Project Review and Closure Report

BRACCIO: VOICE-CONTROLLED ROBOTIC SYSTEM

Innovation Central Perth

Curtin University

(17/11/2025) - (03/02/2026)

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. 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: Dave Mitchell Qiu Masters of Computing in Computer Science

DATE: February 17, 2026

ACCEPTED: _____DATE: _____

(<Industry Supervisor>)

Contents

Executive Summary

This 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. The system captures spoken commands and translates them into structured instructions using a local LLM (Gemma 3:4B / GPT-4o-mini), ensuring intuitive and responsive human-robot interaction.

Recent developments have significantly upgraded the system's architecture to improve modularity, resilience, and hardware safety. The system now features a unified, environment-driven PC interface and operates on a headless Raspberry Pi (Ubuntu) environment using a local amqtt Python broker. The Arduino Braccio++ firmware has been recalibrated with a hardware-level safety lock to prevent motor shorting. Furthermore, a robust Vision module using the Luxonis OAK-D Lite has been established, featuring 3D SLAM and multi-model object detection, setting the stage for autonomous visual integration. Hardware is stored with the academic supervisor at ICP, and software with full documentation is available at the GitHub Repository.

Acknowledgements

I would like to express my sincere gratitude to Stanley, my project supervisor, for his guidance and support, particularly his work on the LLM integration. Special thanks to Dan Marrable for 3D printing assistance, Anna Schmidt for intern coordination, and the Innovation Central Perth interns for their insights.

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 for educational and research purposes. The scope has expanded to include a modular PC frontend, a headless Raspberry Pi backend, and advanced computer vision capabilities for future autonomous object recognition.

The scope 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 the OAK-D Lite, for autonomous object recognition.

1.2 Project Requirements and Specifications

Functional Requirements: Interpret natural language voice commands, execute collision-free pick-and-place operations, provide real-time GUI feedback, and support modular MQTT communication.

Performance Requirements: Low-latency command execution, accurate voice recognition, and reliable sequential handling of multiple commands.

User Experience Requirements: Simple unified interface (main.py), clear visual feedback, and environment-based configuration (.env) for scalability.

1.2.1 Software and Tools Requirements

The project utilised a highly specialized stack of software tools and frameworks to enable resilient, voice-controlled robotic operation and modular computer vision. Key selections and their justifications include:

- Operating System (Ubuntu 24.04): Chosen as the core operating system for the Raspberry Pi. Ubuntu 24.04 was selected specifically because it is the officially supported, native environment for ROS 2 Jazzy. Furthermore, it provides a highly stable, headless server environment necessary for running background robotic orchestrator processes without unnecessary graphical overhead.

- Python: Selected as the primary programming language for its extensive libraries and ease of integrating voice recognition, MQTT communication, and GUI development. It serves as the vital "glue" that bridges the high-level AI models with low-level hardware serial communication.

- Speech Engine (OpenAI Whisper): Upgraded from the original Google Speech API to a hybrid OpenAI Whisper architecture. This was chosen because Whisper provides superior, context-aware transcription accuracy for varied speech patterns. Its hybrid logic provides top-tier cloud accuracy while ensuring a resilient local fallback if the robot loses Wi-Fi connectivity.

- LLM Processing (Ollama with Gemma 3:4B / GPT-4o-mini): Selected as a local LLM used to parse natural language commands into structured JSON for robotic execution, avoiding cloud dependency. Using an LLM rather than traditional keyword-matching enables the robot to understand flexible, conversational intents instead of requiring the user to memorize rigid syntax.

- MQTT Broker (amqtt v0.11.3): Chosen as a lightweight messaging protocol enabling modular communication between the user interface, Raspberry Pi, and Arduino controller. The specific amqtt Python package was chosen to replace the system-level Mosquitto service, effectively bypassing systemd limitations in the new headless Ubuntu environment and allowing the broker to launch dynamically via subprocesses.

- Vision Stack (ROS 2 Jazzy, depthai_ros_driver, OpenCV, Rerun SDK): Chosen to maximize the capabilities of the OAK-D Lite camera. ROS 2 provides the industry-standard framework necessary for distributed 3D SLAM mapping. OpenCV handles standard bounding-box rendering, and the Rerun SDK was specifically chosen to provide a high-speed, low-overhead alternative for rapid point-cloud debugging without the heavy overhead of full ROS visualization.

- Arduino IDE: Used for programming the Braccio servos and executing precise movements. It is essential for compiling low-level C++ firmware, which allowed the team to successfully implement critical hardware-level safety locks (such as hardcoding the base motor position) to protect the compromised physical hardware.

1.3 Constraints and Standards

The project operated within significant hardware, software, and budgetary constraints. These limitations heavily dictated the system architecture, necessitating creative engineering workarounds to meet safety and reliability standards:

1. Compatibility Issues (OS, Visualization, 3D Mapping, and Vision AI): Integrating a modern robotics stack on a Raspberry Pi 5 resulted in severe dependency conflicts. Using Ubuntu 24.04 to natively support ROS 2 Jazzy created compatibility gaps with older, established Vision AI libraries and 3D mapping tools (rtabmap), which often expect older OS environments (like Ubuntu 22.04). Reconciling the AI inference engines, ROS 2 nodes, and visualization tools (like the Rerun SDK) required strict, isolated execution pipelines to prevent the system from crashing due to conflicting dependencies.

2. Outdated Arm Robot: The project utilized an aging Arduino Braccio++ arm, which suffers from physical degradation, including a compromised main motor and a non-functional secondary servo. Furthermore, this older hardware lacks modern closed-loop feedback (absolute encoders), meaning the robot cannot independently verify its physical position if bumped. This constraint necessitated strict safety standards, such as hardcoding the base position to 0 in the Braccio_11_wloop_reset.ino firmware to prevent electrical shorting and catastrophic hardware failure.

3. Weak 3D Mapping Device: While the Luxonis OAK-D Lite is capable of basic object detection, it is a low-power edge device with limited compute capability and depth fidelity. Pushing the device to handle complex 3D mapping caused power instability. To prevent the camera from crashing, the project had to enforce strict performance constraints: forcing USB 2.0 mode, artificially throttling the AI Shaves (processing cores) to 4/5, and disabling full-depth video streams to preserve bandwidth.

4. Lack of Gyroscope for 3D Mapping: The OAK-D Lite lacks a built-in Inertial Measurement Unit (IMU/Gyroscope). Robust 3D mapping (SLAM) algorithms heavily rely on gyroscope data to estimate camera motion between frames (odometry). Without hardware motion tracking, the system was forced to rely entirely on pure Visual Odometry (calculating movement by tracking shifting pixels). [Image demonstrating Visual Odometry drift in 3D mapping without an IMU] This makes the 3D map highly susceptible to spatial drift, tracking loss during rapid movements, and complete failure in environments with few visual features (like blank walls).

5. Time and Resources: Development, testing, and hardware mitigation had to be completed within a strict academic project timeline, requiring a balance between theoretical capabilities (like full ROS 2 autonomy) and practical, deliverable milestones (the finalized MQTT voice-control system).

1.3.1 Initial Consultation and Literature Review

Advanced Speech Recognition & Intent Parsing: Existing methods for speech integration were evaluated, contrasting cloud-only APIs with hybrid local architectures. Research focused on utilizing advanced engines like OpenAI Whisper for high-fidelity transcription, and replacing rigid hardcoded command triggers with local Large Language Models (Gemma 3:4B / GPT-4o-mini) to achieve dynamic, semantic intent understanding.

Headless Orchestration & Modular Communication: An evaluation of lightweight messaging protocols, specifically MQTT, for scalable and reliable control between software and hardware layers. Literature focused on implementing Python-based local brokers (amqtt) within headless, non-systemd Linux environments (Ubuntu 24.04) to completely decouple the user interface from backend hardware execution.

Hardware Mitigation & Kinematic Control: Studies on Arduino-based servo control, collision avoidance, and task sequencing for pick-and-place operations. Due to the physical degradation of the provided Braccio arm, research specifically targeted firmware-level safety implementations (such as hardcoding base positions) to prevent electrical shorting and catastrophic physical collisions.

Edge AI Mapping & Computer Vision: A review of literature regarding the deployment of 3D SLAM and object detection pipelines (MobileNet-SSD, custom TFLite) on low-power edge devices like the Luxonis OAK-D Lite. This included researching methods to mitigate visual odometry drift in the absence of an IMU/Gyroscope, managing hardware power-draw via USB 2.0 throttling, and resolving ROS 2 (Jazzy) network discovery limitations using static peering.

Human-Robot Interaction (HRI): Best practices for intuitive, low-latency interfaces and real-time feedback mechanisms. Research prioritized laying the groundwork for a unified "One-Terminal" CLI and preparing the underlying architecture for future conversational feedback loops.

---

2. Design and Implementation Methodology (Part A)

2.1 Initial Conceptualisation

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 block diagrams outlined a workflow where voice commands were processed into actions, integrated with object detection for localization, and executed via coordinated movements between a Raspberry Pi and an Arduino.

The original plan leveraged Edge Impulse for object detection using the Luxonis OAK-D camera, with the intention of hosting heavy ROS 2 path-planning nodes directly on the Raspberry Pi 5. This early design was heavily inspired by the ROS 2 Pick and Place System documentation utilizing the Braccio++ and OAK-D.

However, practical constraints—such as OS dependency conflicts, hardware power limitations on the Pi, and the need for a more responsive user interface—prompted a strategic pivot. Feedback

from initial testing directed the architecture away from a monolithic ROS/Edge Impulse deployment on the Pi toward a decoupled, headless MQTT-based architecture. This shifted heavy AI inference to a unified PC frontend while retaining a modular, parallel vision pipeline.

Updated Conceptual Workflow:

Voice Command --> Hybrid Speech-to-Text (OpenAI Whisper on PC)

--> Intent Parsing --> Local LLM (Gemma 3:4B / GPT-4o-mini)

--> Command Dispatch --> PC Frontend (main.py) publishes via MQTT

--> Core Processing --> Raspberry Pi 5 (Headless Ubuntu running local amqtt and run.py subprocesses)

--> Parallel Vision --> Luxonis OAK-D Lite (DepthAI/ROS 2 SLAM streaming UDP video to PC)

--> Hardware Control --> Arduino Braccio++ (Executing safety-locked servo movements via serial)

--> Completion --> Pick & Place Operation with MQTT status feedback

This refined conceptualization maintained the original goal of intuitive robotic control while vastly improving system modularity, responsiveness, and readiness for future multimodal integration.

2.2 System Development and Iteration

Development shifted away from heavy dependencies on the Pi toward a lightweight, environment-driven approach. The Raspberry Pi 5 was migrated to a headless Ubuntu setup. Because xterm and systemd are unsupported in this headless environment, the execution logic (run.py) was modified to launch all Python components (main.py, llm_sender.py) in the background as subprocesses. The PC side was consolidated into a single smart_stacker_frontend package driven entirely by a .env file, allowing easy switching between simulators and physical hardware.

The final development environment includes:

- PC Frontend & AI: Python-based unified CLI utilizing a hybrid OpenAI Whisper engine for speech recognition and GPT-4o-mini / Gemma 3:4B for local natural language processing.
- Core Orchestration (Raspberry Pi): Headless Ubuntu 24.04 executing Python background subprocesses to validate commands and calculate movement sequences without graphical overhead.
- Messaging Protocol: A local, Python-based amqtt broker providing lightweight, modular MQTT communication between the PC interface, the Pi's processing layers, and the Arduino.
- Computer Vision: Luxonis OAK-D Lite integrated via ROS 2 Jazzy and DepthAI APIs for 3D SLAM and object detection, streaming spatial data over UDP.
- Hardware Actuation: Arduino IDE utilized to compile and flash the safety-restricted Braccio_11_wloop_reset.ino firmware for precise servo control.

2.3 Data Management and Storage

The system uses file-based storage and .env configurations to manage operational data without relying on heavy databases. Structured storage (JSON files, Python dictionaries) handles system states and parsed sequences. Unstructured storage handles raw data like captured images, video streams, or log files.

2.4 System Architecture and Integration

The final system architecture is designed around a decoupled, highly modular framework that prioritises responsiveness, simplicity, and hardware safety. By moving away from heavy, monolithic ROS dependencies for core command execution, the architecture effectively separates the user interface, central processing, vision processing, and physical hardware control. These independent layers communicate seamlessly using lightweight messaging protocols (MQTT) and serial connections, allowing the system to operate robustly across a network while the core logic runs on a headless embedded environment.User Interface (PC Side): A unified CLI (main.py) handles text, voice, manual moves, and robot status.

User Interface (PC Side): A unified Command Line Interface (CLI) launched via main.py serves as the frontend. Driven entirely by a centralized .env configuration file, it handles natural language voice inputs (via OpenAI Whisper), text commands, manual move triggers, and live robot status monitoring from a single terminal.

Messaging Layer: A local Python-based amqtt broker (mqtt_broker.py) acts as the communication hub. Running on the Raspberry Pi, it handles lightweight, asynchronous message publishing and subscribing (e.g., topics like stacker/command and stacker/status) between the PC frontend and the Pi backend.

Processing (Raspberry Pi 5): Acts as the core orchestrator operating in a headless Ubuntu 24.04 environment. It receives structured JSON commands parsed by the LLM, validates the requests, calculates the necessary physical moves, and coordinates task execution by running Python component scripts (run.py, llm_sender.py) in the background as subprocesses.

Vision (Luxonis OAK-D Lite): A dedicated perceptual module operating on the Raspberry Pi. To ensure system stability and prevent power crashes, the camera is software-forced to USB 2.0 mode with AI Shaves reduced to 4/5. It runs ROS 2 Jazzy for 3D SLAM mapping (with local, remote, and fast-view Rerun SDK modes) and utilizes DepthAI/OpenCV for object detection (MobileNet-SSD and custom TFLite models) and Z-Depth distance calculations.

Hardware (Arduino Braccio++): The low-level physical actuation layer. It receives validated serial commands from the Raspberry Pi and executes the corresponding servo movements. To ensure operational safety and prevent electrical shorting due to hardware degradation, the Braccio_11_wloop_reset.ino firmware strictly restricts the base motor (index 0) to a fixed 0 position.

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

A command spoken into the PC frontend is processed by OpenAI Whisper. The text is parsed by the LLM into structured coordinates. The PC publishes this via MQTT to the Raspberry Pi's local amqtt broker. The headless Pi processes the task in the background, updating internal states and sending serial commands to the Arduino, which executes the physical movement and returns an acknowledgment.

The vision subsystem operates on a parallel data pipeline to prevent bottlenecking the core command loop. The Luxonis OAK-D Lite camera interfaces physically with the Raspberry Pi via a software-forced USB 2.0 connection. Raw spatial and visual data is processed locally on the Pi using the DepthAI API and ROS 2 nodes. To interface with the user's PC without relying on heavy ROS installations on the frontend, the headless Pi streams the processed video feeds (including object bounding boxes and depth heatmaps) over the local Wi-Fi network via lightweight UDP protocols. On the PC side, the vision/video_receiver.py script catches this UDP stream and utilizes OpenCV to render the live visual feedback. Additionally, for 3D SLAM mapping, ROS 2 nodes on the Pi communicate using DDS (Data Distribution Service) network protocols, relying on ROS_STATIC_PEERS configurations to successfully interface with remote visualization tools across network firewalls.

2.4.2 Process/Script Execution

In the upgraded system, the unified PC interface (main.py launched via run.bat) serves as the primary entry point for user commands. The streamlined execution workflow is as follows:

- Input & Parsing: The user speaks a command. The hybrid OpenAI Whisper engine transcribes the audio to text, which is then processed by the configured LLM (GPT-4o-mini or Gemma 3:4B) to generate structured JSON instructions.
- Command Dispatch: The parsed command is published by the PC frontend via MQTT to the Raspberry Pi's local amqtt broker.
- Backend Processing (Headless): The Raspberry Pi (running background scripts launched via ./start_game.sh) receives the command. It validates the request, calculates the necessary movement sequences, and updates internal status maps.
- Parallel Vision Execution: Simultaneously, the OAK-D Lite module runs its specific detection or SLAM scripts, streaming live visual feedback and depth data over UDP to the PC's video_receiver.py.
- Hardware Interaction: Validated movement instructions are sent over serial to the Arduino Braccio++. The Arduino executes the servo movements (adhering to the firmware safety locks) and sends an acknowledgment back to the Pi, which subsequently updates the UI and MQTT status topics.
- This architecture ensures real-time, modular execution, with clear separation between the AI processing on the PC, the headless orchestration on the Pi, and the low-level physical control on the Arduino.

2.4.3 Deployment/Hosting

The final system is deployed across a distributed, multi-device architecture to ensure low-latency performance, robust AI processing, and scalable standalone operation:

- PC Environment (Frontend & AI): Hosts the unified Python package (smart_stacker_frontend). It processes hybrid speech recognition (OpenAI Whisper), performs the LLM command parsing (GPT-4o-mini), manages central .env configurations, and runs the UDP video receiver.
- Raspberry Pi 5 (Headless Backend): Deployed on Ubuntu 24.04 without a physical display. It runs background Python scripts (launched via start_game.sh) to validate commands, orchestrate movement sequences, and operate the Luxonis OAK-D Lite vision pipelines.
- Local MQTT Broker (amqtt): A lightweight, Python-based local broker (mqtt_broker.py) running on the Raspberry Pi to facilitate modular messaging between the PC's AI frontend, the processing scripts, and the hardware.
- Arduino Braccio++ (Actuation): Operates on custom safety firmware (Braccio_11_wloop_reset.ino). It executes the precise servo control and grip actions based on validated serial instructions received from the Raspberry Pi.
- Vision Sensor (Luxonis OAK-D Lite): Deployed as a dedicated edge-AI peripheral connected to the Raspberry Pi via USB (software-forced to USB 2.0). It executes local spatial processing, 3D SLAM (via ROS 2 Jazzy), and object detection algorithms, streaming

lightweight UDP video and depth telemetry back to the PC to avoid overwhelming the headless Pi.

This distributed deployment ensures responsive, reliable control of the robotic arm. It effectively offloads heavy AI computation to the PC while maintaining a secure, modular, and headless embedded environment for real-time hardware and vision execution.

2.4.4 API Topology

(Primary Command & Control Flow:

User Voice Command / Text Input (PC Side)

--> Speech-to-Text (OpenAI Whisper Hybrid)

--> LLM Intent Parsing (GPT-4o-mini / Local Gemma 3:4B)

--> PC Frontend (main.py) publishes Command via MQTT (stacker/command)

--> Raspberry Pi Local Broker (amqtt) receives message

--> Raspberry Pi Headless Processing (Validate & Calculate Moves)

--> Send Serial Command

--> Arduino Braccio++ Firmware (Execute Pick & Place w/ Safety Locks)

--> Send Serial Acknowledgment

<-- Raspberry Pi receives Serial Status

<-- Raspberry Pi publishes Status via MQTT (stacker/status)

<-- PC Frontend receives Status Updates

<-- Unified CLI updates Real-Time Status Display

Parallel Vision Data Flow:

Luxonis OAK-D Lite (Camera Input on Pi)

--> Raspberry Pi Vision Module (ROS 2 SLAM / DepthAI Object Detection)

--> Stream Video & Depth Heatmaps via UDP

<-- PC UDP Video Receiver (video_receiver.py) captures stream

<-- PC displays live visual feedback and spatial data

---

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 distributed robotic systems and edge-AI integration. Key considerations included the safe operation of servo-driven arms—especially when dealing with degraded hardware components—and modular software design patterns for headless embedded systems (Ubuntu 24.04). Research also prioritized network resilience standards for decoupled systems, ensuring seamless communication between the PC frontend, the Raspberry Pi backend, and the OAK-D Lite vision sensor.

3.2 Initial Assessment and Analysis

An initial feasibility and risk assessment evaluated the practical implementation of a multi-node, voice-controlled robotic arm. The study considered several critical factors:

- Hardware Degradation Risks: Assessing the physical state of the Arduino Braccio++, specifically the main motor which requires maintenance, to ensure safe, collision-free pick-and-place operations without risking electrical shorting.
- Peripheral Power and Bandwidth: Evaluating the power draw of the Luxonis OAK-D Lite, noting that pushing maximum AI Shaves (processing cores) or streaming full-depth video over standard USB 3.0 led to system crashes on the Pi.
- System Integration and Headless Limitations: Analyzing the risks of transitioning from a desktop OS to a headless Ubuntu environment, specifically the absence of graphical terminals (xterm) and the unreliability of system-level services (like systemd for Mosquitto) for background execution.
- Voice Recognition Reliability: Balancing the superior accuracy of cloud-based APIs (OpenAI Whisper) with the necessity of operational uptime in typical indoor environments with unstable Wi-Fi.

3.3 Technical Safeguards and Solutions

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

- Administrative/Procedural Solutions: Centralizing all system variables (API keys, MQTT host IPs) into a secure .env configuration file to prevent hardcoded errors and allow rapid switching between simulated and physical robots. Streamlined setup scripts (setup.sh, start_game.sh) were introduced to enforce strict startup protocols.
- Technical/Software Solutions: * To bypass the lack of xterm on the headless Pi, execution logic was refactored to launch local Python components (including the local amqtt broker) as background subprocesses with non-interactive stdin handling.
- The speech engine was upgraded to a hybrid architecture: it defaults to high-accuracy cloud processing but automatically falls back to a local Whisper model if internet connectivity drops.

- To prevent vision-induced power failures, the OAK-D Lite was software-forced to USB 2.0 mode, AI Shaves were reduced to 4/5, and high-bandwidth data was routed via UDP to the PC rather than processing locally on the Pi.
- Physical/Hardware Solutions: Due to hardware wear on the main motor, a critical hardware-level safety lock was implemented in the Braccio_11_wloop_reset.ino firmware. The base position (index 0) is hardcoded to 0 (e.g., POS_A[6] = { 0, 100, 130, 75, 140, 0 }) to strictly anchor the arm and prevent electrical shorting or erratic movement..

3.4 Legal and Ethical Considerations

The project ensured user privacy and responsible data management through its modular architecture. While the system utilizes cloud-based OpenAI APIs for optimal speech-to-text accuracy, API keys are securely isolated within local .env files. Furthermore, the hybrid speech engine allows the system to operate entirely offline using a local Whisper model and local LLM (Gemma 3:4B / Ollama), ensuring zero voice data leaves the local network when maximum privacy is required

3.5 Societal and Community Impact Considerations

The system was designed for educational and research use, providing an accessible platform for students and engineers. By heavily decoupling the complex AI/Vision logic (running on the PC) from the physical actuation (running on the Pi/Arduino), the project serves as an inclusive, highly scalable blueprint. It allows researchers to experiment with advanced multimodal AI and 3D SLAM mapping safely, without needing to deeply modify the low-level embedded hardware.

---

4. Results

4.1 Overview of Project Outcomes (Part A)

Highlight	Description
Headless Command Processing	Python orchestrator scripts (run.py) run as background subprocesses on Ubuntu 24.04 to validate parsed commands and sequence physical tasks without graphical overhead.
Modular Communication	A lightweight, local amqtt Python broker facilitates reliable MQTT messaging between the PC frontend and the embedded hardware.
Edge Vision Processing	The Luxonis OAK-D Lite successfully runs ROS 2 Jazzy (3D SLAM) and DepthAI object detection, safely streaming heavy video data over UDP to avoid Pi crashes.
Hardware Actuation	The Arduino Braccio++ executes precise, collision-free pick-and-place actions, protected by a custom firmware lock restricting the base motor to a safe 0 position.
Status Feedback Loop	Serial communication between the Pi and Arduino ensures reliable execution; the Pi receives physical acknowledgments and publishes live status updates back to the PC via MQTT.

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

Highlight	Description
Unified User Interface	A single CLI entry point (main.py) replaces multiple scripts, allowing users to send voice commands, text, and manual moves from one terminal.
Voice Command Capture	Upgraded from standard APIs to a hybrid OpenAI Whisper engine that captures speech with high accuracy, falling back to a local model if offline.
LLM Parsing	GPT-4o-mini and local models (Gemma 3:4B via Ollama) interpret natural language commands and dynamically generate structured JSON instructions.
Centralized Configuration	An environment-based .env file manages API keys and MQTT host IPs, allowing seamless switching between a physical robot and a simulator without touching code.
Remote Vision Receiver	A dedicated UDP video receiver (video_receiver.py) successfully captures and displays live video and depth streams broadcasted by the Raspberry Pi.

4.3 Testing and Verification

Planned Objective	Achieved?	Key Contributing Factors or Challenges
Voice command capture and parsing	Yes	Upgraded to a hybrid **OpenAI Whisper** engine and **GPT-4o-mini / Gemma 3:4B** LLMs, successfully ensuring resilient, context-aware command interpretation even during network drops.
Collision-free pick-and-place by the Braccio arm	Yes	Re-calibrated **Braccio_11_wloop_reset.ino** firmware successfully implemented a hardware-level safety lock (restricting the base to a 0 position) to prevent electrical shorting from the degraded main motor.
Real-time remote UI updates of arm status	Yes	The headless Raspberry Pi successfully publishes background status updates via the local **amqtt** broker, which the PC-side unified CLI (main.py) receives and accurately displays.
Computer vision integration for object recognition	Partially	3D SLAM (ROS 2) and Object Detection (DepthAI) were successfully stabilized and deployed as standalone scripts within the **Vision/** module. However, fully autonomous integration with the main.py movement orchestrator remains a future objective.
Planned Objective	Achieved?	Key Contributing Factors or Challenges
3D SLAM mapping and spatial visualization	Yes (Standalone)	Successfully deployed ROS 2 Jazzy and rtabmap_ros using the OAK-D Lite. Overcame ROS network discovery issues using static peering. Hardware crashes were mitigated by forcing USB 2.0 mode, though the lack of a hardware IMU/gyroscope makes the system reliant on visual odometry, which is susceptible to spatial drift.

Centralized environment configuration and integration	Yes	Successfully migrated the PC frontend to a unified .env driven architecture (smart_stacker/config.py). By eliminating hardcoded variables, the system allows for instant, seamless switching between physical hardware and local simulators simply by modifying the MQTT_HOST IP.

---

5. Discussion

5.1 Conclusion

The project successfully developed a highly decoupled, voice-controlled Braccio robotic arm system capable of translating natural language commands into precise, collision-free pick-and-place tasks. Key core objectives were fully achieved, including resilient voice capture via a hybrid Whisper engine, dynamic intent parsing using advanced LLMs (Gemma 3:4B / GPT-4o-mini), and real-time remote status monitoring facilitated by a unified PC interface and a local amqtt messaging layer.

Crucially, the system was successfully migrated to a robust, headless embedded environment on the Raspberry Pi, featuring vital firmware-level safety locks to protect the degraded physical hardware. While the computer vision pipeline was not fully integrated into the main autonomous movement orchestrator, it advanced significantly beyond initial plans; it was successfully deployed as a stable, standalone module capable of processing ROS 2 3D SLAM and DepthAI object detection on edge hardware while streaming live telemetry over UDP.

Overall, the project demonstrated a functional, highly modular, and safe system that meets its intended goals for intuitive human-robot interaction. By establishing a clean, network-driven separation between heavy AI inference on the PC and low-level physical actuation on the Pi, the project provides a rock-solid, field-ready foundation for the ultimate goal: transitioning the system into a fully autonomous, multimodal robotic assistant.

5.2 Future Works and Recommendations

The ultimate goal of this project is to evolve the current voice-controlled stacker into an intelligent, multimodal, and fully autonomous robotic manipulator. Future development will transition the system through the following key upgrade phases:

1. Cognitive and Perceptual Upgrades

Multimodal Intent Understanding: Transitioning from text-based LLM parsing to multimodal models (e.g., Gemini 1.5 Pro, Llama 3 Vision) that process camera input and voice simultaneously. This enables real-time natural language grounding (e.g., understanding contextual commands like "No, the red one next to it").

Advanced Vision Pipeline: Replacing static object mapping with a dynamic computer vision pipeline. Using RGB-D cameras combined with YOLOv10 or Segment Anything (SAM 2) will allow the robot to detect unknown objects, estimate their pose, and calculate grasp points dynamically without pre-mapping.

Conversational Feedback Loop: Upgrading human-robot interaction (HRI) so the robot can ask clarifying questions (e.g., "I see two red blocks. Which one?") before executing tasks.

2. Architectural and Control Evolution

Dynamic Motion Planning: Moving away from step-based Arduino sequences to inverse kinematics (IK) and trajectory planning. Integrating ROS 2, MoveIt! 2, and Gazebo will allow the system to compute optimal, collision-free paths in real-time, relegating the Arduino to a low-level PWM controller.

ROS 2 DDS Network: Upgrading the local MQTT communication to a real-time, low-latency Data Distribution Service (DDS) network. This will support cloud offloading for heavy LLM inference while maintaining local control for safety, alongside a digital twin for live simulation monitoring.

High-Level Task Reasoning: Implementing an LLM-powered task planner (e.g., LangGraph) connected to a real-time world model. Instead of parsing single actions, the system will reason about complex goals (e.g., "Clean up the desk" by classifying items and determining their proper destinations).

3. Hardware Expansion and Mobility

Manipulator Enhancements: Upgrading from a basic servo Braccio arm to a 6-7 DOF robotic arm equipped with force, torque, and tactile sensors. This will allow for adaptive, soft gripping and mid-motion corrections based on physical resistance.

Autonomous Mobility: Mounting the manipulator onto a ROS-compatible mobile base . By combining LiDAR and Visual SLAM, the system will achieve full spatial navigation (e.g., "Bring me the cup from the kitchen").

Performance & Safety: Implementing AI-based real-time motion prediction for smoother actuation, soft-stop fail-safes, and dynamic power load balancing.

4. The Final Form: Intelligent Voice-Driven Manipulation System

The culmination of these upgrades will result in a fully autonomous, self-planning pick-and-place robot capable of complex human-like interactions. It will process commands like "Sort everything on the table by color, then bring me my phone," autonomously handling the perception, reasoning, and physical execution from start to finish.

---

6. Closure Activities

6.1 Project Documentation and Archiving

The Braccio robotic arm, Raspberry Pi 5, Luxonis OAK-D Lite, and associated peripherals are currently stored with the academic supervisor at Innovation Central Perth (ICP). Note that the main motor on the Braccio may require maintenance, the gripper servo has been upgraded to PWM control, and the second small servo is non-operational.

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

Stage 1: Arduino & Braccio Actuation Layer

- Hardware:

Arduino Braccio++ Robotic Arm

Main motor (degraded, requires cautious operation)

Gripper servo (upgraded to PWM control)

Second small servo (non-functional)

USB-C power supplies, mounting mat, and animal toys

- Software:

Braccio_11_wloop_reset.ino (Custom Arduino firmware with hardware-level safety lock restricting the base motor to index 0).

Stage 2: Raspberry Pi (Headless Backend)

- Hardware:

Raspberry Pi 5 (Operating in a headless Ubuntu 24.04 environment)

Shared cables and power supplies with Stage 1

- Software:

Local amqtt Python broker (mqtt_broker.py)

Background orchestrator scripts (run.py, launched via start_game.sh)

Serial communication handlers for Arduino interfacing

Internal state maps and JSON logs

Stage 3: PC Side (Unified Frontend & AI)

- Hardware:

Dedicated PC/Laptop (connected to the same local network as the Pi)

Microphone for voice capture

- Software:

Unified CLI package (smart_stacker_frontend launched via run.bat / main.py)

Centralized configuration framework (.env file)

Hybrid OpenAI Whisper engine for speech-to-text

Local/Cloud LLM parsers (Gemma 3:4B via Ollama / GPT-4o-mini)

video_receiver.py for rendering incoming UDP vision streams

Stage 4: Edge Vision Module

- Hardware:

Luxonis OAK-D Lite RGB-D Camera (Software-forced to USB 2.0 to prevent Pi power failure)

- Software:

ROS 2 Jazzy and depthai_ros_driver

3D SLAM mapping and Rerun SDK visualization (start_slam_local.sh)

Object detection pipelines (MobileNet-SSD and custom TFLite via start_distance.sh)

UDP streaming architecture for broadcasting telemetry to the PC

6.2 Handover

Stage 1 Setup: Arduino & Braccio

CRITICAL NOTICE: The instructions below provide a high-level operational overview of the system's setup. Because the system has been upgraded to a headless architecture with .env configurations, you MUST read the README.md file in the GitHub repository for the exact, step-by-step terminal commands required to launch each section.

Stage 1 Setup: Arduino & Braccio Hardware

- Hardware Setup:

Prepare workspace: Place the black mat on a flat table to define the working area.

Position Braccio: Place the Braccio arm in the centre of the mat and secure it lightly with tape to prevent sliding.

Arrange animal holders: Place holders labelled A–F around the arm at reachable positions. Tape the holders on top of the yellow marks.

Connect power: Plug in the USB-C Braccio power supply and ensure the lights are green on the Braccio Carrier Board.

Place animal toys: Position the animal toys in their starting locations.

- Software Setup:

Open the Arduino IDE and download the Arduino_Braccio_plusplus library; select Arduino Nano RP2040 Connect as the board.

Upload Arduino firmware: Connect the Arduino Braccio++ to a computer via micro-USB. Upload the custom Braccio_11_wloop_reset.ino firmware (which contains the critical safety lock for the base motor).

Power Gripper: Connect the Raspberry Pi to power via USB-C and turn it on, as the 5V rail from the Pi is required to power the upgraded PWM gripper servo.

Test servo movements: Open the Arduino IDE serial monitor and input the commands below to ensure safe movement:

A : Go to position A (A-F)

80,120,125,50,10 : Move to specific raw coordinates

A, C : Go from position A to C

X : Go to Top

Y : Close gripper, stay in position

Z : Open gripper, stay in position

- Troubleshooting:

Carrier lights turn red / Braccio is stuck / "Power not connected" error: Unplug all power sources. Manually move the claw to an upright, neutral position (it may be physically binding). Reconnect the USB-C power.

All servos work EXCEPT the gripper: This is a power issue with the upgraded micro-servo. Check continuity from the Raspberry Pi 5V pin to the servo. Ensure a common ground (GND) is connected between the Raspberry Pi and the Arduino.

Note: Standard Braccio library examples for the gripper will not work, as the upgraded micro-servo is no longer controlled by the Braccio Carrier Board.

Stage 2 Setup: Raspberry Pi (Headless Backend)

- Hardware Setup:

Prepare workspace: Place the Raspberry Pi 5 securely near the Braccio setup.

Headless Deployment: Connect the Raspberry Pi 5 to its power supply. Note: A monitor, keyboard, and mouse are no longer required for standard operation. * Connect to Arduino: Connect the Raspberry Pi to the Arduino Braccio++ via a standard USB cable.

Network connection: Ensure the Raspberry Pi is connected to the local Wi-Fi network.

- Software Setup:

Access the Pi via SSH (or local terminal if debugging). Navigate to the backend directory: /home/unknown/Rasberry Pi Side/

Execute the orchestrator script: run ./start_game.sh

This script will automatically launch the local amqtt broker and run the run.py background processing scripts without requiring graphical windows (xterm).

The system is now passively waiting for MQTT commands.

- Troubleshooting:

Scripts not executing: Confirm Python 3 and all pip dependencies (amqtt, pyserial) are installed in the Ubuntu 24.04 environment.

Serial Errors: If you see [Errno 2] No such file or directory: '/dev/ttyACM0', the USB connection to the Arduino is loose or unplugged.

Stage 3 Setup: PC Side (Unified Frontend & AI)

- Hardware Setup:

Microphone setup: Ensure a working microphone is connected to the PC/Laptop for capturing voice commands.

Network: Ensure the PC is connected to the exact same Wi-Fi network as the Raspberry Pi.

- Software Setup:

Configure Environment: On the PC, navigate to the /Computer Side folder. Open the .env file and set the MQTT_HOST variable to the local IP address of the Raspberry Pi.

Launch CLI: Run run.bat (or execute main.py via Python).

Voice Command Capture: Speak into the microphone (e.g., "Move the lion to position C"). The hybrid Whisper engine will capture the text, and the LLM (GPT-4o-mini/Gemma) will parse it into JSON.

Execution: The PC automatically publishes the command to the Pi via MQTT. The CLI will display real-time status updates as the physical arm moves.

- Troubleshooting:

No voice detected: Check the PC's default microphone input settings.

No arm response (Timeout): Ensure the MQTT_HOST IP in the .env file exactly matches the Pi's current IP address. Check that the Pi's firewall allows traffic on port 1883.

Stage 4 Setup: Edge Vision Module (OAK-D Lite)

- Hardware Setup:

Connect the Luxonis OAK-D Lite to the Raspberry Pi using a USB-C cable. Crucial: Ensure it is plugged into a USB 2.0 port (or software-forced to USB 2.0) to prevent the Pi from browning out.

- Software Setup:

On the Pi, navigate to the /home/unknown/Rasberry Pi Side/Vision/ directory.

Execute the desired vision pipeline (e.g., ./start_slam_local.sh for 3D mapping or ./start_distance.sh for object detection).

On the PC, run vision/video_receiver.py to catch the UDP stream and view the live camera feed and depth maps.

---

7. References

- Edge Impulse Documentation: ROS 2 Pick and Place System - Arduino Braccio++ Robotic

Arm and Luxonis OAK-D. Available at: https://docs.edgeimpulse.com/projects/expert-network/robotic-arm-sorting-arduino-braccio (Accessed: February 2026).

- Project Source Code: Innovation Central Perth. Intelligent Pick & Place Robot Repository. Available at: https://github.com/InnovationCentralPerth/intelligent-pick-place-robot
- Core Architecture Credit: LLM and Voice Parsing integration architecture developed in consultation with and guided by academic supervisor, Stanley.
- Software Frameworks: Documentation and frameworks utilized include ROS 2 Jazzy, Ubuntu 24.04 LTS, OpenAI Whisper API, and the Luxonis DepthAI platform.
- AI Support: Supported by the AI models ChatGPT