Assignment: Distributed Vision-Control System (Face-Locked Servo)

Instructor: Gabriel Baziramwabo

Keywords: *Computer Vision, Distributed Systems, MQTT Messaging, Publish–Subscribe Model, Real-Time Control, WebSocket Communication, Edge Actuation, Embedded Networking*

1. Objective

Design and implement a complete distributed vision-control system using:

- A PC-based vision engine
- An ESP8266 edge controller
- A cloud backend service (VPS)
- A real-time web dashboard

The system must detect and track a face, publish movement commands via MQTT, and control a servo motor to respond accordingly.

The system must allow multiple teams to operate simultaneously on the same MQTT broker without interfering with one another.

2. Development Phases

This project will be implemented in **two phases**.

Phase 1: Open-Loop Actuation (Simulation Stage)

- The camera remains fixed on the PC.
- The servo motor has a stick/pointer attached.
- The servo rotates based on face movement.
- The camera frame does NOT change when the servo rotates.

This phase validates:

- MQTT communication
- Topic isolation
- Servo actuation logic
- WebSocket real-time updates
- System stability

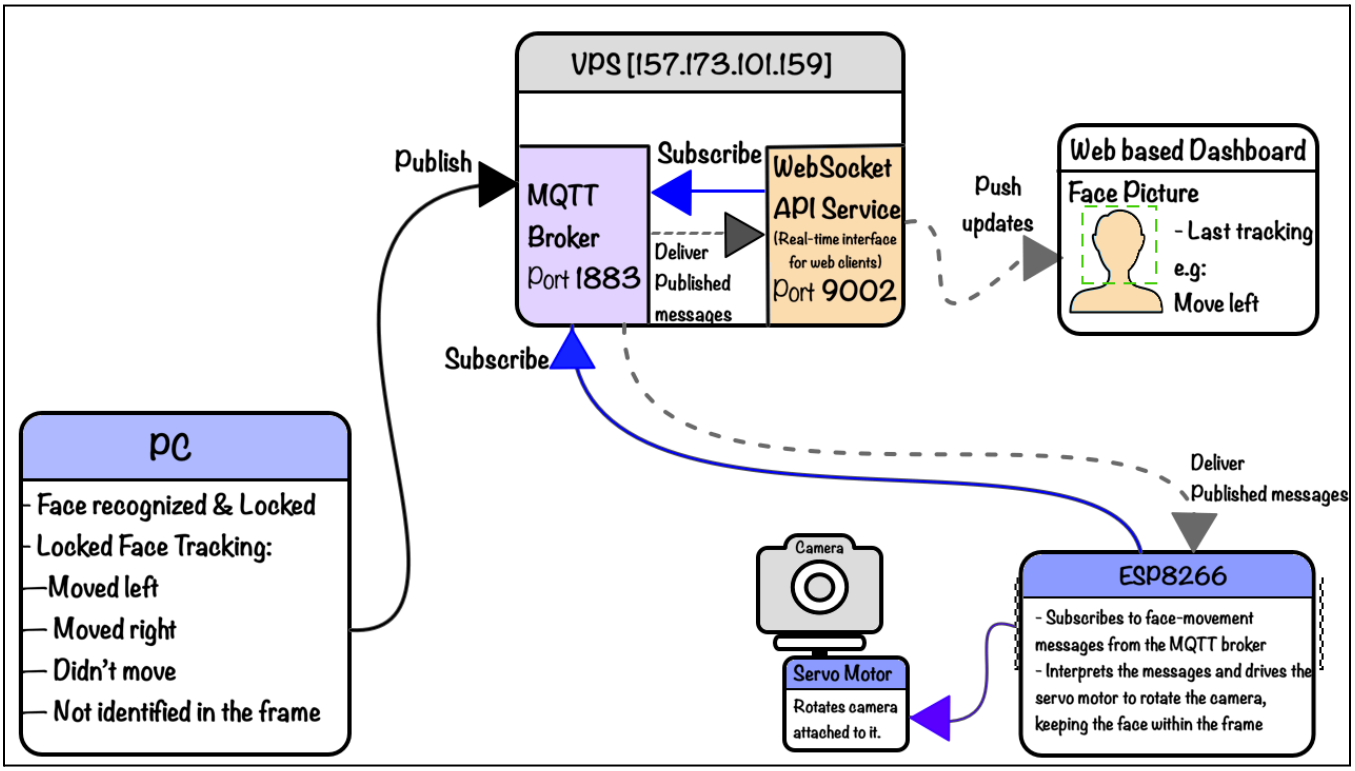
Phase 2: Closed-Loop Tracking (Mechanical Integration)

- The camera will be mounted directly on the servo.
- When the servo rotates, the camera rotates.
- The vision frame changes dynamically.
- The system becomes a true feedback loop.

This phase validates:

- Mechanical integration
- Closed-loop tracking stability
- Smooth servo control
- Jitter minimization

3. System Architecture



Distributed Vision-Control Architecture

Your implementation must follow this architecture strictly:

PC (Vision Node)

- Captures camera frames
- Detects and tracks a face
- Determines movement state:
    - MOVE_LEFT
    - MOVE_RIGHT
    - CENTERED
    - NO_FACE
- Publishes movement messages via MQTT

Must NOT:

- Communicate directly with ESP8266
- Communicate directly with the browser
- Use HTTP or WebSocket for device communication

ESP8266 (Edge Controller)

- Subscribes to movement messages from MQTT
- Interprets movement instructions
- Drives the servo motor accordingly

Must NOT:

- Use HTTP
- Use WebSocket
- Connect directly to browser

Backend API Service (VPS)

- Hosts MQTT broker (Port 1883)
- Hosts WebSocket API service (Port 9002)
- Subscribes to MQTT movement messages
- Pushes real-time updates to browsers via WebSocket

The backend relays real-time events.

Web Dashboard (Browser)

- Connects to backend via WebSocket
- Displays:
    - Last movement status
    - Tracking state
    - Timestamp
- Updates in real-time

Must NOT:

- Connect directly to MQTT
- Use polling

4. Critical Rule: MQTT Topic Isolation

All teams share the same MQTT broker.
To avoid conflicts, each team must use a unique topic namespace.
Team Identifier: Each team must choose a unique team_id
(example: team01, alpha, y3_grp2)

Base Topic:

vision/<team_id>/

4.1. Required MQTT Topics

Movement Message (PC → Broker)

vision/<team_id>/movement

Payload example:

```
{
  "status": "MOVE_LEFT",
  "confidence": 0.87,
  "timestamp": 1730000000
}
```

Optional Heartbeat (Any Node → Broker)

vision/<team_id>/heartbeat

Payload example:

```
{
  "node": "pc",
  "status": "ONLINE",
  "timestamp": 1730000000
}
```

4.2. Forbidden Practices

Using generic topics such as:

vision/ movement
movement
servo

Subscribing to wildcard topics such as:
vision/#
#

Publishing or subscribing to another team's namespace. Any team interfering with others will lose marks.

5. WebSocket Requirements

The backend must:

- Subscribe to vision/<team_id>/movement
- Push movement updates to browsers via WebSocket

The dashboard must:

- Receive real-time updates via WebSocket
- Not use polling

6. Submission Requirements

Each team member must individually push the project files to their own GitHub repository.

The GitHub repository link must be submitted individually via a Google Form shared later.

The live dashboard URL (hosted on the VPS or any other hosting platform) must be submitted via the same Google Form.

A quiz link will also be included in the same form for assessment.

⚠️ Repositories must be:

- Public
- Well structured
- Clearly documented (README.md is mandatory)

The README must explain:

- Architecture
- MQTT topics
- Setup instructions
- Dependencies
- How to run each component

7. Evaluation Focus

Assessment will emphasize system engineering discipline, architectural correctness, and real-time performance.
Evaluation will consider:

- Correct and appropriate use of MQTT and WebSocket
- Proper MQTT topic isolation on the shared broker
- Complete end-to-end system functionality
- Real-time responsiveness (no noticeable delay or message flooding)
- Servo stability and smooth movement (no uncontrolled jitter or oscillation)
- System stability in a shared-broker environment
- Code clarity, structure, and repository organization

📌 **Note:** For the ESP8266, the Arduino IDE may be used; however, **MicroPython is**

strongly encouraged.

You're advised to review the MicroPython resources available on my [Gabriel Baziramwabo] **ResearchGate account** and the **BenaxMedia YouTube Channel** for guidance and implementation support.

Golden Rule

> ***Vision computes. Devices speak MQTT. Browsers speak WebSocket. The backend relays in real time.***

Final Note

> ***This assignment is designed to test distributed system design, real-time control, and integration discipline. Architectural violations will be penalized even if the system appears to work. Build it like engineers.***