ICP Robotics

InnovationCentralPerth/intelligent-pick-place-robot: Develop a speech controlled pick and place robot with stereo machine vision using gen AI processing of commands to robot controller
Edge Impulse

- Voice Operated Robotic Pick-and-Place System
  - PC Frontend (AI & UI):
    - Unified CLI (main.py) driven by .env configurations.
    - Hybrid OpenAI Whisper (Speech-to-Text) + LLM Intent Parsing (Gemma 3:4B / GPT-4o-mini).
  - Raspberry Pi 5 (Headless Backend):
    - Ubuntu 24.04 running background Python orchestrators (start_game.sh).
    - Local Python MQTT Broker (amqtt) for modular messaging.
  - Hardware Actuation:
    - Arduino Braccio++ with safety-locked custom firmware.
  - Computer Vision (Standalone Edge Pipeline):
    - Luxonis OAK-D Lite (forced USB 2.0 to prevent Pi power failure).
    - ROS 2 Jazzy & DepthAI streaming over UDP.

==========

- Milestone 1.1: Installation & Architecture Pivot
  - Install Ubuntu 24.04 (Headless Environment) on Raspberry Pi 5 ✅
  - Install ROS 2 Jazzy & DepthAI for Vision pipeline ✅
  - Set up Python Virtual Environments and dependencies (amqtt, pyserial) ✅
  - Migrate away from monolithic ROS/Edge Impulse architecture to decoupled MQTT framework ✅

- Milestone 1.2: Hardware Mitigation & Serial Control
  - Test Braccio hardware and diagnose degraded main motor & servos ✅
  - Develop custom Arduino firmware (Braccio_11_wloop_reset.ino) ✅
  - Implement hardware-level safety lock (hardcoding Base = 0) to prevent shorting ✅
  - Upgrade and integrate PWM Gripper servo ✅
  - Establish reliable two-way Serial communication between Pi and Arduino ✅

- Milestone 1.3: Modular Communication & Headless Orchestration
  - Deploy local Python amqtt broker on Raspberry Pi ✅
  - Refactor backend execution to bypass xterm (run.py as subprocesses) ✅
  - Establish stacker/command and stacker/status MQTT topics ✅

- Milestone 2.1: Voice Capture & AI Intent Parsing
  - Integrate OpenAI Whisper for hybrid local/cloud speech-to-text ✅
  - Integrate Local LLM (Gemma 3:4B / GPT-4o-mini) for dynamic intent parsing ✅
  - Successfully parse natural language into structured JSON coordinates ✅

- Milestone 2.2: PC Interface & Environment Configuration
  - Develop unified frontend CLI (main.py) via run.bat ✅
  - Implement central .env configuration for dynamic IP/hardware switching ✅
  - Demonstrate end-to-end task execution: Voice -> LLM -> MQTT -> Pi -> Arduino ✅

- Milestone 3.1: Computer Vision (Standalone Deployment)
  - Configure OAK-D Lite (USB 2.0 mode, 4/5 Shaves) to stabilize hardware power ✅
  - Deploy ROS 2 3D SLAM mapping locally (start_slam_local.sh) ✅
  - Deploy Object Detection pipeline (start_distance.sh) ✅
  - Establish UDP streaming architecture and PC-side video_receiver.py ✅

- Milestone 3.2: Full Vision-to-Motion Autonomy (Future Endgame)
  - Merge Vision spatial mapping dynamically with LLM movement orchestration ❌
  - Generate dynamic trajectory planning / IK via ROS 2 MoveIt! 2 ❌
  - Achieve fully autonomous object discovery and grasping ❌

==
ROBOTICS PICK AND PLACE PROJECT - SESSION CONTEXT
==
Date: November 17, 2025
Platform: Distributed System (PC + Headless
Raspberry Pi 5 on Ubuntu 24.04)
User: icp
Working Directory: /home/icp/ & PC /Computer Side/

==
PROJECT OVERVIEW
==
Developing a modular, voice-operated robotic pick-and-place system using:

- Dedicated PC/Laptop as the AI frontend (Whisper STT + Gemma/GPT LLM)
- Raspberry Pi 5 acting as a headless orchestrator and local MQTT broker
- Arduino Nano RP2040 Connect on Braccio Carrier for physical manipulation
- Luxonis OAK-D Lite camera for edge computer vision and 3D SLAM
- ROS 2 Jazzy for standalone vision processing and UDP streaming

==
HARDWARE SETUP STATUS
==
✅ PC Environment (Frontend & AI)

- Unified CLI (main.py) driven by centralized .env configuration
- Hybrid OpenAI Whisper (Speech-to-Text) functional
- Local LLM (Ollama/Gemma 3:4B) and Cloud LLM (GPT-4o-mini) functional
- Network: Connected to local Wi-Fi (Target IP configured in .env)

✅ Raspberry Pi 5 (Headless Backend)

- OS: Ubuntu 24.04 LTS (Native environment for ROS 2 Jazzy)
- Execution: Headless background subprocesses (start_game.sh)
- Messaging: Local amqtt Python broker active on port 1883
- Network: Connected to local Wi-Fi

✅ Arduino Nano RP2040 Connect + Braccio Carrier

- Hardware detected via USB Serial (/dev/ttyACM0)
- Mounted on Braccio robotic arm kit (Main motor degraded)
- Firmware: Braccio_11_wloop_reset.ino successfully flashed
- Hardware Safety: Base motor successfully locked to index 0 to prevent shorting
- Gripper: Upgraded PWM micro-servo powered via Pi 5V rail

✅ OAK-D Lite Camera (Edge Vision Module)

- Hardware detected via USB-C
- Power Mitigation: Software-forced to USB 2.0 to prevent Pi power failure
- Compute Mitigation: AI Shaves successfully throttled to 4/5
- Capabilities: 3D SLAM and DepthAI Object Detection functional
- Telemetry: UDP spatial video streaming to PC receiver functional

✅ System Access & Control

- GUI completely decoupled; backend execution is entirely headless
- Unified PC CLI handles all remote command dispatch and status monitoring via MQTT
- SSH access enabled for Raspberry Pi backend management

==
SOFTWARE ENVIRONMENT STATUS

==========
PC Environment Workspace: /Computer Side/

- smart_stacker_frontend: ✅ Unified CLI and Python package functional
- .env configuration: ✅ Centralized API keys and MQTT IPs established
- video_receiver.py: ✅ UDP stream rendering ready
- Environment sourced via run.bat

Raspberry Pi Backend Workspace: /home/unknown/Rasberry Pi Side/

- mqtt_broker.py: ✅ Local amqtt Python broker functional
- Background execution: ✅ start_game.sh and run.py subprocesses ready
- Serial handlers: ✅ PySerial active for Arduino communication
- OS Environment: Ubuntu 24.04 (Headless server mode)

OAK-D Vision Workspace: /home/unknown/Rasberry Pi Side/Vision/

- ROS 2 Jazzy: ✅ Core installation complete and sourced
- depthai_ros_driver: ✅ Integrated for spatial mapping
- Rerun SDK: ✅ Fast-view visualization tools active
- Execution scripts: start_slam_local.sh, start_distance.sh

==========
CURRENT CAPABILITIES
==========

✅ Voice & AI Intent Parsing
- Hybrid OpenAI Whisper engine (Cloud + Local fallback) transcribing speech
- Local (Gemma 3:4B) and Cloud (GPT-4o-mini) LLMs dynamically parsing natural language
- Automatic generation of structured JSON coordinate commands

✅ Robot Communication & Control
- amqtt broker securely managing stacker/command and stacker/status topics
- PySerial transport configured (/dev/ttyACM0) for low-level Arduino execution
- Hardware-level safety lock actively preventing base motor shorting (Index fixed to 0)
- Upgraded PWM gripper fully integrated and responsive

✅ Edge Computer Vision
- 3D SLAM mapping operational via ROS 2 Jazzy
- Object detection (MobileNet-SSD and custom TFLite) functional via DepthAI
- Live UDP video and depth-heatmap streaming to PC Receiver (avoiding Pi GUI crashes)
- Hardware stabilized (Forced USB 2.0 and AI Shaves throttled to 4/5)

✅ Distributed Development Environment
- Fully decoupled architecture (Heavy AI on PC, Hardware Control on Pi)
- Zero-GUI headless execution on Raspberry Pi (No VNC or xterm required)
- Seamless switching between physical hardware and simulator via .env file

==========

NEXT DEVELOPMENT PHASES
==========

🔄 Phase 1: Vision Pipeline Integration
- Merge the standalone ROS 2 Jazzy / DepthAI vision modules with the main Python orchestrator
- Translate live UDP spatial telemetry and bounding boxes into actionable MQTT coordinate data
- Establish closed-loop visual feedback between the OAK-D Lite camera and the physical Braccio arm

🔄 Phase 2: Dynamic Path Planning & Kinematics
- Transition from hardcoded pick-and-place sequences (Positions A-F) to dynamic, real-time spatial coordinates
- Implement Inverse Kinematics (IK) for precise trajectory generation
- Enhance collision avoidance logic to utilize live 3D SLAM depth maps rather than static assumptions

🔄 Phase 3: Full Multimodal Autonomy
- Enable autonomous object discovery and grasping (e.g., the robot visually searches for the object before moving)
- Expand the LLM intent parsing to handle complex, multi-step reasoning based on live visual context from the camera
- Implement conversational AI feedback loops so the system can verbally update the user via the PC interface

🔄 Phase 4: Hardware Restoration & Optimization
- Perform physical maintenance on the degraded Arduino Braccio++ main motor to safely

remove the 0 index hardware lock
- Repair or replace the non-functional secondary small servo to restore full kinematic range
- Investigate integrating an external IMU/Gyroscope to mitigate visual odometry drift during 3D SLAM mapping

==========
KEY COMMANDS FOR CONTINUATION
==========
Launch Backend Orchestrator & MQTT Broker (Raspberry Pi):
cd "/home/icp/Rasberry Pi Side"
./start_game.sh
(This runs mqtt_broker.py and run.py as background subprocesses)

Launch AI Frontend & Voice Control (PC Side):
cd "/Computer Side"
Ensure the .env file has the correct MQTT_HOST IP, then run:
run.bat
OR manually:
python main.py

Launch Edge Vision / 3D SLAM (Raspberry Pi):
cd "/home/unknown/Rasberry Pi Side/Vision"
./start_slam_local.sh
OR for DepthAI Object Detection:
./start_distance.sh

Receive Live Vision Stream (PC Side):
cd "/Computer Side/vision"
python video_receiver.py
Access Raspberry Pi Headless Backend (Remote PC):
ssh icp@<RASPBERRY_PI_IP>

==========
IMPORTANT FILE LOCATIONS
==========
PC Frontend Workspace:
- /Computer Side/ - Unified CLI package and AI integration
- /Computer Side/.env - Centralized configuration (MQTT IPs, API Keys)
- /Computer Side/vision/video_receiver.py - PC-side UDP video streaming receiver

Raspberry Pi Backend Workspace:
- /home/unknown/Rasberry Pi Side/ - Headless orchestrator and local MQTT broker
- /home/unknown/Rasberry Pi Side/start_game.sh - Launch script for background subprocesses
- /home/unknown/Rasberry Pi Side/Vision/ - ROS 2 Jazzy and DepthAI standalone vision scripts

Arduino Actuation:
- Braccio_11_wloop_reset.ino - Custom Arduino firmware featuring the critical hardware-level safety lock (Base = 0)

==========
CONTACT & COLLABORATION
==========
GitHub Repository: https://github.com/InnovationCentralPerth/intelligent-pick-place-robot
Collaboration: This project architecture was developed collaboratively with AI assistance ChatGPT and guided by the academic supervisor at Innovation Central Perth (ICP). All future development sessions should reference the README.md for specific execution commands.
Last Updated: 17 February 2026

Session Status: Highly decoupled, headless architecture deployed. Voice-to-MQTT movement orchestrator fully functional. Edge Vision pipeline deployed standalone. Ready for future Phase: Vision-to-Motion integration.

🔍 Root Cause Identified: ONNX Model Compatibility on Edge Hardware

Earlier in the project lifecycle, timothy successfully completed Edge Impulse data collection (capturing over 450 images) and model training. However, deploying the model to the Luxonis OAK-D Lite failed because the camera strictly required an ONNX model format, which the standard Linux deployment did not readily support.

🛠️ The Vision Pivot (Workaround Implemented):
Instead of relying entirely on the Edge Impulse Linux runner, the architecture was pivoted to utilize native ROS 2 Jazzy nodes and the DepthAI C++ libraries.

Current Vision Status:
✅ OAK-D camera working perfectly via USB-C (Forced to USB 2.0 to prevent Pi crashing).

✅ Standalone ROS 2 3D SLAM mapping active.

✅ Custom ONNX/TFLite blob conversions utilized for object detection.

✅ UDP Video Streaming established to send heavy visual telemetry to the PC.

❌ Pending: Final integration of the standalone Vision pipeline into the main MQTT movement orchestrator for full autonomy

==========
SESSION UPDATE: Nov 17-18, 2025 - Handover & Initial Code Understanding
==========

🔄 COMPLETED IN SESSION:

✅ Project Handover: Conducted a full hardware inventory (Raspberry Pi 5, Braccio++ arm, OAK-D Lite) and secured access to the intelligent-pick-place-robot GitHub repository.

✅ Initial Baseline Review: Evaluated the inherited codebase, specifically reviewing the earlier attempts at Edge Impulse integration and the existing Arduino firmware logic.

✅ Hardware Diagnostics: Performed initial power tests on the Braccio Carrier board and noted the physical degradation of the main motor, establishing the need for future firmware-level safety constraints.

📋 KNOWN ISSUES:

⚠️ Hardware Degradation: The Braccio++ main motor is physically degraded and requires cautious operation. The secondary small servo is completely non-functional.

⚠️ Power Delivery: Initial tests showed that while 7.4V is transferred to the board, signal transfer to certain servos is inconsistent.

==========

SESSION UPDATE: Nov 24-25, 2025 - Deep Dive: Code Structure Analysis

==========

🔄 COMPLETED IN SESSION:

✅ Architectural Mapping: Traced the execution flow from the user's voice input down to the physical servo movements.

✅ Module Isolation: Identified the critical separation needed between the heavy AI processing (Speech-to-Text and LLM parsing) and the low-level hardware orchestration (serial commands to the Arduino).

✅ Communication Audit: Analyzed how the system currently handles messaging, paving the way for the implementation of a more robust MQTT broker (amqtt) to decouple the frontend from the backend.

📋 KNOWN ISSUES:

⚠️ Monolithic Bottlenecks: Hosting the entire ROS 2 stack, LLMs, and Voice APIs directly on the Raspberry Pi 5 causes severe latency and memory constraints.

⚠️ Legacy Examples: Standard Arduino_Braccio_plusplus library examples fail to work out-of-the-box due to hardware wear and the upgraded micro-servo not being controlled by the Braccio Carrier Board.

==========
SESSION UPDATE: Dec 1-2, 2025 - Code Integration & Bug Fixing
==========

🔄 COMLPETED IN SESSION:

✅ Pipeline Integration: Successfully merged the disparate Python modules (voice capture, LLM parsing, and serial communication) into a more cohesive execution loop.

✅ Debugging Serial Drops: Fixed code errors related to PySerial connection timeouts between the Raspberry Pi and the Arduino Nano RP2040.

✅ Message Routing: Refined the JSON payload structure being sent to the robot so the Arduino could consistently parse coordinate commands (e.g., moving from position A to C) without crashing.

📋 KNOWN ISSUES:

⚠️ Gripper Power Draw: The upgraded PWM gripper servo cannot be powered reliably by the Arduino board; it requires a direct 5V continuity line from the Raspberry Pi with a common ground.

⚠️ Serial Disconnects: If the USB-C connection between the Pi and Arduino is bumped, the system throws a [Errno 2] No such file or directory: '/dev/ttyACM0' error and requires a manual restart.
==========
SESSION UPDATE: Dec 8-9, 2025 - West Tech Festival
==========
🔄 COMPLETED IN SESSION:

✅ Event Participation: Attended the West Tech Festival. Project development was temporarily paused to accommodate networking, exploring emerging edge-AI trends, and showcasing initial robotics concepts.

==========
SESSION UPDATE: Dec 15-16, 2025 - Centralized Environment Setup (.env)
==========
🔄 COMPLETED IN SESSION:

✅ Hardcode Removal: Stripped out all hardcoded IP addresses, API keys, and local file paths from the main execution scripts.

✅ Environment Configuration: Implemented a centralized .env file structure (smart_stacker/config.py).

✅ Seamless Switching: This upgrade allows developers to instantly switch the system's target (e.g., pointing the AI to a physical robot vs. a local software simulator) simply by changing the MQTT_HOST parameter, drastically speeding up future testing.

==========
SESSION UPDATE: Dec 17, 2025 - Jan 4, 2026 - Christmas Holiday
==========
🔄 COMLPETED IN SESSION:
✅ System development officially paused for the academic holiday break.

==========
SESSION UPDATE: Jan 5-6, 2026 - ROS & Camera Setup (The Debian Hurdle)

==========
🔄 COMLPETED IN SESSION:

✅ Initial Deployment: Attempted to install the ROS 2 framework and the Luxonis DepthAI packages on the Raspberry Pi's native Debian Bookworm OS.

✅ Compatibility Wall: Encountered severe dependency conflicts. Discovered that the advanced 3D spatial mapping and vision packages required for the OAK-D Lite do not officially support the Debian environment.

✅ Architectural Pivot: Made the critical decision to wipe the Raspberry Pi and migrate the entire backend operating system to Ubuntu to ensure native ROS support.

📋 KNOWN ISSUES:

⚠️ OS Incompatibility: Debian Bookworm lacks native pre-compiled binaries for ROS 2 Jazzy and standard DepthAI ROS wrappers, causing constant build failures via colcon build.

==========
SESSION UPDATE: Jan 12-13, 2026 - OS Migration & Successful Vision Setup
==========
🔄 COMLPETED IN SESSION:

✅ Ubuntu Flashing: Successfully flashed and configured a headless Ubuntu 24.04 LTS environment on the Raspberry Pi 5.

✅ ROS 2 Jazzy Installation: Installed the native ROS 2 Jazzy framework, resolving the previous dependency nightmares.

✅ Camera Initialization: Successfully compiled the depthai_ros_driver. The OAK-D Lite camera is now properly recognized by the new OS and capable of streaming raw RGB and depth data.

📋 KNOWN ISSUES:

⚠️ Headless Execution Constraints: Because Ubuntu 24.04 was set up as a headless server, standard graphical terminals (xterm) and systemd user services for GUI apps are unsupported. Launch scripts (start_game.sh) had to be heavily modified to run Python components as background subprocesses.

==========
SESSION UPDATE: Jan 19-20, 2026 - AI Custom Object Detection Integration
==========
🔄 COMLPETED IN SESSION:

✅ Object Recognition Pipeline: Deployed the AI vision pipeline via DepthAI, enabling the camera to detect custom objects and draw bounding boxes.

✅ Depth Calculation: Configured the stereo depth sensors to calculate the Z-axis distance, allowing the system to identify which object is physically closest to the robotic arm.

✅ Strategic Adjustment: Planned the integration of a full 3D SLAM (Simultaneous Localization and Mapping) system to give the robot persistent spatial memory of its environment.

📋 KNOWN ISSUES:

⚠️ Model Format Conflicts: Edge Impulse downloads failed to provide the .ONNX model format strictly required by the OAK-D Lite. The workaround required using a C++ library and manually converting it to

ONNX via OpenVINO and Blobconverter, which is highly complex.

⚠️ Power Draw Spikes: Running complex object detection at maximum capacity causes the OAK-D Lite to pull too much power, crashing the Raspberry Pi. Mitigation: The camera must be software-forced to USB 2.0 mode and AI Shaves throttled to 4/5.

==========
SESSION UPDATE: Jan 26-27, 2026 - 3D SLAM Mapping Integration
==========
🔄 COMLPETED IN SESSION:

✅ SLAM Deployment: Successfully launched 3D spatial mapping (rtabmap_ros) using the OAK-D Lite's visual odometry.

✅ Data Exportation: Verified that the system can generate, export, and analyze 3D point-cloud scans of the physical workspace.

❌ Integration Status: While the 3D mapping runs beautifully as a standalone module, it is not yet dynamically tied into the LLM/MQTT movement orchestrator.

📋 KNOWN ISSUES:

⚠️ Visual Odometry Drift: The Luxonis OAK-D Lite lacks a built-in IMU (Gyroscope). Without hardware motion tracking, 3D mapping relies purely on visual odometry, leading to severe spatial drift in featureless environments (like blank walls).

⚠️ ROS Network Discovery: ROS 2 Jazzy struggles with dynamic node discovery across distributed networks without static peering configurations.

==========
SESSION UPDATE: Feb 2-3, 2026 - Final Presentation Setup & Handover
==========
🔄 COMLPETED IN SESSION:

✅ Documentation: Finalized the Project Review and Closure Report, detailing the decoupled architecture, hardware mitigations, and the successful .env integration.

✅ Demonstration Prep: Prepped the physical workspace (black mat, target positions A-F) and ensured the unified PC CLI (main.py) was ready for the live voice-controlled pick-and-place demonstration.

✅ Repository Update: Pushed the final code structures to GitHub and updated the README.md to ensure future students can seamlessly deploy the multi-stage system.

📋 KNOWN ISSUES:

⚠️ Vision-to-Motion Gap: The computer vision model successfully processes environment data, but dynamic path planning (generating new IK coordinates from vision instead of hardcoded A-F positions) remains a future milestone.

==========
SESSION UPDATE: End-to-End Integration ("Animal Game" Validation)
==========
🔄 COMPLETED IN SESSION:

✅ End-to-End Workflow Validation: Successfully executed the interactive "Animal Game" workflow. This tested the full pipeline from voice capture (speech_text.py) to the Arduino physical control (animal_ard.py).

✅ Hardware & Firmware Execution (Phase 1): Validated the custom Arduino serial commands, ensuring the arm safely executed point-to-point movements (e.g., A,C to move from position A to C), returned to top (X), and properly toggled the upgraded Jaycar 9g micro-servo gripper (Y to close, Z to open).

✅ MQTT Messaging Layer (Phase 4): Successfully established and tested the decoupled communication architecture. Demonstrated publishing complex JSON payloads (e.g., {"E": "L2", "L": "R2", "F": "L1"}) to the stacker/command topic, and successfully received physical state updates via stacker/positions and stacker/status.

✅ Phase Completion: Officially validated and marked all four core project phases (Arduino Firmware, CV Integration, ROS 2 Integration, System Integration) as completed ✅.

📋 KNOWN ISSUES:

⚠️ Terminal Clutter / Overhead: During this testing phase, the system required running up to six separate terminal windows simultaneously across the PC and Pi (for the camera, GUI, speech-to-text, and Arduino serial). Mitigation: This messy workflow directly prompted the final architectural upgrade to create the unified main.py frontend and the headless start_game.sh background orchestrator.

⚠️ Positional Mapping Shifts: The team experimented with shifting the physical workspace grid from simple letters (A-F) to a more directional layout (L1, L2, F, R1, R2, B). This requires strict, tedious recalibration of the Arduino's hardcoded inverse kinematic angles whenever the physical mat is moved.

⚠️ VNC Dependency: Initial UI testing relied heavily on VNC, which suffered from lag and high bandwidth usage. Mitigation: The UI was subsequently decoupled from the Pi entirely, shifting status rendering to the PC.