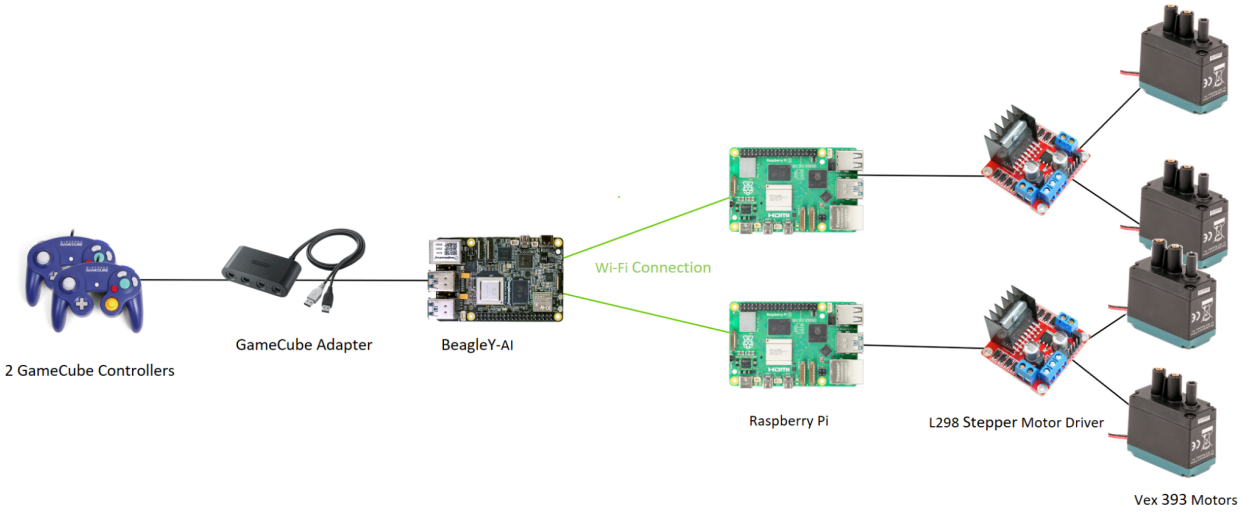
BattleBots Write Up

Our project implements a real-time, two-player "BattleBots" game bridging multiple embedded Linux platforms. A Beagle Y AI acts as the central game console, managing the match state and processing inputs from USB GameCube controllers. It commands two remote Raspberry Pi robots, where the objective is to hit a physical target button on the opponent's chassis. When a hit is detected, the target robot loses a life, and the score is instantly updated on the BeagleBone's OLED display.

The system functions as a closed control loop divided into three subsystems. The Console (BeagleBone) acts as the server and game engine. The Robots (Raspberry Pi) act as clients, translating network packets into motor actions. Bridging them is the Relay Host (Laptop), which routes traffic between the USB-connected Console and the WiFi-connected Robots. Data flows from the user inputs to the console, translates into commands sent to the robots, and physical hit events are reported back to update the game state.



Challenges and Resolutions

We encountered a critical issue where the BeagleBone could send data, but robot response packets were dropped at the Host PC interface. To resolve this, we built a UDP Forwarder tool (udpForwarder.c) on the PC to explicitly bridge the USB and WiFi networks, ensuring reliable bi-directional communication. Additionally, we faced hardware instability where the weapon arm moved uncommanded due to vibrations. This was traced to loose wiring on the motor driver inputs and was permanently fixed by securing the terminal connections and adding strain relief.

Software Architecture

The system uses a Producer-Consumer model with a thread-safe event queue. Input threads (Producer) capture raw controller data, while the main game loop (Consumer) processes logic and network commands at a fixed 50Hz rate. This decoupling ensures that slow network packets never block the controller input, maintaining responsiveness.

Description	Host/Target	Comp	Code	Author	Notes
Game Manager Engine	Target	5	C	Gabriel	Multi-threaded Central logic; handles hit detection, and rules.
Web Dashboard	Target	5	JS/HTML	Gabriel	Real-time Node.js interface using Socket.IO to display scores and start/stop game.
IPC Pipeline	Target	4	C/JS	Gabriel	Pipes C `stdout` to Node.js. Encountered intermittent state desynchronization; the C process occasionally failed to receive or process the `SIGINT` kill signal from Node.js, preventing a clean stop/restart.
Fail-Safe System	Target	5	C	Gabriel	Signal handling (SIGINT) to ensure motor drivers stop safely when game is killed
UDP Network	Host & Target	4	C	Zach	Broadcast handshake packets to auto-discover Raspberry Pi IP addresses. Would have trouble connecting with both players on occasion.
OLED Display (ssd1306)	Target	5	C	Gary	Displays “Ready” and live score updates on 12C screen via system calls. Worked Perfectly.
Controller Input	Target	5	C	Ahmet	Reads raw events from /dev/input/js* via Linux Joystick API for GameCube controllers.
Robot Firmware	Other	5	Python	Zach	Runs on RPI; translates UDP packets into GPIO signals for L298N motor drivers.

Extra Hardware

- **Raspberry Pi (x2):** Used for onboard controllers of the robots. They run a python listener script that receives UDP commands from the BeagleY-AI and toggles the GPIO pins for motor control.
- **Nintendo GameCube Controllers (x2) & USB Adapter:** Used as the input device for controls. The USB adapter allows the BeagleY-AI to read the controllers as standard Linux Input devices (js0, js1).
- **L298N Motor Drivers:** Interfaces the Raspberry Pi logic level with the Vex motors.
- **Vex 393 Motors:** Provide drivetrain and swinging arm motion for carts.
- **Robot Kit Frame:** Used as the structural backbone for the carts/robots. Provided mounting points for the motors, Raspberry Pi, and battery systems.
- **SSD1306 OLED Display (I2C):** A 128x64 pixel screen connected to the BeagleY-AI's bus. It provides immediate feedback to the player with the game state and score.

Extra Software & Libraries

- **wii-u-gc-adapter:** A userspace driver that communicates with the GameCube USB adapter via libusb and creates a virtual input driver (uinput). This let us read the proprietary Nintendo signals as standard Linux joystick events.
- **Node.js & Express:** Used to serve the web dashboard.
- **Socket.IO**: A JavaScript library enabling real-time, bi-directional communication between the web browser and the Node.js server. This was critical for updating the scoreboard instantly when a hit and occurred without requiring a page refresh
- **libusb & libudev:** Required dependencies to compile and run the GameCube controller driver on the BeagleY-AI.
- **ssd1306_bin:** A pre-compiled helper binary used to drive the OLED screen commands from our C application.

Quality and Functionality:

Our final system was functional and met the core requirements, with all major subsystems, controller input, networking, robot firmware, OLED output, and the game engine, working together as a coherent, real-time system. The game reliably detected hits, updated scores instantly, and controlled both robots with minimal latency, and the overall design made sense as a reasonably complete system with good integration, since all components communicated effectively to support uninterrupted gameplay. Although the system operated reliably overall, one minor issue remained, where the game would occasionally require the user to press Stop on the web interface and then Start again for normal operation to resume. This did not require restarting the program itself and did not affect gameplay stability, but it was a small incomplete aspect of the system that we were not able to fully eliminate.

Societal Impact:

Designed for entertainment, this project fosters social interaction through shared recreation, as observed during our demonstration. Beyond gaming, the system serves as a cost-effective proof-of-concept for remote teleoperation, demonstrating that accessible, off-the-shelf components can effectively achieve responsive, real-time control.