

Household pet monitoring robot Based on Arduino UNO Q

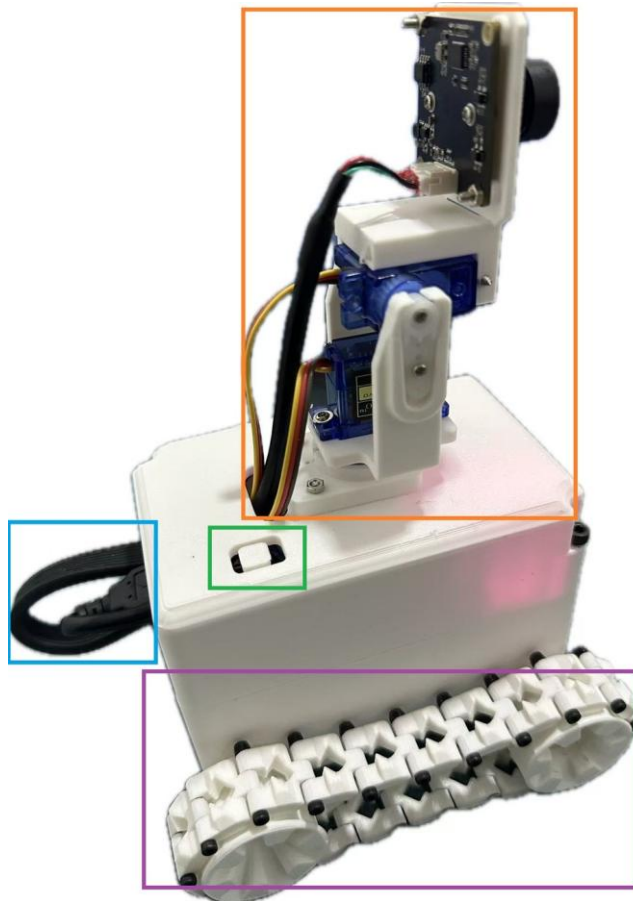
Tags

- FPV Live Video
- AI Object Detection
- Tracked Robot
- Web Control
- Gimbal Control

Difficulty Level

Intermediate — This project spans multiple technical layers: tracked chassis assembly, Arduino firmware programming (including software PWM techniques), Python backend communication with AI vision, and responsive web frontend development. Beginners can complete the basic remote control with guidance, but understanding advanced features like hybrid drive mode and AI vision integration requires some embedded systems and full-stack web development experience.

Project Cover Image



Project Cover Image

Figure 1: FPV AI Desktop Surveillance Robot Project Cover

Hardware Components

Building your own Household pet monitoring robot AI Desktop Surveillance Robot requires the following key components. Each part plays a crucial role in enabling this tracked vehicle to move freely, stream real-time video, and detect objects.

Drive Motors (2 Geared Motors)

The core of the robot's movement lies in two geared DC motors, driving the left and right tracks respectively. The tracked chassis uses differential steering — turning is achieved by varying the speed difference between the left and right motors:

- **Left Motor:** Connected to digital pins D5 and D6, controlled via the A4950 motor driver module. Forward/reverse rotation is determined by the logic level combination of these two pins
- **Right Motor:** Connected to digital pins D9 and D10, also driven by the A4950. Due to the mirror-symmetric mounting on the tracked chassis, the right motor has reverse logic processing applied at the code level

Motor speed is controlled via PWM signals, ranging from 0–255, corresponding to stationary to full speed. The system supports an adjustable speed range of 50–255, allowing users to fine-tune for indoor or outdoor environments.

Gimbal Servos (2 SG90 Servos)

The two-axis gimbal gives the robot a “flexible field of view,” enabling the camera to observe independently of the chassis movement:

- **Pan Servo (Horizontal Rotation):** Connected to digital pin D11, controls the camera's left-right rotation with a range of 0°–180°, neutral position at 90°
- **Tilt Servo (Vertical Pitch):** Connected to digital pin D3, controls the camera's up-down angle with a range of 60°–120° (center at 90°, $\pm 30^\circ$ limit), preventing mechanical collision of the gimbal

SG90 servos are known for their compact size and fast response, making them ideal for gimbal applications carrying lightweight USB cameras.

Arduino UNO Q Development Board

The brain of the entire system is the Arduino UNO Q — a modern Arduino platform based on the STM32/Zephyr architecture. Unlike the traditional AVR-based UNO, the UNO Q features:

- **Built-in WiFi:** Enables wireless communication and web control
- **RouterBridge Communication Protocol:** Seamless integration between Python middleware and Arduino firmware
- **Greater Processing Power:** 48MHz clock speed, sufficient for real-time motor control and servo driving

An important note: the UNO Q's pin mapping differs from the traditional Uno. Code must use prefixed constant definitions like `D5`, `D6`, rather than plain numbers `5`, `6`.

A4950 Dual Motor Driver Module

The A4950 module converts Arduino's low-power control signals into the high-current output needed to drive the motors:

- Supports 7.4V–12V lithium battery input
- Dual independent outputs, controlling left and right motors separately
- Supports forward/reverse rotation and PWM speed control
- Maximum continuous current of 3.5A per channel, sufficient for small geared motors

USB Camera

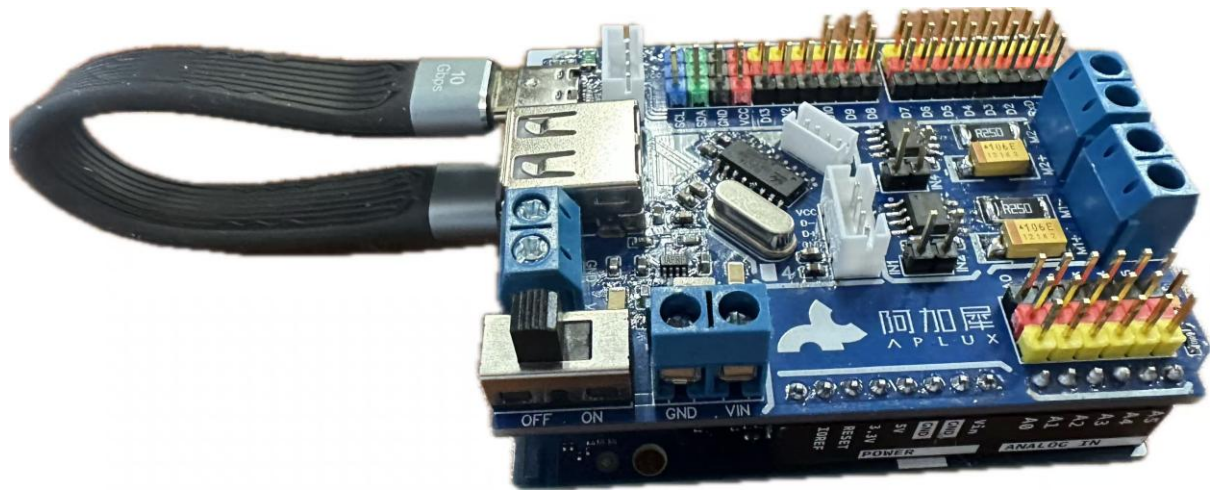
The robot's "eyes" — a standard USB camera connected to the Arduino UNO Q via USB port:

- Provides a live video stream via port 4912
- Supports AI object detection
- Video stream is embedded in the web control interface with low latency

Custom Expansion Board

The expansion board serves as the hub connecting all components, performing critical integration functions:

- **Power Distribution:** Safely distributes battery voltage to the motor driver, servos, and main controller
- **Signal Routing:** Routes Arduino digital pins to the motor driver and servos respectively
- **Interface Integration:** Connects to the Arduino UNO Q via a C-to-C data cable



Arduino UNO Q and Custom Expansion Board

Figure 2: Arduino UNO Q and custom expansion board connected via C-to-C data cable

Power System

The robot is powered by a 7.4V–12V lithium battery, providing ample driving current for the motors while delivering stable 5V power to the Arduino and servos through a voltage regulator module. The independent power system allows the robot to operate fully autonomously without a computer.



Motor and Battery Layout

Figure 3: Internal chassis layout — 2800mAh 12V lithium battery pack and geared motors on both sides

Tracked Chassis

The mechanical chassis uses a tracked structure, offering superior ground traction and obstacle-crossing capability compared to wheeled designs:

- **Track Modules:** One set on each side, directly driven by their corresponding motors
- **Chassis Frame:** Carries electronic components, battery, and gimbal module

- **Gimbal Mount:** Located on top of the chassis, supporting the camera and two-axis servos

Complete Hardware List

Component	Quantity	Model	Purpose
Geared Motors	2	DC Geared Motor	Left and right track drive
Gimbal Servos	2	SG90	Camera horizontal rotation (Pan) and vertical pitch (Tilt)
Motor Driver	1	A4950 Dual	Motor direction and speed control
Development Board	1	Arduino UNO Q	STM32/Zephyr-based main controller
Camera	1	USB Camera	Live video streaming and AI detection
Expansion Board	1	Custom	Power distribution and IO expansion

Battery	1	7.4V–12V Li-ion	Autonomous power supply
C-to-C Data Cable	1	USB-C	Connects expansion board to Arduino UNO Q
Tracked Chassis	1 set	Custom	Robot mechanical structure
Screws	Various	M2, M3	Hardware assembly

Tools and Machines

- **Personal Computer:** For programming and initial configuration
- **Soldering Kit:** For any necessary wire connections
- **Screwdriver Set:** For chassis and gimbal assembly (primarily Phillips head)
- **3D Printer** (optional): For printing custom gimbal brackets

Development Platform

- **Arduino App Lab:** Web IDE for programming the Arduino UNO Q

- **Python 3.13+:** Backend communication processing and AI vision
 - **Modern Web Browser:** Chrome, Firefox, or Edge for the control interface
-

Project Description

Introduction

FPV (First Person View) technology first gained prominence in the drone industry, allowing pilots to see a real-time aerial view from the drone's perspective. This project brings that concept to the desktop, creating an AI surveillance robot that can freely roam across desktops and floors.

Unlike traditional RC cars, this robot not only moves in four directions but also features a pair of “intelligent eyes” — a real-time video streaming system with AI object detection. The entire system uses the Arduino UNO Q development board as its core controller, combined with a Python backend (handling AI vision and command routing) and a JavaScript frontend (providing a cyberpunk-styled control interface), building a complete FPV robot control system.

What makes this project stand out?

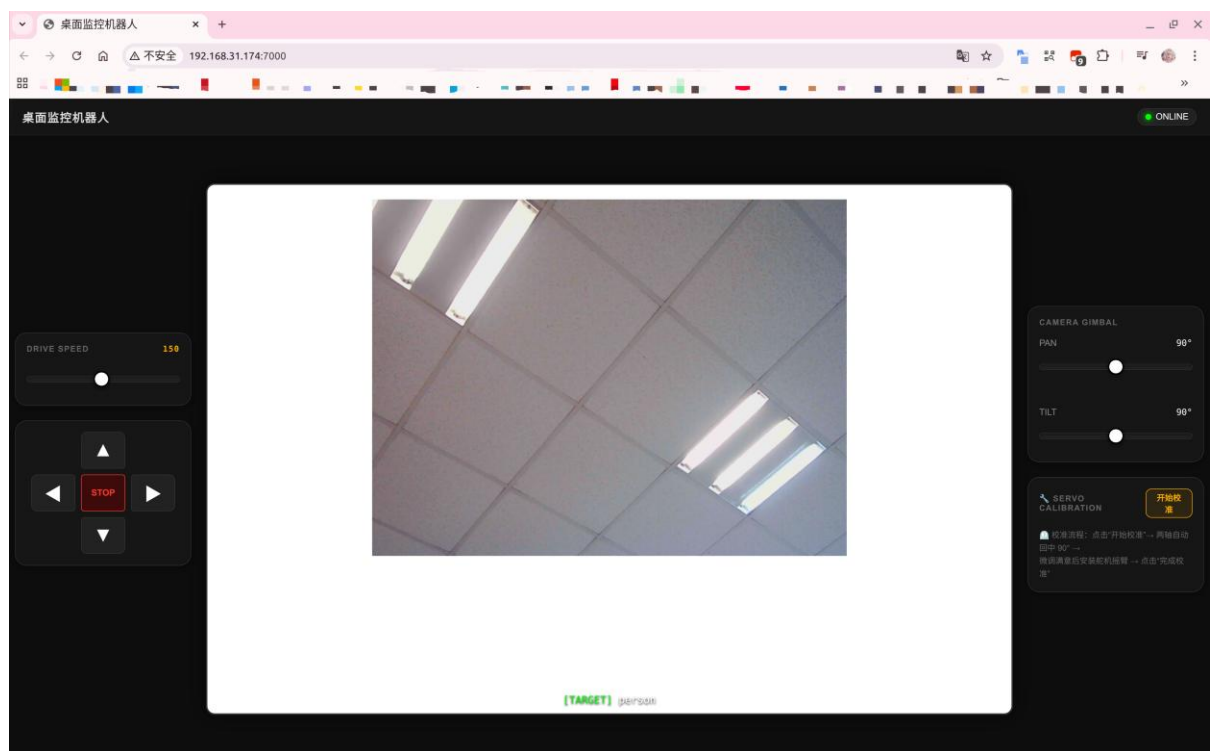
First, **zero installation, pure browser control**. No need to download any app — open a browser, enter the robot's IP address, and you're immediately in the control console. Laptops, tablets, phones — any device with a browser can become your remote controller.

Second, **built-in AI object detection**. The robot is equipped with a real-time object detection engine capable of identifying people, objects, and other targets in its field of view, displaying them in the HUD (Heads-Up Display) as `[TARGET] person` in real time. This isn't just an RC car — it's a mobile AI sentinel.

Third, **independent two-axis gimbal**. The camera can rotate freely within a horizontal range of 0°–180° and vertical range of 60°–120°,

independent of chassis movement. This means you can drive the car forward while using the gimbal to look around — a true FPV experience. Additionally, the web interface includes a **servo center calibration function**, allowing one-click centering of both axes to 90° during installation, with $\pm 1/\pm 5$ degree fine-tuning buttons for precise calibration before securing the horn, ensuring accurate initial gimbal posture.

Fourth, **adaptive responsive interface**. The console features a cyberpunk-style design, displaying a three-column layout on PC (driving–video–gimbal), automatically rearranging to a single-column layout (video → gimbal → driving) in phone portrait mode, perfectly adapting to one-handed operation.



Web Console PC Interface

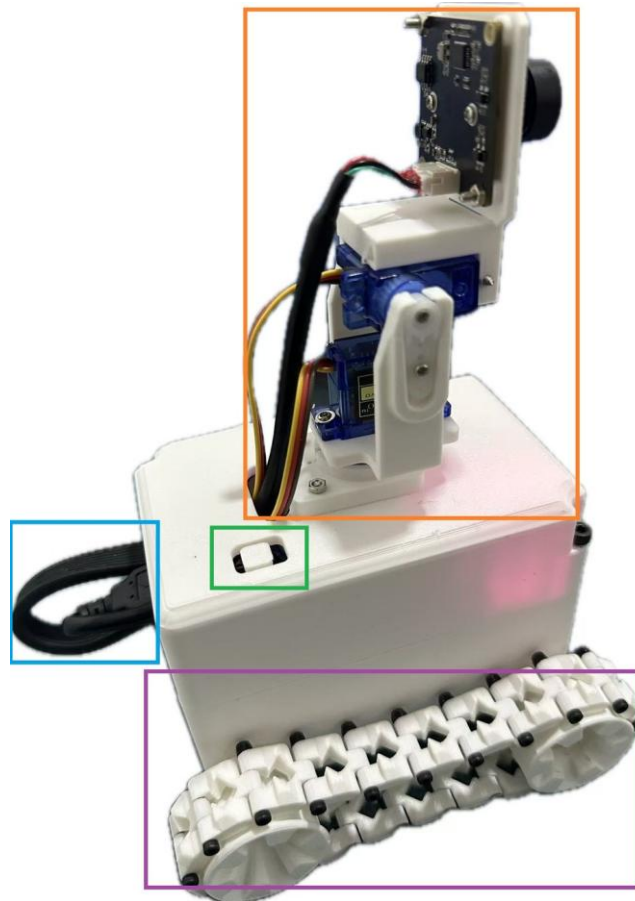
Figure 4: PC three-column layout — driving controls on the left, FPV video stream (with AI detection labels) in the center, gimbal and calibration panel on the right



Web Console Mobile Interface

Figure 5: Phone portrait single-column layout — video stream, gimbal sliders, servo calibration, speed and direction controls arranged top to bottom, optimized for one-handed operation

Fifth, **hybrid drive technology**. At the firmware level, the project creatively solves the timer conflict between the Servo library and analogWrite — using hardware timers for servo driving to ensure zero jitter, while using non-blocking software PWM for motor driving to achieve smooth speed control. This approach elegantly enables the harmonious coexistence of servos and motors within limited hardware resources.



Robot Body

Figure 6: Household pet monitoring robot body, showing the gimbal, tracks, and electronic modules

Circuit Diagram and Pin Configuration

Understanding pin configuration is essential for correct assembly and troubleshooting. The FPV robot uses a dual-motor differential drive plus two-axis gimbal structure, utilizing 6 digital pins in total.

Pin Assignment:

Function	Pin	Description
----------	-----	-------------

Left Motor IN1	D5	Left track forward/reverse control pin 1 Left motor positive	M1+:
Left Motor IN2	D6	Left track forward/reverse control pin 2 Left motor negative	M1-:
Right Motor IN3	D9	Right track forward/reverse control pin 1 Right motor positive	M2+:
Right Motor IN4	D10	Right track forward/reverse control pin 2 Right motor negative	M2-:
Gimbal Pan	D3	Camera horizontal rotation (0°–180°)	
Gimbal Tilt	D11	Camera vertical pitch (60°–120°, center at 90°)	

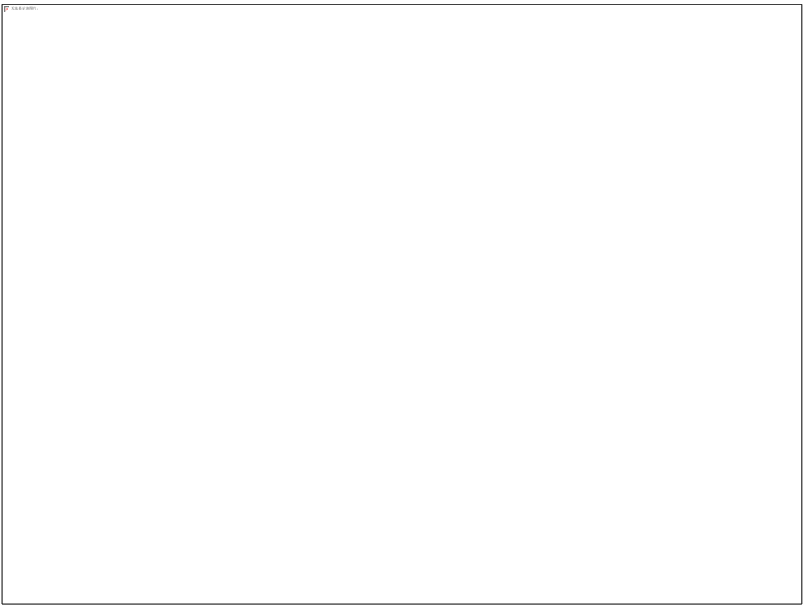
Motor Direction Control Logic:

For each motor, the logic level combination of its two pins determines the motor's behavior:

IN1	IN2	Motor State
LOW	LOW	Stop (coast)
HIGH	LOW	Forward
LOW	HIGH	Reverse

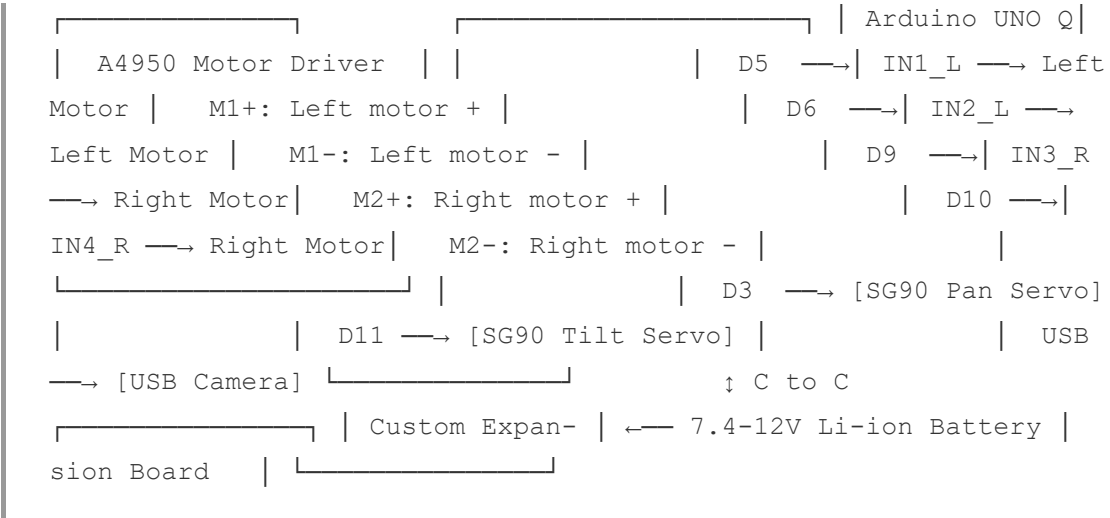
Since the left and right motors on the tracked chassis are **mirror-symmetrically mounted** (back-to-back), applying the same signal to

both causes them to rotate in opposite directions. Therefore, the firmware has already applied reverse processing to the right motor — sending the same “forward” command causes both motors to rotate in the same direction, driving the car straight ahead.



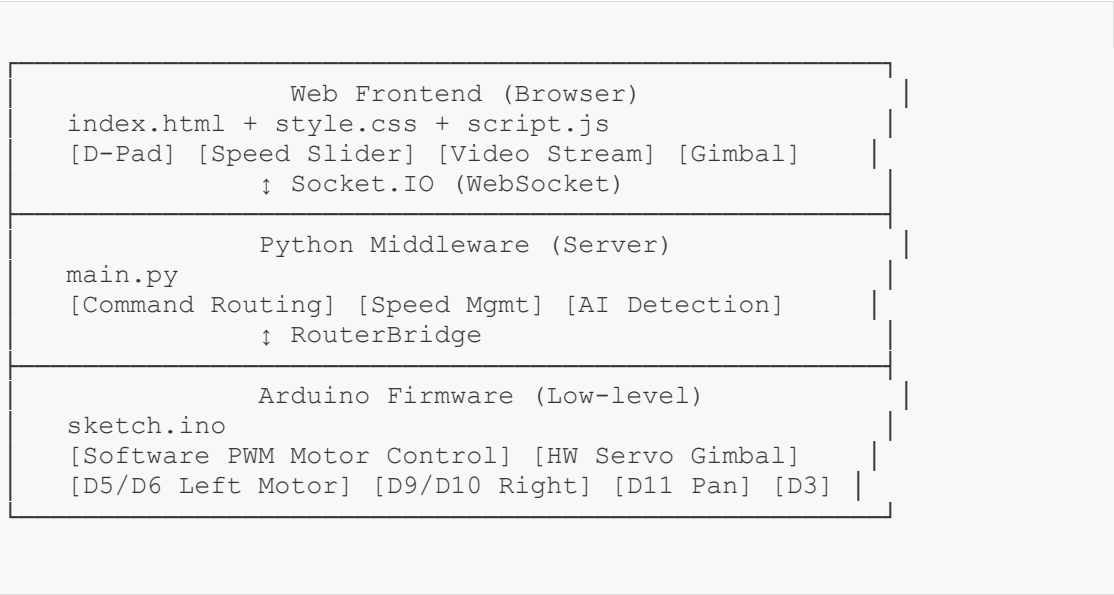
Wiring Diagram

*Figure 7: Inter-module wiring logic diagram



Hardware Architecture

The system adopts a three-layer separation architecture, where each layer handles its own responsibilities and communicates through standardized interfaces:



Layer 1: Web Frontend (Browser)

The user interface runs entirely in the browser, built with HTML5, CSS3, and JavaScript. It provides three main functional areas:

- **Driving Control Panel (Left):** Contains the Drive Speed slider (50–255 range) and directional D-Pad (up/down/left/right/emergency stop — five buttons)
- **Video Monitoring Panel (Center):** Embeds the live video stream iframe and HUD overlay, displaying AI detection results
- **Gimbal Control Panel (Right):** Provides Pan (horizontal) and Tilt (vertical) sliders to control the camera’ s rotation angles

Communication between the browser and backend uses WebSocket ([Socket.IO](#)) for real-time bidirectional messaging. The frontend prioritizes loading the local `socket.io.min.js` library, ensuring functionality even without internet access.

Layer 2: Python Middleware

The Python server acts as a “translator,” handling command routing and intelligent processing:

- **main.py:** Web server initialization, [Socket.IO](#) event listening, and AI vision module management

The Python middleware has three core responsibilities:

1. **Speed Management:** Maintains a global `current_speed` variable, receives frontend slider values and constrains them to the 0–255 range
2. **Command Routing:** Converts frontend semantic commands (`up`, `down`, `left`, `right`, `stop`) into specific left/right motor speed parameters, sent to Arduino via Bridge
3. **AI Vision:** Runs the object detection engine in the background, pushing detection results to the frontend HUD via Socket in real time

Layer 3: Arduino Firmware

The microcontroller runs optimized C++ code for real-time hardware control:

- Non-blocking software PWM motor driving
- Hardware Servo library gimbal control
- RouterBridge communication protocol handling
- Servo and motor initialization with safety limits

The key innovation here is the **hybrid drive mode** — using hardware timers for servos (zero jitter) and software PWM for motors (avoiding

timer conflicts), achieving harmonious coexistence of both actuator types within limited hardware resources.

Feature Description

1. FPV Live Video Streaming

The core selling point — first-person view experience. The USB camera's live feed is streamed back via port 4912, embedded in the central video panel of the web interface. The entire video transmission chain has extremely low latency, creating an “immersive” control experience.

javascript

```
// Video stream embedding — auto-detect robot IP, construct stream URL
const streamUrl = `http://${window.location.hostname}:4912/embed`;
document.getElementById('cam-frame').src = streamUrl;
```

2. AI Object Detection

The robot is equipped with a real-time object detection engine based on `VideoObjectDetection`, with a confidence threshold set to 0.5. Detected object categories are pushed to the frontend via WebSocket in real time, displayed in the HUD overlay at the bottom of the video feed.

python

```
from arduino.app_bricks.video_objectdetection import
VideoObjectDetection

detection = VideoObjectDetection(confidence=0.5)

# When objects are detected, push results to the frontend
def on_detect(detections):
    content = list(detections.keys())
    if content:
        ui.send_message("ai_result", {"objects": content})
detection.on_detect_all(on_detect)
```

After receiving AI results, the frontend displays them in the HUD with a green `[TARGET]` label followed by the object name:

javascript

```

socket.on('ai_result', (data) => {
    if(data.objects && data.objects.length) {
        aiText.innerHTML = `[TARGET]</span>`
    }
});

```

3. Omnidirectional Movement and Speed Control

The tracked differential drive enables forward, backward, left turn, and right turn movement. Speed is adjusted in real time via the frontend slider (50–255), and direction buttons use a “press to go, release to stop” safety mechanism.

python

```

def move_car(cmd):
    global current_speed
    SPEED = current_speed

    if cmd == 'up':      sl, sr = SPEED, SPEED      # Forward: both
wheels same speed
    elif cmd == 'down': sl, sr = -SPEED, -SPEED    # Reverse: both
wheels reverse
    elif cmd == 'left': sl, sr = -SPEED, SPEED     # Left turn:
left reverse, right forward
    elif cmd == 'right':sl, sr = SPEED, -SPEED     # Right turn:
left forward, right reverse
    elif cmd == 'stop': sl, sr = 0, 0              # Stop

    Bridge.call("motor_l", sl)
    Bridge.call("motor_r", sr)

```

The principle of differential steering is intuitive:

- **Straight line:** Both motors at the same speed, same direction
- **Pivot turn:** Both motors at the same speed, opposite directions
- **Stop:** Both motors at zero speed

4. Two-Axis Gimbal Control

The Pan servo (horizontal) and Tilt servo (vertical) provide the camera with viewing angle adjustment independent of the chassis. Pan supports the full 0°–180° range, Tilt is limited to $\pm 30^\circ$ (60°–120°), and

both axes center at 90°. Frontend slider values are sent via Socket to the Python middleware, then forwarded to Arduino via Bridge for execution.

To prevent frequent control signals from causing communication congestion, the frontend implements a 30-millisecond throttle mechanism:

javascript

```
let lastSend = 0;
function updateGimbal(axis, val, displayId) {
  document.getElementById(displayId).innerText = val + "°";
  const now = Date.now();
  if (now - lastSend > 30) { // Throttle: at least 30ms interval
    socket.emit(axis, val);
    lastSend = now;
  }
}
```

5. Responsive Cyberpunk Interface

The console features a dark cyberpunk-style design, with visual elements primarily in Cyan and Amber color tones. The layout uses CSS Flexbox for responsive design:

- **PC/Tablet Landscape:** Left (driving controls)–Center (video stream)–Right (gimbal controls) three-column layout
- **Phone Portrait:** Automatically rearranges to video (top) → gimbal (middle) → driving (bottom) single-column layout, matching one-handed operation habits

CSS

```
/* Phone portrait layout restructuring */
@media (max-width: 900px) and (orientation: portrait) {
  .dashboard { flex-direction: column; }
  #panel-video { order: 1; } /* Video stream on top */
  #panel-gimbal { order: 2; } /* Gimbal controls in middle */
  #panel-drive { order: 3; } /* Movement controls at bottom */
}
```

6. Safety Protection Mechanisms

Multiple safety measures prevent accidental loss of control:

- `touchcancel` and `mouseleave` event listeners automatically stop the car when a finger slides off the button area
- Central red STOP emergency button
- Motor speed constrained to 0–255 range (`constrain` function)
- Servo angle limits (Pan: 0–180°, Tilt: 60–120°) prevent mechanical collision
- Right-click context menu disabled to avoid accidental triggers during operation

7. Weak Network Auto-Reconnection

In real-world use, unstable WiFi signals are common — brief phone disconnections, the robot driving out of WiFi coverage, etc. The system includes a complete weak network reconnection mechanism, ensuring that when the network recovers, the control channel is automatically re-established without requiring a manual page refresh.

The reconnection mechanism operates on two levels:

- **Hardware Level (Arduino UNO Q):** The onboard WiFi module has automatic reconnection capability. After losing WiFi, once the known hotspot is rediscovered, it reconnects automatically without code intervention
- **Application Level (Socket.IO Frontend):** The browser-side [Socket.IO](#) client is configured with an automatic reconnection strategy, with real-time UI feedback on connection status

Frontend reconnection configuration:

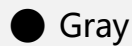
javascript

```
const socket = io({
  reconnection: true,           // Enable auto-reconnection
  reconnectionAttempts: 30,     // Maximum 30 retry attempts
  reconnectionDelay: 1000,     // Initial retry delay: 1 second
  reconnectionDelayMax: 5000,  // Maximum retry delay: 5 seconds
  (exponential backoff cap)
  timeout: 10000,              // Single connection timeout: 10
                                seconds
});
```

Reconnection status is displayed in real time via the top-right status indicator and the video area HUD:

Status	Indicator	HUD Text	Description
Connected	 Green (solid)	SYSTEM READY	Normal operating state
Disconnected	 Gray	DISCONNECTED	Network interrupted, awaiting reconnection
Reconnecting	 Yellow (blinking)	 RECONNECTING... Attempt #N	Auto-reconnecting
Reconnected	 Green (solid)	 RECONNECTED	Network restored, control resumed

Reconnect



✗ RECONNECT

Exceeded 30

Failed

FAILED Please refresh attempts

javascript

```
// Weak network reconnection status listeners
socket.io.on('reconnect_attempt', (attempt) => {
  // Status indicator turns yellow blinking, shows reconnection
  progress
  dot.classList.add('reconnecting');
  stat.innerText = `Reconnecting #${attempt}`;
});
socket.io.on('reconnect', (attempt) => {
  // Reconnection successful, restore green
  dot.classList.add('online');
  stat.innerText = "ONLINE";
});
socket.io.on('reconnect_failed', () => {
  // Exceeded maximum retry attempts
  stat.innerText = "Reconnect Failed";
});
```

The reconnection uses an **exponential backoff** strategy: 1st attempt waits 1 second, 2nd waits 2 seconds, 3rd waits 4 seconds... until reaching the 5-second cap, after which it retries at a constant rate. This avoids a storm of requests when the server hasn't yet recovered, while ensuring the connection is re-established within seconds once the network recovers.

Whether it's the phone disconnecting or the Arduino losing WiFi, it's essentially the same to [Socket.IO](#) — a TCP connection interruption. The only difference is detection latency: when the phone actively disconnects, the browser detects it immediately, while Arduino losing WiFi requires waiting for the [Socket.IO](#) heartbeat timeout (default ~20 seconds) to be detected.

Core Algorithms

The most technically challenging part of this project lies in the firmware — how to simultaneously achieve precise servo control and smooth motor speed regulation within Arduino's limited hardware resources.

This all depends on the carefully designed **hybrid drive mode** and **non-blocking software PWM** algorithm.

1. Timer Conflict Between Servo Library and analogWrite

In the Arduino ecosystem, both the `Servo` library and `analogWrite` function rely on hardware timers to generate PWM signals. When the `Servo` library is imported and attached to a pin, it “monopolizes” the corresponding timer, causing the `analogWrite` function on pins associated with that timer to stop working.

Specifically for this project:

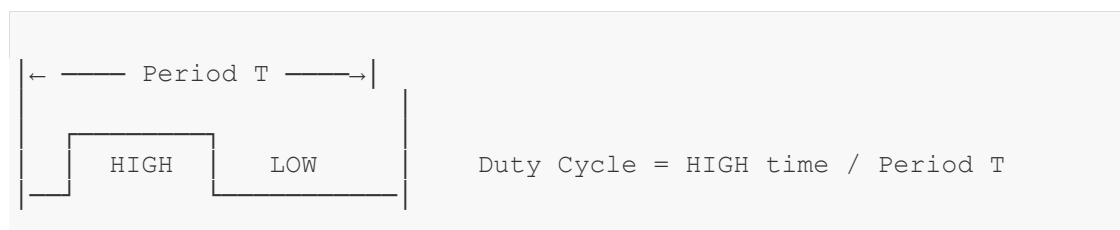
- SG90 servos require the `Servo` library to provide precise 50Hz PWM signals (20ms period) for accurate angle control
- DC motors need PWM signals for speed regulation, typically achieved via `analogWrite`
- However, once the `Servo` library occupies the timer, `analogWrite` on pins D9 and D10 stops functioning

The traditional solution is to move the motors to unaffected pins — but in this project, due to the physical wiring constraints of the expansion board, pins cannot be easily reassigned. Therefore, we need an innovative software-level solution.

2. Non-Blocking Software PWM Algorithm

The core idea: since hardware PWM is occupied by the servo library, we’ll “manually create” PWM waveforms in software.

The essence of PWM (Pulse Width Modulation) is quite simple — within a fixed period, control the proportion of time spent at high level (duty cycle). The higher the duty cycle, the higher the equivalent voltage, and the faster the motor spins.



Below is the complete implementation of the non-blocking software PWM, running in Arduino's `loop()` function:

cpp

```
void loop() {
    Bridge.update();

    // === Non-blocking Software PWM (Core Innovation) ===
    unsigned long now = micros();           // Get current microsecond
    timestamp
    const int period = 2000;                // PWM period 2000µs = 2ms
    (500Hz)
    long cycle_time = now % period;         // Position within current
    cycle (0~1999)

    // Calculate HIGH duration based on target speed
    int on_time_l = abs(target_speed_l) * period / 255;
    int on_time_r = abs(target_speed_r) * period / 255;

    // --- Left wheel control ---
    if (target_speed_l == 0) {
        // Speed is 0: pull both pins LOW, motor stops
        digitalWrite(PIN_L1, LOW);
        digitalWrite(PIN_L2, LOW);
    } else {
        // Determine direction pin based on speed sign
        int pin_pwm = (target_speed_l < 0) ? PIN_L1 : PIN_L2;
        int pin_low = (target_speed_l < 0) ? PIN_L2 : PIN_L1;

        digitalWrite(pin_low, LOW); // Direction pin always LOW
        // Key: determine PWM pin level based on position within
    cycle
        digitalWrite(pin_pwm, (cycle_time < on_time_l) ? HIGH : LOW);
    }

    // --- Right wheel control (reverse logic) ---
    if (target_speed_r == 0) {
        digitalWrite(PIN_R1, LOW);
        digitalWrite(PIN_R2, LOW);
    } else {
        int pin_pwm = (target_speed_r < 0) ? PIN_R2 : PIN_R1;
        int pin_low = (target_speed_r < 0) ? PIN_R1 : PIN_R2;

        digitalWrite(pin_low, LOW);
        digitalWrite(pin_pwm, (cycle_time < on_time_r) ? HIGH : LOW);
    }
}
```

Physical Meaning of Algorithm Parameters:

- **period = 2000 (microseconds):** PWM period of 2ms, corresponding to a 500Hz switching frequency. This frequency far

exceeds the mechanical response speed of DC motors (typically tens of Hz), so the motor perceives a smooth “average voltage” rather than pulsed jitter

- **cycle_time = now % period**: Uses modulo operation to get the current position within the cycle, range 0–1999 microseconds
- **on_time = abs(speed) * period / 255**: Linearly maps the speed value (0–255) to HIGH duration. For example, at speed 128, on_time $\approx 1004\mu\text{s}$, ~50% duty cycle
- **abs(speed)**: Absolute value is used for duty cycle calculation; the sign of speed is only used to select the direction pin

Why is it “Non-Blocking” ?

The key is that the entire algorithm **uses no delay() or any blocking loops**. Each time `loop()` executes:

1. Read current time (`micros()`)
2. Calculate whether to output HIGH or LOW
3. Set pin level
4. Return immediately

This means `loop()` can run at high speed (tens of thousands of times per second), making only one instantaneous decision per iteration.

`Bridge.update()` can also be called on every loop iteration, ensuring communication is never blocked.

Comparison with Traditional `analogWrite`:

Feature	<code>analogWrite</code> (Hardware PWM)	Software PWM (This Approach)
Implementation	Hardware timer auto-toggles pin	Code manually evaluates and toggles pin
CPU Usage	Nearly zero	Calculation needed each loop
Precision	Extremely high (hardware-level)	Depends on loop frequency
Servo Compatibility	✗ Timer conflict	☑ Fully compatible
Use Case	No servo conflict	Servo and motor coexistence

3. Bridge Callbacks and Variable Separation

The firmware adopts a “command-execution separation” architecture — Bridge callback functions only update target variables without performing any actual hardware operations:

cpp

```
void set_motor_l(int speed) { target_speed_l = constrain(speed, -255, 255); }
void set_motor_r(int speed) { target_speed_r = constrain(speed, -255, 255); }
void set_servo_pan(int angle) { servoPan.write(constrain(angle, 0, 180)); }
void set_servo_tilt(int angle) { servoTilt.write(constrain(angle, 60, 120)); }
```

The benefits of this design:

- **Thread Safety:** Variables modified with `volatile` ensure data consistency between interrupt context and the main loop
- **Real-time Performance:** Callbacks return instantly, never blocking the communication protocol stack
- **Safety Limits:** The `constrain` function ensures input values stay within legal ranges, preventing hardware damage

4. Frontend Command Translation Pipeline

From the user pressing a direction button to the motor spinning, the entire command flows through four stages:

```
User presses "↑" button
    ↓
Frontend: socket.emit('move', 'up')           // Stage 1: Semantic
command
    ↓ WebSocket
Middleware: move_car('up') → sl=150, sr=150    // Stage 2: Speed
calculation
    ↓ Bridge.call("motor_l", 150)
    ↓ Bridge.call("motor_r", 150)
Firmware: target_speed_l=150, target_speed_r=150 // Stage 3:
Parameter caching
    ↓ loop() → Software PWM
Left wheel forward @59% duty, Right forward @59% // Stage 4:
Physical execution
```

The advantage of this layered pipeline is that each layer focuses only on its own responsibility: the frontend handles user interaction, the middleware handles business logic, and the firmware handles hardware timing. When modifying one layer's implementation, other layers remain unaffected.

Three layout configurations cover all usage scenarios:

1. **PC/Tablet Landscape** (default): Three columns side by side, maximizing video area

2. **Phone Portrait:** Single-column rearrangement, video on top (the information users care about most), controls at bottom (thumb operation zone)
3. **Short Landscape:** Between the two, maintaining three columns but with compressed dimensions

The brilliance of the `order` property: the DOM order in HTML is driving → video → gimbal (left → center → right), but in portrait mode, `order` rearranges them to video → gimbal → driving — reorganizing the layout purely through CSS without changing the HTML structure.

Usage Instructions and Notes

Initial Setup:

1. Connect the motors, servos, and camera to the corresponding ports on the expansion board according to the wiring diagram
2. Connect the expansion board to the Arduino UNO Q via the C-to-C data cable
3. Connect the Arduino UNO Q to a computer via USB
4. Open Arduino App Lab and load the `tracked_plz` project
5. Upload the sketch firmware and Python program
6. Configure WiFi credentials

Important: Power-On Sequence Note

Before powering on, you **must first disconnect** the C-to-C data cable between the expansion board and Arduino UNO Q (unplug either end). Reconnect it only after the board has finished booting. Otherwise, the USB camera may not be recognized properly.

Starting the Program:

Method 1: Via App Lab

1. Click "Run" in App Lab to start the program

Method 2: Via SSH

1. SSH into the Arduino UNO Q
2. Navigate to the app directory: `cd ArduinoApps`
3. Start the program: `arduino-app-cli app start tracked_plz`
4. Seeing ✓ App "tracked_plz" started successfully indicates a successful launch

Accessing the Control Interface:

1. After launch, visit `http://<robot IP>:7000` in a web browser
2. A green "ONLINE" status indicator in the top-right corner confirms successful connection
3. The central video panel should display the live camera feed

Driving Controls:

1. Use the Drive Speed slider on the left/bottom to adjust speed
(recommend starting at 100–150 for indoor use)
2. Hold direction buttons to move, release to auto-stop
3. In emergencies, click the central red STOP button

Gimbal Controls:

1. Drag the PAN slider to control the camera's horizontal viewing angle

2. Drag the TILT slider to control the camera's vertical pitch
3. Initial position: PAN=90° (straight ahead), TILT=90° (horizontal)

Servo Center Calibration (For First-Time Installation):

A dedicated calibration module is located below the gimbal control panel, used to ensure precise servo horn positioning during installation:

1. Connect the servo hardware but **do not tighten** the horn fixing screws
2. Start the program and open the web console
3. Click the "Start Calibration" button to auto-center both axes to 90°
4. Use the $\pm 1/\pm 5$ degree fine-tuning buttons for precise adjustment
5. Once satisfied, secure the horns and tighten the screws
6. Click "Finish Calibration" to exit calibration mode

Viewing AI Detection:

1. The HUD area at the bottom of the video feed displays detection results in real time
2. Default display shows "SYSTEM READY" ; after detecting a target, it shows `[TARGET] object name`
3. The confidence threshold is 0.5, adjustable in the Python code

Troubleshooting

Q: The page loads but the video stream shows blank?

A: Check whether the camera is recognized by the system. Execute `ls -l /dev/video*` via SSH — you should normally see 4 device nodes. If there are 2 or fewer, follow the power-on sequence again: power off → unplug C-to-C cable → power on → wait for boot → reconnect C-to-C cable.

Q: Only one side of the wheels is spinning?

A: Check battery level. The A4950 motor driver may not be able to drive both channels simultaneously at low voltage. Recharge and try again.

Q: What to do about servo jitter?

A: The code already uses hybrid drive mode optimization for anti-jitter. If jitter persists, check whether the servos share a power line with the motors. Consider providing separate power for the servos or adding a 100µF electrolytic capacitor in parallel at the servo power input for filtering.

Q: Phone portrait layout arrangement looks wrong?

A: Ensure the latest `style.css` is referenced. The `@media` queries in the stylesheet automatically move the video to the top and buttons to the bottom when the screen width is $\leq 900\text{px}$ in portrait orientation.

Q: Can it be used without internet?

A: Yes. As long as the `assets/` folder contains the local `socket.io.min.js` file, it can operate normally within a local network without internet access.

Q: Will control disconnect during unstable network?

A: It won't disconnect permanently. The system has a built-in weak network auto-reconnection mechanism — after a network interruption, the status indicator turns yellow and blinks showing the retry count. Once the network recovers, the connection is automatically re-established (indicator returns to green) without requiring a manual refresh. It retries up to 30 times; the retry strategy can be adjusted via the `io({...})` configuration at the top of `script.js`.
