mount and the connecting rods. The dummies are aligned to the same position at the start of the simulation and serve to constrain the motion of the connecting rods and the head mount in relation to the robot base movements. When the base joints rotate, the dummies ensure that the dependent joints (those in the connecting rods and the head mount) follow suit, accurately replicating the real-world motion of BEATRIX. *Kinematic chain*: The kinematic chain formed by the joints and dummies allows BEATRIXsim to simulate realistic motion. Only the three base joints are controlled, while the remaining joints respond dynamically to changes in position. This structure enables the simulation to run smoothly, with all joints and links moving in harmony. When running the simulation and providing target positions, the robot successfully moves as expected, demonstrating the efficacy of the closed-loop kinematic system. Figure 4B shows the model tree of the simulation in CoppeliaSim.

C. Software Architecture

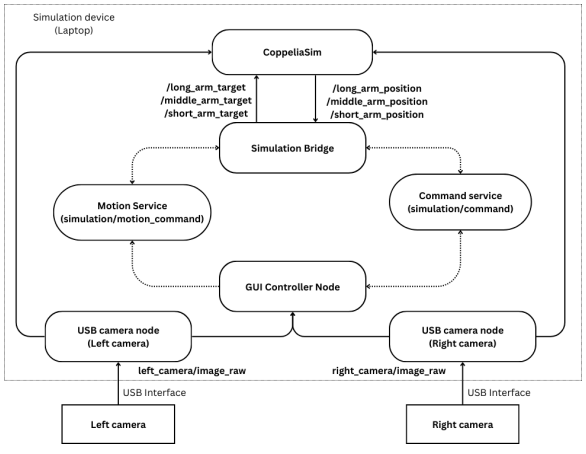
1) ROS environment: ROS environment forms the backbone of BEATRIX's control and communication infrastructure. The workspace is built on ROS Noetic, enabling seamless integration between the hardware, simulation, and the GUI. Custom packages, *beatrix_control* and *beatrix_messages*, are at the core of the ROS environment, handling the robot's motion commands, service requests and the communication between different components. Key ROS packages: *beatrix_control* manages the robot's control functions, launch files, and interface with both the simulation and the hardware. *beatrix_ messages* contains custom message types (Vector3.msg) and services (motionCommand.srv, commandService.srv) that define the commands and responses required for robot movement and operations.

2) ROS services and messages: To operate BEATRIX's motion and commands efficiently, custom ROS messages and services were developed, specifically *motionCommand.srv* and *commandService.srv*, along with the *Vector3.msg* message. These services handle movement operations and robot control commands, respectively, and play a key role in integrating BEATRIX with ROS. *Vector3.msg*: Custom ROS message representing a three-dimensional vector consisting of three fields: x, y, and z, each of type float32, which correspond to the X, Y, and Z motor axes, respectively. This message type is used

to define both the position and speed for BEATRIX's movement along these axes. *motionCommand.srv*: This service manages the movement of BEATRIX's joints. The parameters are: 1) absolute (boolean) which determines whether the movement is in absolute mode (target position) or relative mode (incremental movement); 2) position (Vector3) which defines the target position of each axis (X, Y, Z); 3) speed (Vector3) that specifies the speed for each axis. The service returns a boolean (result) indicating whether the motion command was successfully executed (true for success, false for failure). This allows for fine-grained control over the robot's movement, with both position and speed specified as vectors for each axis. *commandService.srv*: This service handles operational commands for BEATRIX, using an integer-encoded system (Int16) for command identification. The parameters include: 1) command (int16) that specifies the command to be executed; 2) value (boolean) which is used for commands that require an additional parameter, such as enabling or disabling the motors. The service returns a boolean (result) indicating the success or failure of the command execution.



(A)



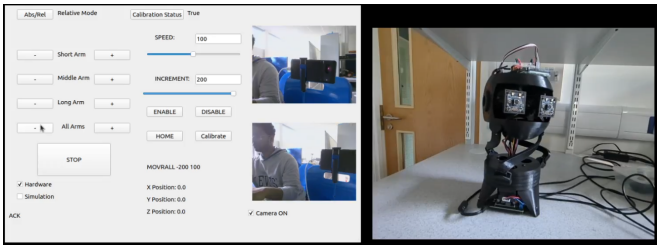
(B)

Fig. 5. (A) Data flow diagram for communication between the GUI and the physical robot. (B) Data flow diagram for communication between the GUI and the digital robot.

D. GUI Controller

1) GUI Layout: The graphical user interface (GUI) for control of BEATRIXsim is designed using PyQt5, with a layout that organises controls for joint movements, camera views and operation modes (see Figure 4C). The main window consists of the following components that provide real-time feedback of the robot status and control options. *Arm control buttons*: The GUI features dedicated buttons for incrementing and decrementing the position of each of the three arms (short, middle, and long) individually. The buttons marked with '+' or '-' adjust the respective arm's position, while the "All Arms" buttons control all three arms simultaneously. The arm positions are managed in either relative or absolute modes, depending on the selected mode. *Stop button*: The "STOP" button halts all movement of the robot's arms immediately. This is a critical control for emergency stops or to pause operations during manual control. *Speed and increment sliders*: The two sliders control the speed and increment values for joint movements. The user can adjust the sliders, and the selected values are shown in text boxes next to the sliders. These values directly influence how fast and far the joints move when a command is issued. *Calibration and home buttons*: The "Calibrate" button triggers a calibration process to reset the robot current position, preparing it for further operations. The "HOME" button returns all the joints to their default home position. The "ENABLE" and "DISABLE" buttons engage and disengage the motors, respectively. *CheckBoxes (Hardware, Simulation, Camera)*: When checked "Hardware" the GUI sends commands to the physical BEATRIX robot. When checked "Simulation" commands are sent to BEATRIXsim in CoppeliaSim. When "Camera ON" is checked, activates the camera views from BEATRIX's cameras, displaying live feed on the GUI. *Position and command labels*: Real-time positional feedback of the robot's X, Y, and Z coordinates is displayed in the labels, which update dynamically as the robot moves. Additionally, a label is provided to show the last command issued by the user.

2) Control flow and data flow: The control and data flow in BEATRIX's system manage real-time interactions between the GUI, hardware, and simulation. Commands from the GUI are relayed to either the Arduino for the real robot or to CoppeliaSim for the simulation. Feedback, such as joint positions and camera feeds, is returned to the GUI for monitoring. The system flow for communication between the GUI and hardware is illustrated in Figure 5A. The diagram shows all the hardware and software elements involved in the process like the ROS nodes, serviced, the hardware and topic based data flow. The control flow for the simulation is similar to the hardware but routed through the simulation bridge as shown in Figure 5B.



(A)

Fig. 6. Experiments of communication and control of BEATRIX (physical robot) using the GUI with ROS.

III. EXPERIMENTS AND RESULTS

A. Real robot control

1) Existing Firmware: The original firmware used for BEATRIX was built around simple string-encoded commands transmitted over serial communication. These commands controlled the stepper motors responsible for moving the neck joints. Upon receiving a command, the firmware parsed the string into individual components to determine the action to be taken. Commands were processed as follows: 1) The string received over serial was broken down into a 2D char array, with each command separated by spaces and carriage return characters; 2) Once the firmware detected a carriage return, it constructed the 2D array, where the first word (command) would be compared against a predefined list of valid commands; 3) If the command matched a valid entry, it was passed to the relevant function for execution. 4) There were a number of valid commands such as @MOVRX, @CALNOW, @MOVHOME, and @STOPALL, which handled motor movements, calibration, and system status.

2) Firmware Integration with ROS: The firmware was significantly restructured to handle commands through ROS services and messages, ensuring efficient communication and control of BEATRIX's joints. Motion and general operational commands are handled using two custom services: motionCommand.srv and commandService.srv. *Motion commands*: These are processed via the motionCommand.srv service. This service receives a boolean parameter (indicating absolute or relative motion) and vector parameters for the target position and speed along the X, Y, and Z axes. The firmware then translates these values into motor movements, precisely controlling BEATRIX's neck joints. The response is sent back as a boolean, indicating whether the command was successfully executed. *General commands*: Commands such as motor enabling, calibration and moving to the home position are handled via the commandService.srv. Each command is encoded as an integer, making it straightforward to add new commands in the future. The firmware handles these commands efficiently and returns a response (true or false) based on the success of the operation. *Real-time feedback*: The firmware publishes real-time feedback of the joint positions every second to the */beatrix/motor positions* ROS topic. This allows for continuous monitoring of the robot's state through the GUI or other ROS tools.

3) Testing and debugging the real robot: After integrating ROS into the firmware, several testing iterations were conducted to ensure that all functions performed correctly and the communication between ROS and the hardware was stable. Figure 6A shows the GUI operating the robot hardware and controlling the real robot.

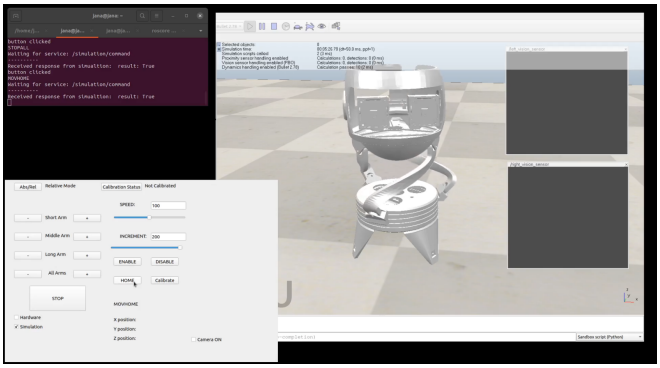
Initial testing: The initial tests involved validating the ROS services. First, the rosserial arduino communication was verified, followed by basic motion tests to ensure the ROS service was correctly received and executed by the firmware. ROS logs were monitored to capture any errors or missing communications during these tests. *Motor direction and command execution*: One of the initial bugs encountered was the reversed direction of the motors, where the joints moved in the opposite direction to the commanded position. This was quickly resolved by adjusting the sign in the motor control logic. Additionally, some commands were not responding due to missing service links, which were identified and fixed by inspecting the service request and response in ROS logs. *Improved feedback*: Compared to the original firmware, the integration of ROS provided more transparency in monitoring the robot's state. Acknowledgements for commands and feedback from the motors could now be directly displayed in the GUI, allowing users know when commands were successfully executed. *Final testing*: Once all services were validated, the final round of testing involved testing the robot under real-time conditions. All motion and operational commands were executed through the GUI, and feedback was continuously monitored. The new firmware proved to be more robust, responding accurately to commands and providing real-time feedback.

4) Real-time feedback and monitoring: The new firmware offers a real-time feedback loop that continuously monitors the positions of BEATRIX's joints. Every second, the firmware publishes the current positions to the */beatrix/motor* positions topic, ensuring that the ROS environment is always aware of the robot's state. This feedback loop enhances control precision and allows the GUI to display real-time updates on the robot's position. Users can observe the current position of each joint in the GUI, ensuring that BEATRIX is performing the commanded actions accurately.

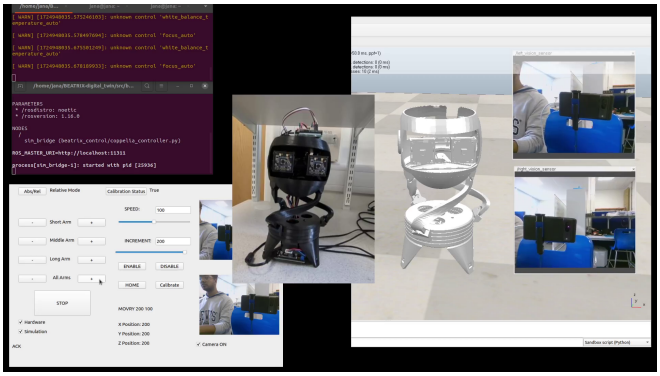
B. Control of the Digital Robot

1) Development of the Simulation Script: A Lua child script was developed and attached to the CoppeliaSim model of BEATRIX. This script was designed to control the movement of the robot's joints by setting a fixed target position. The initial test was successful, with the robot's joints moving from point A to point B and then back to point A. This Lua script showed to provide good stability for ROS integration in CoppeliaSim.

The Lua script subscribes to ROS topics */long arm target*, */middle arm target*, and */short arm target* each



(A)



(B)

Fig. 7. (A) Experiments for communication and control of BEATRIXsim (digital robot) alone using the GUI. (B) Experiments for communication and control of both the physical and digital robots using the GUI with ROS and CoppeliaSim.

of which receives a message of type std msgs/Float32 containing the radian values for the target position of the respective joint. Once a message is received the actions is triggered in CoppeliaSim, moving the joint to the specified target position at a predefined velocity. The movement of the robot's head in the simulation mirrors its physical counterpart. The connecting rods and head are affected by the movement of the arm links, with all joints moving in response to the command of any individual joint. This approach ensures that the closed-loop kinematic structure is properly simulated within CoppeliaSim. Testing the simulation script involved publishing motion commands to the ROS topics directly from the terminal and observing the joint movement.

2) Development of the simulation bridge: The next stage involved the development of the simulation bridge program, written in Python, which links ROS to CoppeliaSim. This bridge was required to ensure that BEATRIXsim can receive commands from the ROS network and respond with feedback, just like the real robot.

The bridge program subscribes to joint command topics and sends these commands to the CoppeliaSim model by setting the target positions of the joints. It publishes the current joint positions from the simulation back to ROS, providing real-time feedback on the robot's state. This feedback is essential to ensure that the movements in the digital model accurately reflect those of the physical robot. The simulation bridge operates by launching service servers for motion control and general commands. When a motion command is received from a ROS service client (via */simulation/motion* command), the bridge processes it and sends the appropriate target positions to CoppeliaSim.

3) Testing the digital robot control: Validation of joint movement: Commands for joint movements were sent via the GUI and terminal, and the corresponding joint movements in CoppeliaSim were monitored. The positions published in ROS were checked to ensure that they matched the target positions. Figure 7A shows an example of the simulation robot successfully tested using only the digital robot alone. *Synchronisation with real robot*: One of the critical tests involved comparing the movements of the physical robot with those in the simulation environment in CoppeliaSim. An example of the testing setup and results of controlling both the digital and physical robots is shown in Figure 7B. The system is designed such that the simulation in CoppeliaSim moves first, providing the user with a visual preview of how the real robot will behave. Then, the command is sent to the physical robot, which executes the same movement. Although both the digital and physical robots perform the same movements, the simulation typically moves slightly ahead of the physical robot. This is due to differences in processing speed between the software simulation and the hardware execution. The time difference is intentional, as it allows for early detection of any potential issues in the robot's movements, providing an opportunity to abort or modify the command before it is sent to the real hardware.

Feedback accuracy: The real-time feedback loop was tested by continuously publishing the joint positions from the simulation. This feedback was compared against expected values to ensure that the simulation accurately reflected the commands sent by ROS. These tests confirmed that the digital robot in CoppeliaSim functioned as expected, providing a reliable platform for simulating BEATRIX's movements and control behaviours.

IV. CONCLUSIONS

The development of BEATRIXsim has successfully created a functional platform, combining real-time simulation with physical control, for robotics education. Using CoppeliaSim and integrating ROS has established an efficient communication bridge between the physical robot and its simulation. This enables seamless control via a GUI, allowing real-time tracking and comparison of both the digital and physical systems. The final system provides an adaptable platform for further improvements, such as sensor integration for precise feedback and better synchronisation between the physical and digital robots. Overall, this work lays the foundation for the continued development of the BEATRIX robot education platform.

References

[1] T. Dieu, T. Ngo, K. Capek, elektischen Orgel, and B. Meschen, "Humanoid robot," *International Journal of Advanced Research in Science, Communication and Technology*, 2023.

[2] D. J. v. d. J. Michelle M. E. van Pinxteren *et al.*, "Trust in humanoid robots: Implications for services marketing," *Journal of Services Marketing*, vol. 33, no. 4, pp. 289–302, 2019.

[3] M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita, "Interactive humanoid robots for a science museum," in *IEEE Intelligent Systems*, vol. 22, 2006, pp. 25–32.

[4] D. M. Botín-Sanabria *et al.*, "Digital twin technology challenges and applications: A comprehensive review," *Remote Sensing*, vol. 14, no. 6, p. 1335, 2022.

[5] D. Wagg, K. Worden, R. Barthorpe, and P. Gardner, "Digital twins: State-of-the-art and future directions for modeling and simulation in engineering dynamics applications," *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, 2020.

[6] H. Orsolits, S. F. Rauh, and J. G. Estrada, "Using mixed reality based digital twins for robotics education," in *2022 IEEE international symposium on mixed and augmented reality adjunct (ISMAR-Adjunct)*. IEEE, 2022, pp. 56–59.

[7] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks, "Characterising the digital twin: A systematic literature review," *CIRP Journal of Manufacturing Science and Technology*, vol. 29, pp. 36–52, 2020.

[8] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin modeling," *Journal of Manufacturing Systems*, vol. 64, pp. 372–389, 2022.

[9] M. Singh *et al.*, "Digital twin: Origin to future," *Applied System Innovation*, vol. 4, no. 2, p. 36, 2021.

[10] W. Kritzinger *et al.*, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018.

[11] G. A. Al, N. Hedworth, D. Tilley, S. Ahmed, R. Afeawo, and U. Martinez-Hernandez, "Beatrix: An open source humanoid head platform for robotics teaching and research," *HardwareX*, vol. 20, p. e00591, 2024.

[12] A. Farley, J. Wang, and J. A. Marshall, "How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion," *Simulation Modelling Practice and Theory*, vol. 120, p. 102629, 2022.

[13] C. Forum, "Ros integration issues with python in coppeliasim," https://forum.coppeliarobotics.com/viewtopic.php?t=10319, 2023, accessed: 09. Aug, 2024.

[14] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[15] O. Robotics, "Robot operating system (ros)," https://www.ros.org/, 2023, accessed: 08. Feb, 2024.

[16] Open Robotics. (2023) Robot operating system (ros). Accessed: 08.Feb, 2024. [Online]. Available: https://www.ros.org/

[17] M. R. Wilkinson, B. Castro-Dominguez, C. C. Wilson, and U. Martinez-Hernandez, "Low-cost, autonomous microscopy using deep learning and robotics: A crystal morphology case study," *Engineering Applications of Artificial Intelligence*, vol. 126, p. 106985, 2023.