LAM: HARDWARE HUSTLE

System Architecture, Firmware Logic & Integration

Indian Institute of Technology, Guwahati Team 5

November 30, 2025

1 Problem Definition

The objective of this project is to architect and deploy a cooperative multi-agent robotic system within a **"Smart Arena"** environment. The successful execution of the mission relies on the seamless integration of mechanical design, embedded control logic, and human-in-the-loop interaction.

The operation is strictly enforced within a **10-minute operational window**, requiring the synchronized operation of two distinct robotic agents:

- **Autonomous Agent (ALFR):** Logic carrier and trigger mechanism.
- **Teleoperated Agent (SARM):** Support and maintenance unit.

2 System Agents & Specifications

2.1 The Autonomous Line Follower Robot (ALFR)

- **Primary Role:** Acts as the logic carrier and trigger mechanism for the arena's embedded systems.
- **Locomotion & Control:** Operates autonomously utilizing a PID-based control loop to track a predefined path.
- **Constraint Handling:** Possesses Ultrasonic/IR sensing capabilities to detect dynamic obstacles. Upon detection, it halts navigation and enters a "wait state" until the path is cleared.
- **Software Stack:** Simulation and control logic leverage the **ROS 2** (Robot Operating System) framework.

2.2 The Single Arm Robot (SARM)

- **Primary Role:** Functions as the support and maintenance unit responsible for dynamic environment manipulation.

- **Locomotion:** Utilizes Mecanum or Omni-wheels to achieve holonomic motion (omnidirectional maneuverability), allowing for precise alignment in constrained spaces.
- **Manipulation:** Equipped with a multi-degree-of-freedom (DoF) robotic arm designed to identify, grasp, and relocate physical obstacles.
- **Control Interface:** Manually teleoperated via a robust communication link between the operator and hardware.

3 The Smart Arena Infrastructure

The arena is an active circuit comprising three sequential "Gates" requiring electromechanical integration:

1. **Gate 1 (Fluid Dosing System):** Upon ALFR arrival, a peristaltic pump activates to dispense a precise volume of **125 ml** of fluid. Exact flow control logic terminates pumping immediately upon reaching the target volume.
2. **Gate 2 (Visual Feedback):** A sensor-triggered "LAM" shaped LED array illuminates as the ALFR passes, serving as a visual checkpoint.
3. **Gate 3 (Verification & Data Display):** A closed-loop validation system where a load cell measures the total weight. If the weight is within tolerance, the credentials are displayed on an LCD; otherwise, an error state is triggered.

4 Operational Workflow

The project lifecycle follows a strict state-machine logic:

1. **Initialization:** ALFR begins autonomous tracking.
2. **Obstacle Detection:** ALFR detects an obstacle and pauses.
3. **Manual Intervention:** SARM operator navigates to the coordinate, manipulates the obstacle, and clears the path.
4. **Resumption:** ALFR re-evaluates the path and resumes tracking.
5. **Process Execution:** ALFR triggers Gate 1 (Pump), Gate 2 (LED), and docks at Gate 3.
6. **Validation:** The system computes payload mass and updates the LCD interface.

5 Technical Implementation

5.1 Mechanical Implementation

Manual-Operated SARM:

- **Chassis:** Specialized platform with Mecanum/Omni wheels.
- **Manipulator:** Constructed using 3D-printed parts and servo brackets. The critical challenge was the gripper modification; jaw geometry was customized to securely grasp specific obstacle shapes found in Stages 1, 2, and 3.
- **Actuation:** High-torque **MG996R (11kg-cm)** and **Waterproof (20kg-cm)** servos handle the leverage required at full extension.

ALFR (Autonomous):

- **Chassis:** 3D printed body designed to fit arena track constraints.
- **Fluid Integration:** Dedicated mounting point engineered to receive exactly 125 ml of fluid.

5.2 Electronics Architecture

The electronics are divided into three isolated subsystems:

1. **SARM Electronics:**
   - **Controller:** ESP32 Development Board.
   - **Drive System:** Orange Johnson DC Geared Motors (12V, 600rpm) driven by L298N Motor Drivers.
   - **Communication:** Bluetooth signals received from a mobile app.
2. **ALFR Electronics:**
   - **Line Detection:** Smart Elex RLS-08 Analog & Digital Line Sensor Array.
   - **Safety:** HC-SR04 Ultrasonic sensors and IR modules serve as emergency stops, cutting power if thresholds are breached.
   - **Power:** 12V to 5V buck converter for logic boards; motors draw directly from the battery.
3. **Arena Automation:**
   - **Gate 1 (Fluid):** 5V Single Channel Relay controlling a 12V Peristaltic Pump.
   - **Gate 3 (Weighing):** 5kg Load Cell integrated with an HX711 amplifier module.
   - **Output:** 1.8-inch TFT LCD Module.

6 Firmware Logic and Algorithms

6.1 ALFR Firmware (PID Architecture)

The firmware integrates sensor data, navigation algorithms, and safety protocols into a real-time loop.

6.1.1 Sensor Fusion and Calibration

- **Dynamic Thresholding:** `calibrateSensors()` samples the environment for **400 cycles** to find min/max reflectance values. The trigger threshold is the mathematical average of these extremes.
- **Weighted Average Positioning:** The `readSensors()` function assigns weights ranging from **-3.5 (left) to +3.5 (right)**. The position is calculated as:

$$\text{Position} = \frac{\sum(\text{SensorValue} \times \text{Weight})}{\sum(\text{ActiveSensors})} \tag{1}$$

6.1.2 Obstacle Detection and Safety

- **Ultrasonic Interface:** The `getDistance()` function polls the HC-SR04 sensor.
- **Stop-and-Wait Logic:** If an object is detected within **10 cm** (`STOP_DISTANCE`), `stopMotors()` is called. The system enters a `while` loop, pausing navigation until the obstacle is removed.

6.1.3 PID Control Algorithm

The core navigation minimizes `Error = Current_Position - Setpoint (0)`.

- **Derivative Smoothing:** Uses a Low Pass Filter ($\alpha = 0.9$) to prevent motor jitter from sensor noise.
- **Integral Windup:** The Integral term is constrained between **-200 and 200** to prevent overshoot.
- **Dynamic Tuning:** Adaptive gain scheduling halves K_p and K_d on straight lines, allowing aggressive correction only on turns.

6.1.4 *Motor Mixing and Failsafe*

- **Differential Drive:**

$$\text{LeftSpeed} = \text{BaseSpeed} + \text{PID\_Output}$$
$$\text{RightSpeed} = \text{BaseSpeed} - \text{PID\_Output}$$

- **Cornering:** If `fabs(error) > 2.5`, BaseSpeed is halved to prioritize traction.
- **Lost Line Recovery:** If `!s.onLine`, the robot pivots in the direction of the last known error (`prevError`) until the line is re-acquired.

6.2 SARM Firmware Implementation

- **Wireless Interface:** Uses Bluetooth Serial. Parses packets to distinguish between Direction Commands (0-14) and Speed Commands (≥ 16).
- **Motor Drive Abstraction:** The `setMotor` function utilizes ESP32's hardware LEDC peripheral to generate **30kHz, 8-bit PWM signals**.
- **Holonomic Kinematics:**
  - *Linear:* All wheels drive with same polarity.
  - *Strafing:* Diagonal pairs (e.g., Front-Left/Back-Right) drive forward while opposing pairs drive backward to cancel forward force vectors.
  - *Zero-Radius Turn:* Left and right sets drive in opposite directions.

6.3 Robotic Arm Firmware (5-DOF Bluetooth Control)

- **System Architecture:** Controls 5-DOF (Base, Shoulder, Elbow, Wrist, Gripper) using the `ESP32Servo` library. Servos are mapped to GPIO pins **13, 12, 14, 27, and 26**.
- **Incremental Control Logic:** Uses Relative Step Control via `handleCommand()`. Single-byte integer commands (1-10) adjust joints by $\pm 5°$:
  - **Odd Integers (1, 3...):** Increment angle.
  - **Even Integers (2, 4...):** Decrement angle.
- **Safety:** `adjustServo()` uses `constrain(angle, 0, 180)` to prevent mechanical stalling.
- **Interface:** ESP32 acts as a Bluetooth slave ("AG_Arm"). A blocking delay (`speedDelay = 15ms`) acts as a rate limiter.

6.4 Arena Automation Firmware

- **Sequential State Machine:**
  - *Stage 1:* Triggers peristaltic pump for `pumpRunTimeMs` (approx. 125ml).
  - *Stage 2:* Activates "LAM" LED array.
  - *Stage 3:* Stabilizes robot and reads HX711 Load Cell.
- **Active-LOW Logic:** Sensors/Relays are driven LOW to turn ON, ensuring a safe "off" state during boot-up.
- **Non-Blocking Architecture:** Uses `millis()` instead of `delay()` to allow continuous sensor polling.
- **Signal Integrity:** `handleDebounce()` implements a **50ms** software filter for IR inputs.
- **Final Output:** LCD displays "Success" if weight is **120g–130g**, otherwise "Weight Mismatch".

7 Conclusion

The implementation demonstrated the efficacy of the multi-agent system. The SARM's holonomic kinematics allowed for precise obstacle clearance without disrupting the critical path, while the ALFR's PID algorithms ensured stable high-speed tracking.

Crucially, the Arena Automation tied the system together. The non-blocking state machine ensured that fluid delivery and weight verification were handled with high accuracy. The final integration of the Load Cell and LCD successfully verified the system's end-to-end performance, confirming that the fluid dispensed met the strict 125ml requirement.