Automatic Dog Trainer

Vittorio Rossetto
0001136868
Alma Mater Studiorum - University of Bologna
Bologna, Italia
vittorio.rossetto@studio.unibo.it

Fig. 1. Picture of the entire system

***Abstract*—This paper presents the design and implementation of an automatic dog training system based on RaspberryPi 5 (Figure 1). The proposed system integrates computer vision, audio feedback, and automated treat dispensing to create a comprehensive training platform. The system monitors the dog's presence through a webcam and uses object detection algorithms to determine its position, then uses an image classification algorithm to recognize the dog's posture. Upon voice command transmission through a speaker, the system initiates a timer during which the dog must assume the requested position. Successful compliance results in automatic treat delivery via a servo-controlled dispenser. Additionally, the system features a manual mode allowing direct user intervention through webcam streaming and manual treat dispensing control. A web application provides training progress tracking and performance analytics. The proposed solution offers a cost-effective, automated alternative to traditional dog training methods while maintaining flexibility for user customization and remote monitoring.**

Keywords: automatic dog training, RaspberryPi, computer vision, animal behavior, smart pet devices, IoT systems

I. Introduction

Traditional dog training methods often require significant time investment, consistency, and professional expertise, making them challenging for many pet owners. Recent advancements in embedded systems and computer vision have created opportunities for developing intelligent automated training devices that can provide training sessions without constant human supervision.

This paper introduces a novel automatic dog training system that leverages RaspberryPi 5 technology to create an integrated training environment. The system addresses several key challenges in automated animal training: reliable dog detection and posture recognition, clear command delivery, timely reinforcement, and user customization. The approach combines multiple hardware components including a webcam for visual monitoring, a speaker for audio command transmission, and a servo-controlled treat dispenser for positive reinforcement.

The core innovation of the system lies in its closed-loop training methodology. This project implements a complete training cycle: command delivery, response monitoring, performance evaluation, and reward distribution. The computer vision component continuously analyzes the dog's position and posture, enabling real-time assessment of command compliance.

Furthermore, the system incorporates dual operational modes to accommodate different training scenarios. The automatic mode provides hands-free operation for routine training sessions, while the manual mode allows direct user intervention through remote webcam access and manual control over command delivery and treat dispensing. A complementary web application offers training progress tracking and performance analytics, enabling owners to monitor their dog's learning curve over time.

The main contributions of this work are:

- Design and implementation of a complete automated dog training system using RaspberryPi 5.
- Development of a real-time dog posture recognition system using computer vision.
- Implementation of a configurable training protocol with customizable commands.
- Integration of a web-based interface for remote monitoring and manual control.

The remainder of this paper is organized as follows: Section 2 details the system architecture. Section 3 the system implementation. Section 4 presents the experimental methodology and results. Section 5 discusses limitations and future work.

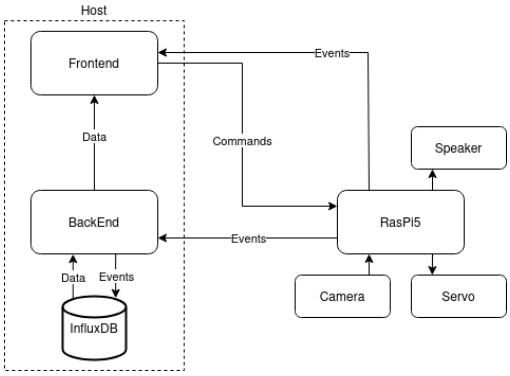


Fig. 2. System Architecture scheme

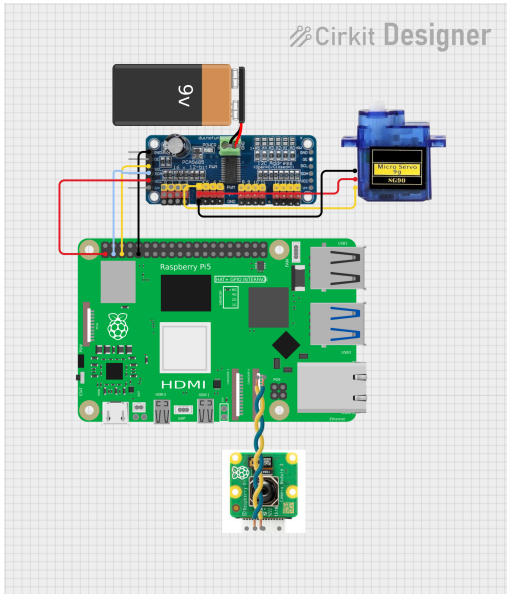


Fig. 3. System Circuit scheme

II. System Architecture

The proposed automatic dog training system employs a **modular distributed architecture** that integrates hardware components, AI-driven computer vision, and web-based monitoring through a centralized RaspberryPi 5 controller. Figure 2 illustrates the comprehensive system architecture and data flow.

A. Hardware Architecture

The hardware layer is built around the **RaspberryPi 5** as the central processing unit, coordinating all peripheral devices, as shown in Figure 3 through the following components:

- **Treat Dispensing System:**
  - *Actuator:* SG90 servo motor with 180° rotation capability
  - *Control:* PCA9685 PWM driver with 12-bit resolution (4096 steps) connected between RapberryPi and servo to avoid jitter
  - *Power:* Independent 9V battery supply
  - *Software Interface:* Adafruit Python library for precise motor control
- **Vision System:**
  - *Camera:* PiCamera V2 module with 5MP resolution: *Freenove 5MP*
  - *Connection:* Direct CSI interface to RaspberryPi
  - *Software Interface:* picamera2 Python library for video acquisition
- **Audio System:**
  - *Speaker:* Bluetooth-enabled portable speaker: *VTIN K1*
  - *Connection:* Bluetooth 5.0 pairing with RaspberryPi
- **Structure:**
  - *Material:* Custom DIY cardboard enclosure
  - *Design:*
    1) Shoe box as the main structure
    2) Bottle to contain treats
    3) Shelf to hold servo in place
    4) Slide to let treats fall in the bowl
  - *Portability:* Lightweight and easily reconfigurable design
  - *Echology:* The structure was entirely made with recycled material.

B. Software Architecture

The software stack is organized in multiple layers, each handling specific functionalities:

1) Computer Vision Layer:

- **Dog Detection:**
  - *Model:* YOLOv8n (nano version) for real-time performance
  - *Framework:* Ultralytics library for model inference
  - *Input:* 640×640 resolution frames at 15 FPS
  - *Output:* Bounding boxes with confidence scores $\geq$ 0.7
- **Posture Classification:**
  - *Model:* Custom CNN trained on Google Colab
  - *Classes:* {`sit`, `lie`, `stand`} positions
  - *Dataset:* Curated collection of dog posture images
  - *Accuracy:* 98.4% validation accuracy on test dataset
- **Libraries:** OpenCV (cv2) for image processing, NumPy for array operations

2) Control Logic Layer:

- **Training State Machine:**
  - *States:* Idle, Command, Monitoring, Reward, Delay
  - *Transitions:* Time-based and event-driven state changes
  - *Timing:* Configurable intervals for each training phase

3) Data Management Layer:

- **Time-Series Database:**
  - *Technology:* InfluxDB for high-frequency training data

  - *Metrics:* Command attempts, success rates, response times
  - *Tags:* Session IDs, command types, dog identifiers
- **Communication Protocol:**
  - *API:* RESTful endpoints for data transmission
  - *Format:* JSON payloads for structured data exchange
  - *Frequency:* Real-time streaming for camera feed, batch for metrics

C. Web Application Architecture

1) Frontend:

- **Technology Stack:**
  - *Framework:* React with functional components and hooks
  - *Build Tool:* Vite for fast development and optimized builds
- **Key Components:**
  - Manual control panel with command buttons
  - Logs window
  - Chart on daily results
  - Tables containing past data

2) Backend:

- **Technology Stack:**
  - *Language:* Python paired with Flask for API calls and websockets for WS communcation
  - *Database:* InfluxDB
- **Key Components:**
  - API to retrieve data from InfluxDB for frontend
  - Writer to influxDB from WS broadcasted data

D. System Integration

The components communicate through well-defined interfaces:

- **Local Communication:** Python multiprocessing for inter-module communication
- **Network Communication:** WebSocket for real-time data streaming, HTTP for API calls to access data from InfluxDB
- **Data Flow:** Camera → AI Models → Control Logic → Hardware Actuation → Data Storage → Web App

This architecture ensures **scalability** for additional training features, **maintainability** through clear separation of concerns, and **extensibility** for future hardware or software enhancements.

III. Project Implementation

A. Device Orchestration

The device's orchestration layer coordinates all core subsystems:

- computer vision
- servo actuation
- audio output
- host communication

This logic is implemented in the main application loop, which acts as the central controller for both autonomous and user-driven operation.

1) Initialization: At startup, the system initializes the `VisionSystem` for real-time dog detection and pose classification, the `ServoController` for treat dispensing, and a WebSocket server that enables bidirectional communication with the external interfaces. Incoming messages from the host are routed through a command handler for:

- mode switching (automatic/manual)
- manual treat dispensing
- audio playback (text-to-speech, file-based, or base64-encoded)
- override controls for safety mechanisms such as temporarily disabling automated rewarding.

2) Automatic Mode: In automatic mode, the orchestration layer follows a finite-state logic that progresses through *stand detection*, *command issuance*, *reward evaluation*, and a *cooldown phase*. Pose estimates streamed from the vision subsystem drive these transitions, allowing the device to deliver context-appropriate feedback. When a correct pose is recognized within the configured time window, the orchestrator triggers a synchronized sequence: a servo sweep to release a treat, audio-based praise, and event logging back to the host interface.

The main loop continuously processes camera frames, updates system state, handles host commands, and broadcasts status messages such as pose transitions, treat events, and operating mode. This event-driven architecture ensures that the device remains responsive to both autonomous logic and immediate user commands, while maintaining consistent synchronization among hardware components and external UI.

B. Computer Vision

The computer-vision component of the system is responsible for real-time dog detection and pose classification using a Raspberry Pi 5 and the Picamera2 imaging pipeline. The module builds on the Ultralytics YOLO family of neural networks, which provide fast object detection and image classification suitable for embedded environments.

The vision system initializes two deep-learning models:

- **YOLO object detector**: used to localize dogs within the camera stream
- **Custom pose-classification model**: trained to differentiate between key dog postures (e.g., standing, sitting)

The camera is configured through the Picamera2 API to capture RGB frames at a fixed resolution, which are preprocessed and passed into the models.

1) Model training: To enable reliable dog-pose recognition, a lightweight image-classification model was trained using a custom dataset. The dataset was prepared and annotated through the Roboflow platform, which handled class labeling, preprocessing, and automatic train/validation/test splits. The final dataset was downloaded directly into the training notebook using the Roboflow Python API.

Model training was performed using the Ultralytics YOLO classifier in order to maintain compatibility with the real-time YOLO detection pipeline deployed on the Raspberry

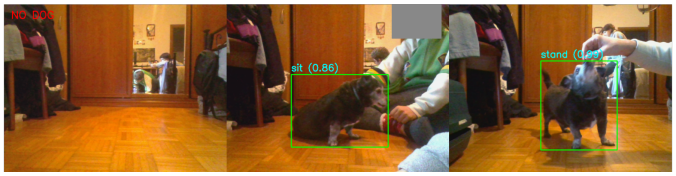


Fig. 4. Examples of detection from the implied computer vision system

Pi. A compact pretrained backbone (YOLOv8-tiny-cls) was fine-tuned on the annotated dog-pose classes (e.g., standing, sitting, lying). Training relied on standard augmentation, cross-entropy loss, and early stopping to prevent overfitting. After convergence, the best checkpoint was exported.

This process produced a small, high-accuracy classifier capable of operating in real time alongside the object detector, enabling the system to distinguish dog postures reliably under varied lighting and viewing conditions.

2) System functioning: Detection is performed by down-sampling each frame to reduce computational cost, after which YOLO outputs bounding boxes and class confidences. When a dog is detected, the bounding box is rescaled to the original frame dimensions and used to extract a region-of-interest (ROI). This ROI is then fed into the classifier to predict the dog's pose. The system returns both bounding-box coordinates and pose labels Figure 4.

C. Servo movement

1) Hardware: The servo-control subsystem is implemented using the Adafruit PCA9685 PWM driver to ensure precise, hardware-timed actuation independent of Raspberry Pi CPU load. This removes jitter and instability that would result from software-based PWM.

2) Software: Servo motion is executed using the *set_angle()* method, which clamps commands to safe limits and avoids redundant writes to minimize power fluctuations. The reward-delivery gesture is encapsulated in the *sweep()* function, which moves the actuator through a predefined sequence, from 90° to 0° and right back. To prevent motor wear, the *stop()* method disables the PWM signal entirely, allowing the servo to relax when not in active use.

D. Audio Control

The audio subsystem is responsible for text-to-speech (TTS), playback of prerecorded audio, and handling of externally supplied audio data. The implementation relies on existing Linux command-line tools, allowing the system to operate without embedding heavy TTS libraries.

1) Text-to-Speech: TTS generation is performed using either `espeak` or `pico2wave`, depending on availability. When speech is requested, the system dynamically produces a temporary WAV file containing the synthesized audio. A small delay ensures the file is fully written before playback. The audio is then routed to the desired Bluetooth speaker by identifying an appropriate PulseAudio sink. Playback is handled through `paplay`, `ffplay`, or `aplay`, depending on which player is installed.

2) Playback of Stored Audio: For prerecorded prompts, the system searches a project `recordings/` directory for matching filenames. Supported formats include WAV, MP3, OGG, M4A, and FLAC. If a file is found, it is played directly using the same backend described above. This mechanism allows quick reuse of common feedback sounds without requiring TTS generation.

3) Raw and Encoded Audio: The system can also play audio delivered as raw bytes or as Base64-encoded strings, which is useful for remote communication over network protocols such as HTTP or WebSocket. Incoming data is written to a temporary file and then passed to the playback routine. Temporary files are automatically removed after use to avoid storage buildup.

E. Communications Architecture

The Dog Trainer device incorporates a communication framework bidirectional interaction between the RaspberryPi, the host user interface, and the database. Communication is implemented through a combination of WebSockets, HTTP endpoints, and time-series database integration. This design supports real-time control, low-latency data streaming, and long-term analytics.

1) Local Host Communication: Local communication between the device and the host machine is managed by the `host_comms` module, which exposes two primary mechanisms:

a) HTTP Interface.: A lightweight Flask server provides REST endpoints for receiving status updates and issuing mode changes. The `/status` endpoint allows the device to push posture state, confidence values, and internal stage information, while `/set_mode` enables the host to switch between automatic and manual modes. HTTP is also used as a fallback for event delivery when WebSocket connectivity is unavailable.

b) WebSocket Interface.: A dedicated asynchronous WebSocket server runs in a separate thread to support low-latency, bidirectional messaging. All messages are dispatched to the command handler on the device. Outgoing messages are broadcasted for real-time UI updates.

2) Event Streaming and Data Logging: To support long-term monitoring and visualization, certain events are forwarded to an InfluxDB time-series database. The `influx_api` module exposes HTTP endpoints used by the host interface to query historical measurements such as dog posture activity, successful command responses, treat dispensing frequency, and audio playback logs. Measurements are timestamped, normalized into a concise schema, and returned in a UI-friendly format.

3) Message Abstraction: All outgoing device messages follow a unified envelope format:

$$\{\text{type},\ \text{event},\ \text{timestamp},\ \text{payload}\}$$

This abstraction allows the same event to be transmitted either over WebSocket (in real time) or via HTTP POST (as a fallback). It also ensures compatibility with logging sinks such

as InfluxDB. Similarly, all inbound messages from the host UI follow the structure:

$$\{\text{cmd},\ \text{parameters}\}$$

enabling a simple and extensible command-routing mechanism within the device.

F. Database

The system employs a lightweight but robust pipeline based on InfluxDB to store events, device actions, and system status updates. This database layer enables long-term analysis of dog–trainer interactions, command effectiveness, and device performance. Two components implement this functionality: a WebSocket event collector running on the host machine, and an InfluxDB writer.

1) Event Collection and Preprocessing: A dedicated collector process subscribes to the device's WebSocket event stream and receives JSON-encoded envelopes containing event type, timestamp, and payload. The collector normalizes these envelopes and performs lightweight preprocessing, including:

- deduplication and internal buffering for `audio_playback` events,
- correlation of commands with subsequent `pose_transition` events (within a 5 s window) to infer command success,
- daily aggregation of treat counts and command success rates.

2) InfluxDB Writer: All events are persisted through a unified writer module that supports both InfluxDB v2 (token-based) and v1 (HTTP or legacy client). The writer maps each event category to a dedicated measurement schema:

- *dog_activity*: registering dog appearances in camera view
- `pose_transition`: tagged by previous and next pose, usefull to understand command following
- `audio_playback`: logging issued commands or played files
- `treat_given`, `mode_change`, `status`, or generic events.

Payloads are flattened into Influx fields, encoded as JSON strings.

3) Reliability: If the v2 client libraries are unavailable, the writer automatically falls back to v1 or raw HTTP writes, increasing portability across environments.

G. Web Application

The system includes a lightweight, browser-based control interface that enables monitoring and manual interaction with the device. The web application is implemented in React and communicates with the Raspberry Pi via a persistent WebSocket channel.

1) Manual Training Interface: The primary interface, `ManualTrainerApp` Figure 5, provides direct control over pose commands, treat dispensing, mode switching, and audio playback. To avoid browser mixed-content restrictions, the application automatically selects `ws://` or `wss://` depending on whether the page is served over HTTP or HTTPS. When the user clicks *Connect*, the interface establishes a WebSocket connection to the device at port 8765, logs all incoming messages, and exposes UI controls only once the channel is open.

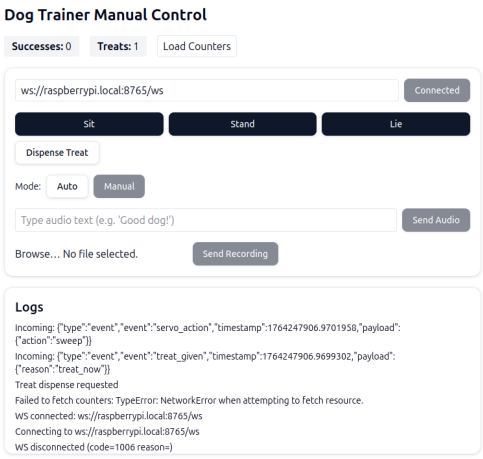


Fig. 5. Manual trainer interface

`command_success` and `treat_given` increment on-screen counters and temporarily highlight success indicators. The interface also supports three forms of audio transmission:

1) raw text for on-device TTS
2) prerecorded audio files
3) base64-encoded audio payloads

File uploads use a `FileReader` to convert the selected recording into a base64 string before transmitting it over WebSocket.

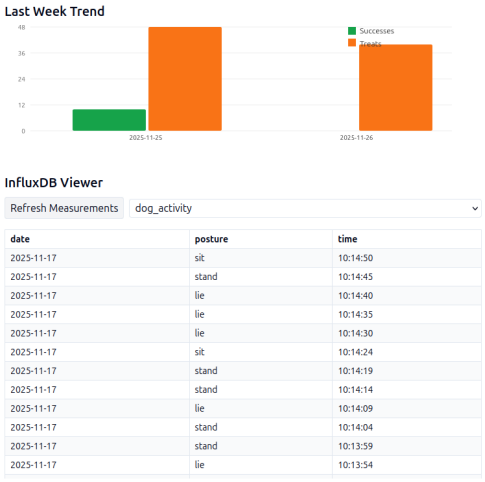
The UI exposes direct pose commands (`sit`, `stand`, `lie`), as well as a *Treat* button for immediate reward dispensing. The user can switch between automatic and manual operating modes, with all actions being logged locally for transparency and debugging.

2) InfluxDB Data Viewer: In addition to manual control, the web interface includes an InfluxDB visualization module Figure 6. This viewer communicates with a local HTTP API that exposes database measurements collected from the training sessions.

Upon loading, the viewer requests the latest seven days of aggregated `daily_counters` points, which include the number of successful command completions and the number of treats dispensed. These are rendered in a custom SVG bar chart, offering an overview of weekly progress.

Users can select any measurement and display the mostrecent points in a scrollable table.

3) System Integration: Combined, the manual control interface and the database viewer provide a complete toolsuite for interacting with the training device. This modular architecture allows the frontend to serve both real-time training actions and long-term analytics.



date	posture	time
2025-11-17	sit	10:14:50
2025-11-17	stand	10:14:45
2025-11-17	lie	10:14:40
2025-11-17	lie	10:14:35
2025-11-17	lie	10:14:30
2025-11-17	sit	10:14:24
2025-11-17	stand	10:14:19
2025-11-17	stand	10:14:14
2025-11-17	lie	10:14:09
2025-11-17	stand	10:14:04
2025-11-17	stand	10:13:59
2025-11-17	lie	10:13:54

Fig. 6. Data viewer Interface

IV. Conclusions

A. Project Accomplishments

This project successfully demonstrates the integration of multiple technological domains - computer vision, embedded hardware control, web development, and real-time communications - into a single, coherent system for assisting in dog training. The implemented device is capable of detecting a dog's presence, classifying its posture, dispensing treats, issuing verbal commands, and presenting a visual interface. Although conceptually ambitious, the system operates reliably in practice, showing that such a multifaceted automation pipeline can be built using accessible tools and modern machine learning models.

B. Limitations

Despite its success, the system presents several limitations that must be acknowledged. First, the project remains fundamentally a *proof of concept*. It has been validated in a controlled environment, but its long-term performance, robustness, and behavioural impact on dogs require extended real-world testing. Only through continuous use and systematic observation would it be possible to assess the true training benefits and the reliability of the system over time.

Secondly, the applicability of the system is currently constrained to *pre-trained* dogs. The device can reinforce commands such as "sit" or "lie", but it cannot teach these behaviours from scratch. As a result, the system focuses on *training maintenance* rather than *skill acquisition*, limiting its usefulness for inexperienced dogs or for owners seeking to teach entirely new behaviours.

C. Future Work

Several directions exist for enhancing and extending the system in future iterations:

- **Mobile application:** Developing an accompanying mobile app would make the system more accessible, enabling remote control, notifications, and progress tracking. A social platform could also be introduced to allow users to share training achievements and compare progress.
- **Improved physical design:** Exploring new materials and structural configurations could yield a more aesthetically pleasing and more robust device. Improving mechanical resistance would increase safety, particularly when used with energetic or easily agitated dogs.
- **Personalised feeding planning:** Integrating adaptive feeding schemes: such as switching between treat types, or adjusting portion sizes based on user preferences or dietary requirements, would greatly extend the system's utility beyond behavioural training.

In summary, while the project demonstrates a successful initial prototype, further refinement, testing, and extension would be required to fully realise its potential as a general-purpose automated dog-training assistant.