Project Review and Closure Report

BRACCIO: VOICE-CONTROLLED ROBOTIC SYSTEM

Innovation Central Perth

Curtin University

(04/08/2025) - (20/10/2025)

Objectives of this report:

This report provides an overview of the voice-controlled Braccio robotic arm system, reviewing its design, development, and key functionality. It evaluates project performance and outputs against the original objectives, highlighting completed features and partially implemented elements such as the computer vision integration. Outstanding issues and limitations are identified, and recommendations for future development and deployment are provided.

Note: Section 6.2 contains critical handover details and should be reviewed carefully for future work and system continuity.

DOCUMENT ACCEPTANCE

Project Title Review and Closure Report is authorised for release once all signatures have been obtained.

PREPARED: Timothy Lance L. Tan, Bachelor of Science in Mechatronics Engineering

DATE: October 28, 2025

ACCEPTED: _____ DATE: _____

(<Industry Supervisor>)

Contents

Executive Summary

The project developed a voice-controlled Braccio robotic arm system to simplify pick-and-place operations, addressing the challenge of converting natural language commands into precise robotic movements without relying on cloud services. Made by the supervisor, the system captures spoken commands and translates them into structured instructions using a local LLM (Gemma 3:4B), ensuring intuitive and responsive human-robot interaction.

The solution employed a Raspberry Pi 5 to orchestrate command processing, GUI updates, and messaging, while the Arduino Braccio++ executed precise, collision-free servo movements. MQTT and serial communication provided reliable coordination between the Pi and the robotic arm. Key objectives, including voice command capture, LLM parsing, robotic execution, and real-time GUI feedback, were fully achieved. Computer vision integration via Edge Impulse was partially completed (model trained) but not deployed in the final system.

Hardware is stored with the academic supervisor at ICP, and software with full documentation is available at the GitHub Repository. The main motor may require maintenance, the gripper servo has been upgraded to PWM, and the second small servo is non-functional. Recommendations for future work include deploying the vision model, expanding pick-and-place sequences, enhancing GUI features, conducting robustness testing, and extending the modular system to integrate additional sensors or robotic arms.

Acknowledgements

I would like to express my sincere gratitude to Stanley, my project supervisor, for his guidance and support throughout this project. His expertise and work on the LLM integration formed a crucial part of the system's development.

Special thanks to Dan Marrable for his assistance with 3D printing and for providing valuable technical advice. I would also like to acknowledge Anna Schmidt for her support and coordination with the interns, and to all the Innovation Central Perth interns who generously shared their insights and offered helpful consultations during various stages of the project.

Lastly, I am deeply grateful to my family, relatives, and to God for their constant support, patience, and motivation that enabled me to complete this project successfully.

1. Introduction

1.1 Project Background and Scope

This project develops a voice-activated Braccio robotic arm capable of interpreting natural language commands to perform precise pick-and-place tasks. It aims to simplify human-robot interaction, making robotic control accessible without programming expertise.

Key stakeholders include engineering students and educators seeking a hands-on platform for human-robot interaction. The project focuses on:

- Voice control of the Braccio arm with natural language parsing.
- Modular communication using MQTT for flexible command handling.
- Collision-free pick-and-place operations.
- Future enhancements, such as computer vision with Edge Impulse, for autonomous object recognition.

The system emphasises modularity, responsiveness, and ease of use, providing a foundation for intuitive robotic control and educational research.

1.2 Project Requirements and Specifications

Functional Requirements:

- Interpret natural language voice commands and convert them into structured JSON instructions.
- Execute precise, collision-free pick-and-place operations with the Braccio robotic arm.
- Provide real-time feedback on task status and arm positions through a GUI.
- Support modular communication via MQTT between user input, processing, and hardware layers.

Performance Requirements:

- Low-latency command execution for responsive operation.
- Accurate voice recognition in typical indoor environments.
- Reliable sequential handling of multiple commands.

User Experience Requirements:

- Simple and intuitive voice-based interface for users without robotics expertise.

- Clear visual and textual feedback on task progress and completion.
- Scalability for future integration of computer vision or additional automation features.

1.2.1 Software and Tools Requirements

The project utilised several software tools and frameworks to enable voice-controlled robotic operation. Key selections include:

- Python: Primary programming language for its extensive libraries and ease of integrating voice recognition, MQTT communication, and GUI development.
- Google Speech API: Chosen for reliable, real-time voice-to-text conversion.
- Ollama with Gemma 3:4B: Local LLM used to parse natural language commands into structured JSON for robotic execution, avoiding cloud dependency.
- MQTT (Mosquitto): Lightweight messaging protocol enabling modular communication between the user interface, Raspberry Pi, and Arduino controller.
- Arduino IDE: Used for programming the Braccio servos and executing precise movements.

These tools were selected for reliability, modularity, and local processing capability, allowing seamless integration of voice input, command translation, and hardware control while maintaining low latency.

1.3 Constraints and Standards

The project operates within several non-functional constraints and standards:

- Cost and Resources: Limited budget for hardware components and development tools, requiring efficient selection of parts and software.
- Time: Development and testing must be completed within the project timeline to ensure timely demonstration and documentation.
- Safety and Reliability: The robotic arm must operate safely, avoiding collisions or unintended movements, and maintain consistent performance.

These constraints guided design decisions, balancing performance, usability, and practical feasibility throughout the project.

1.3.1 Initial Consultation and Literature Review

At the project's outset, consultations with supervisors and subject matter experts helped define the requirements, scope, and performance goals for a voice-activated robotic system. Research focused on:

- Voice-Controlled Robotics: Existing methods for speech recognition integration, including local versus cloud processing and command parsing strategies.
- Human-Robot Interaction: Best practices for intuitive, low-latency interfaces and feedback mechanisms.
- Modular Communication: Evaluation of lightweight messaging protocols like MQTT for scalable and reliable control between software and hardware layers.
- Robotic Arm Control: Studies on Arduino-based servo control, collision avoidance, and task sequencing for pick-and-place operations.

The literature review and expert guidance informed decisions on software tools, hardware selection, and system architecture, ensuring alignment with functional, performance, and usability requirements.

---

2. Design and Implementation Methodology (Part A)

2.1 Initial Conceptualisation (e.g., Wireframes, Sketches, Proof-of-Concept)

The project began with a conceptual design for a voice-operated robotic claw capable of picking up objects using computer vision and robotic control. Early sketches and block diagrams outlined the intended workflow: voice commands processed into actions, object detection for localisation, and coordinated claw movements via Arduino and Raspberry Pi.

The original plan leveraged Edge Impulse for object detection with a Luxonis OAK-D camera, training a model in Edge Impulse Studio for object recognition and localisation. Path planning and claw control were intended to run on a Raspberry Pi 5 hosting ROS 2 nodes, coordinating the Arduino Braccio++ firmware for grip and movement. Milestones included data collection, model testing, deployment, and full ROS-based pick-and-place operation.

- Voice Command --> Voice-to-Text (Google Speech API) --> Command Parsing / LLM --> Raspberry Pi 5 (ROS 2 Node Host) --> Object Detection & Localization (Luxonis OAK-D + Edge Impulse) --> Path Planning & Claw Control --> Arduino Braccio++ (Servo Movement / Grip) --> Pick & Place Operation
- This is heavily inspired by the ROS 2 Pick and Place System - Arduino Braccio++ Robotic Arm and Luxonis OAK-D - Edge Impulse Documentation.

Feedback from supervisors and initial tests prompted a shift away from ROS and full Edge Impulse deployment toward a simpler, MQTT-based architecture focused on voice command execution.

This refined approach maintained the original goal of intuitive robotic control while improving system modularity, responsiveness, and ease of deployment.

2.2 System Development and Iteration

The project initially planned to integrate ROS 2 and Edge Impulse for computer vision-based object detection and pick-and-place operations. Early development involved the Raspberry Pi 5 hosting ROS nodes, coordinating the Arduino Braccio++ firmware, and using the Luxonis OAK-D camera for localisation.

During prototyping, feedback and practical constraints prompted a shift to a simpler, MQTT-based architecture focused on voice command execution. This change reduced system complexity, improved responsiveness, and allowed modular communication between the user interface and the robotic arm.

The final development environment includes:

- Python for voice recognition, command parsing, and GUI.
- Google Speech API and Gemma 3:4B for local natural language processing.
- MQTT for lightweight, modular messaging between the user interface and the Raspberry Pi/Arduino control layers.
- Arduino IDE to control the Braccio++ servo movements.

This iterative development ensured a robust, responsive, and user-friendly system while maintaining flexibility for future enhancements, such as integrating computer vision or more complex task automation.

2.3 Data Management and Storage

The system uses file-based storage and lightweight data structures to manage operational and configuration data, without relying on heavy database systems.

Structured Storage:

- Purpose: Store configuration and system state information required for command execution and GUI updates.
- Implementation: JSON files and Python dictionaries on the Raspberry Pi.
- Examples of Data:
    - User-defined positions and arm slot mappings

    - Command sequences parsed from voice input
    - System status flags (e.g., “Busy”, “Done”)

Unstructured Storage:

- Purpose: Store raw or large-format data such as sensor readings, logs, or future media for computer vision experiments.
- Implementation: File-based storage (CSV, PNG, or JPEG) on the Raspberry Pi filesystem.
- Examples of Data:
    - Captured images for Edge Impulse model training
    - Depth or RGB-D camera streams (for potential localisation/vision integration)
    - Log files for debugging command execution or system performance

2.4 System Architecture and Integration

The final system architecture focuses on voice-controlled robotic operation with modular communication between the user interface, processing, and hardware layers. The architecture eliminates ROS and full computer vision integration, prioritising responsiveness, simplicity, and modularity.

Components and Communication:

1. User Interface (UI):
    - Runs on a PC with ucs_voice_wifi.py
    - Captures voice commands and converts them to text using the Google Speech API
    - Performs initial command parsing and validation via the local LLM (Gemma 3:4B)
2. Messaging Layer (MQTT Broker):
    - Acts as a hub for lightweight, modular communication
    - Publish commands to the robot and receive status updates
    - Topics include stacker/command, stacker/status, and stacker/positions
3. Processing and Control (Raspberry Pi):
    - Receives parsed commands and orchestrates task execution
    - Updates the GUI with live arm positions and system status

    - Sends validated movement instructions to the Arduino over serial

4. Hardware Layer (Arduino Braccio++):
    - Controls servo motors to execute pick-and-place movements
    - Performs collision-free motion sequencing and grip control
    - Sends acknowledgments back to the Raspberry Pi to update the status

2.4.1 Component Interfacing (e.g., Database Connection)

Components communicate using lightweight, local protocols without complex database systems. Key interfaces include:

- MQTT (Mosquitto): Handles command and status messaging between the UI, Raspberry Pi, and Arduino.
- Serial Communication (PySerial / Arduino Serial): Transmits validated movement instructions from the Raspberry Pi to the Braccio servos and receives acknowledgments.
- File-Based Storage (JSON/CSV): Stores structured configuration data and logs, accessed via Python's native file I/O libraries.

This setup ensures efficient, low-latency communication and easy integration of new components or data sources in the future.

2.4.2 Process/Script Execution

In the final system, the voice input application (ucs_voice_wifi.py) serves as the entry point for user commands. The workflow is as follows:

1. Voice Capture & Parsing: The user speaks a command, which is converted to text using the Google Speech API and processed locally by the Gemma 3:4B LLM to generate structured instructions.
2. Command Dispatch: Parsed commands are published via MQTT to the Raspberry Pi, which acts as the central controller.
3. Backend Processing: The Raspberry Pi runs Python scripts to:
    - Validate commands
    - Determine movement sequences
    - Update GUI status maps

4. Hardware Interaction: Validated instructions are sent over serial to the Arduino Braccio++, which executes servo movements and grip actions. The Arduino sends back acknowledgments to the Raspberry Pi, which updates the UI and MQTT status topics.

This architecture allows real-time, modular execution, with clear separation between voice input, processing, and robotic control, ensuring responsive and reliable operation.

2.4.3 Deployment/Hosting

The final system is deployed on dedicated embedded hardware for low-latency, standalone operation:

- Raspberry Pi 5: Hosts the Python scripts for voice command processing, LLM parsing, GUI updates, and MQTT communication.
- Arduino Braccio++: Executes servo control and grip actions based on instructions received from the Raspberry Pi via serial communication.
- Local MQTT Broker (Mosquitto): Runs on the Raspberry Pi to facilitate modular messaging between the user interface, processing scripts, and hardware.
- PC User Interface: Runs ucs_voice_wifi.py to capture voice input and display real-time system status.

This deployment ensures responsive, reliable control of the robotic arm without dependence on cloud services, while remaining modular for future enhancements.

2.4.4 API Topology

User Voice Command

--> Voice-to-Text (Google Speech API)

--> LLM Parsing (Gemma 3:4B)

--> Raspberry Pi Processing

--> Validate & Process Command

--> Update GUI

--> Publish Command via MQTT

--> Arduino Braccio++ (Servo Control)

--> Execute Pick & Place

--> Send Acknowledgment via MQTT

<-- Raspberry Pi receives Status Updates

<-- GUI shows Real-Time Positions & Status

---

3. Design and Implementation Methodology (Part B) / Specialised Methodology

3.1 Initial Research and Applicable Standards

Initial research focused on general engineering and safety standards relevant to robotic systems. Key considerations included the safe operation of servo-driven arms and modular software design.

3.2 Initial Assessment and Analysis

An initial feasibility and risk assessment evaluated the practical implementation of a voice-controlled Braccio robotic arm. The study considered:

- Servo movement precision to ensure safe, collision-free pick-and-place operations.
- Voice recognition accuracy in typical indoor environments for reliable command execution.
- System integration risks include MQTT communication failures, serial delays, and hardware-software coordination.
- Feasibility of processing LLM commands locally on the Raspberry Pi without cloud dependency.

3.3 Technical Safeguards and Solutions

The project implemented safeguards across three categories to ensure reliable and safe operation:

- Administrative/Procedural Solutions: Clear command protocols, stepwise task sequences, and user guidelines for operating the voice-controlled system.
- Technical Solutions: Local LLM parsing for voice commands, MQTT messaging for modular communication, and error handling in Python scripts to prevent command conflicts or misexecution.
- Physical/Hardware Solutions: Servo collision avoidance, secure mounting of the Braccio arm, and controlled workspace boundaries to prevent accidental contact or damage.

3.4 Legal and Ethical Considerations

The project ensured user privacy and responsible AI use, with voice data processed locally and no cloud storage, minimising legal or ethical risks.

3.5 Societal and Community Impact Considerations

The system was designed for educational and research use, providing an accessible platform for students and engineers to explore voice-controlled robotics safely and inclusively.

---

4. Results

4.1 Overview of Project Outcomes (Part A)

Highlight	Description
Computer Vision Model	Edge Impulse Studio trained model for object recognition and localisation (future use).
Camera System	Luxonis OAK-D RGB-D camera for capturing images used in model training.
Raspberry Pi Control	Python scripts handling command processing, GUI updates, and messaging.
Arduino Braccio++	Servo firmware executing collision-free pick-and-place actions.
Command Integration	Serial communication between the Raspberry Pi and Arduino ensures reliable execution.
Task Demonstration	Pick-and-place workflow validated for both robotic control and voice input.

4.2 Overview of Project Outcomes (Part B) / Specialised Outcomes

Highlight	Description
Voice Command Capture	ucs_voice_wifi.py captures user speech and converts it to text using Google Speech API.
LLM Parsing	Gemma 3:4B (via Ollama) locally interprets natural language commands and generates structured instructions.
Command Processing	Raspberry Pi scripts validate parsed commands and determine stepwise task execution.
GUI Updates	Live visualisation of robotic arm positions and system status on the Raspberry Pi.
Modular Communication	MQTT broker running on Raspberry Pi manages messaging between UI and Arduino.
Status Feedback	Raspberry Pi receives acknowledgments from Arduino and updates the GUI and MQTT topics accordingly.

4.3 Testing and Verification

Planned Objective	Achieved?	Key Contributing Factors or Challenges
Voice command capture and parsing	Yes	Google Speech API and Gemma 3:4B LLM accurately interpreted commands.
Collision-free pick-and-place by the Braccio arm	Yes	Arduino firmware and stepwise command execution ensured precision.
Real-time GUI updates of arm positions/status	Yes	Python GUI scripts correctly received and displayed the system status.
Computer vision integration for object recognition	No	Model trained in Edge Impulse but not deployed; ROS/vision setup not completed.

---

5. Discussion

5.1 Conclusion

The project successfully developed a voice-controlled Braccio robotic arm system capable of interpreting natural language commands and performing precise, collision-free pick-and-place tasks. Key objectives, including voice command capture, LLM parsing, robotic execution, and real-time GUI feedback, were fully achieved.

Although computer vision integration was partially completed (model training), it was not deployed in the final system and remains a planned enhancement.

Overall, the project demonstrated a functional, modular, and responsive system that meets its intended goals for intuitive human-robot interaction, providing a solid foundation for future improvements such as vision-based autonomy and advanced task automation.

5.2 Future Works and Recommendations

Future development of the project can focus on the following areas:

- Computer Vision Integration: Deploy the trained Edge Impulse model on the Raspberry Pi to enable autonomous object recognition and localisation.
- Advanced Task Automation: Incorporate more complex pick-and-place sequences, sorting criteria, or multi-step workflows.
- Enhanced User Interface: Improve GUI features with visual indicators for errors, task history, and voice feedback.
- Modularity and Scalability: Extend the architecture to support additional robotic arms or sensors, maintaining MQTT-based modular communication.

---

6. Closure Activities

6.1 Project Documentation and Archiving

The Braccio robotic arm, Raspberry Pi 5, and associated peripherals are currently stored with the academic supervisor at Innovation Central Perth (ICP). Note that the main motor may require maintenance, the gripper servo has been upgraded to PWM control, and the second small servo is not operational.

- GitHub Repository: https://github.com/InnovationCentralPerth/intelligent-pick-place-robot

Stage 1: Arduino & Braccio

Hardware:

- Braccio Robotic Arm (Arduino Braccio++)
- Main motor (may need maintenance)
- Gripper servo (upgraded to PWM)
- Second small servo (non-functional)
- Cables, power supplies, and mounting accessories
- Animal toys

Software:

- Serial communication scripts
- Command processing scripts (Arduino firmware integrated with Braccio)

Stage 2: Raspberry Pi

Hardware:

- Raspberry Pi 5
- Cables, power supplies, and mounting accessories (shared with Stage 1)

Software:

- Command processing scripts (Raspberry Pi)
- GUI application
- MQTT broker (Mosquitto)

- Backup copies
- Data files (images, JSON configuration, logs)

Stage 3: LLM / Voice Control

Hardware:

- Raspberry Pi 5 (shared with Stage 2 for hosting LLM processing)

Software:

- ucs_voice_wifi.py (voice command application)
- Gemma 3:4B LLM parsing scripts

Stage 4: Computer Vision

Hardware:

- Luxonis OAK-D Camera

Software:

- Trained Edge Impulse model (future integration)

6.2 Handover

Stage 1 Setup: Arduino & Braccio

Hardware Setup:

1. Prepare workspace: Place the black mat on a flat table to define the working area.
2. Position Braccio: Place the Braccio arm in the centre of the mat and secure it lightly with tape to prevent sliding.
3. Arrange animal holders: Place holders labelled A–F around the arm at reachable positions. The yellow tape mentions its placement; tape the holders on top of the yellow marks.
4. Connect power: Plug in the USB-C Braccio's power supply and ensure the lights are green in the Braccio Carrier Board.
5. Place animal toys: Position the animal toys in their starting locations according to the planned pick-and-place sequence.

Software Setup:

1. Open the Arduino IDE and download the Arduino_Braccio_plusplus library; select Arduino Nano RP2040 Connect as the board.
2. Upload Arduino firmware: Connect Arduino Braccio++ to a Computer and use a micro-USB to connect it to the Arduino Nano, upload Braccio_11_wloop.ino.
3. Connect the Raspberry Pi to a USB-C and turn it on, as the 5V is needed for the gripper servo.
4. Test servo movements: Open the Arduino IDE serial monitor and input the commands below to ensure each servo moves correctly and safely within its range.
    a. //A go to position A (A-F)
    b. //80,120,125,50,10 Move to position
    c. //A, C Go to position A to C (A-F)
    d. //X Go to Top
    e. //Y Close gripper, stay in position
    f. //Z Open gripper, stay in position
    g. //l Loop the claw

5. Verify pick-and-place: Use a simple sequence A, C to pick an animal at A and place it in the correct holder at C to confirm the system responds as expected.

Troubleshooting:

- If all the Braccio Carrier lights turn red/Braccio is stuck/Braccio displays “power not connected”/any problem with the claw
    - Unplug all power sources
    - Manually move the claw to an upright position (Error may be due to it being stuck)
    - Connect the USB-C
- If all the servos except for the gripper work
    - A possible issue may be the power source supplied to the micro-servo
    - Check that there is continuity from the Raspberry Pi and Arduino to the servo for power
    - Ensure that a common ground is connected with the Raspberry Pi and Arduino
    - Ensure that the Appropriate 5V and GND are connected to the corresponding micro-servo
- Testing Braccio Claw
    - Use the examples found in the Arduino_Braccio_plusplus library, such as the record and replay, and controlling servos manually
    - Note that the microservo gripper will not work with this library, as it is not controlled by the Braccio Carrier Board

Stage 2 Setup: Raspberry Pi

Hardware Setup:

- Prepare workspace: Use a flat, stable surface for the Raspberry Pi near the Braccio setup.
- Connect Raspberry Pi: Plug in the Raspberry Pi 5 with its power supply and connect to a monitor, keyboard, and mouse for setup.
- Connect to Arduino Braccio: Use a USB cable to link the Raspberry Pi to the Arduino Braccio++. Ensure secure connections.
- Network connection: Connect the Raspberry Pi to Wi-Fi for remote operation and MQTT communication.

Software Setup:

- Run the script on icp/animal_game/MQTT10/run.py
    - Also found in https://github.com/InnovationCentralPerth/intelligent-pick-place-robot
    - Make sure to download dependencies
- Six scripts will run; make sure that the two GUI scripts are seen
- Assign animals to the corresponding places A-F by using the GUI
- Test serial communication: Verify messages are published and received correctly between the Raspberry Pi and Arduino using the GUI.

Troubleshooting:

- Scripts not executing: Confirm Python version and dependencies are installed; check file paths.
- GUI not updating: Ensure scripts are running without errors, and the serial connection to Arduino is active.
    - Close all six windows and rerun the main script

Stage 3 Setup: LLM and Voice Control

Hardware Setup:

1. Microphone setup: Connect a USB or 3.5mm microphone to the Raspberry Pi for capturing voice commands.
2. Laptop: Ensure that the laptop has downloaded all the scripts.

3. Confirm internet connection: Ensure the Raspberry Pi and the Laptop are connected to the same Wi-Fi to allow LLM API or local model access.

Software Setup:

1. Launch voice command script: Run ucs_voice_wifi.py on the laptop to enable continuous voice listening.
    a. Speech recognition: The script converts spoken commands to text using a speech-to-text module.
    Command interpretation: The transcribed text is processed by Gemma 3:4B LLM parsing scripts, which convert natural language commands (e.g., “pick up the lion”) into structured actions.
2. Action execution: Parsed commands are sent via MQTT to the Raspberry Pi’s control scripts. Ensure that the Raspberry Pi received the JSON command through the second GUI.
3. The Braccio should follow the positions of the sent command.

Troubleshooting:

- No voice detected: Check microphone connection and default audio input settings (arecord -l in terminal).
- No arm response: Verify LLM output format matches command processor expectations and MQTT communication is active.
- Lag or freeze: Restart the LLM or speech module and check system RAM usage on the Raspberry Pi.

---

7. References

- https://docs.edgeimpulse.com/projects/expert-network/robotic-arm-sorting-arduino-braccio
- The LLM side where the computer includes voice recognition and parses the commands into the JSON format is made the supervisor, Stanley
-
- This project is supported by the AI models ChatGPT, Gemini, Claude, and Grok