Hardware Hustle: Lam Research Challenge 2025

Design Technical Documentation

Submitted By:

Circuit_Raja

Team ID: LRC-U-0053-6

November 2025

Contents

1 System Overview

1.1 Problem Statement Overview

The Hardware Hustle challenge requires two coordinated robots—a manually controlled Single Arm Robot (SARM) and an autonomous Advanced Line Follower Robot (ALFR)—to complete a sequence of tasks inside a predefined arena. ALFR must follow the line path, halt at obstacles, trigger Gate 1 for fluid dispensing, Gate 2 for LED activation, and park at Gate 3 for load-cell weight verification. SARM's role is to remove obstacles blocking ALFR's route. The entire system must operate accurately and finish within the 10-minute time limit.

1.2 Design Philosophy: Decentralized & Deterministic

Our team adopted a philosophy of **"Decentralized Architecture & Deterministic Control."**

Analyzing the complexity of the Hardware Hustle challenge, we determined that a single microcontroller loop would introduce unacceptable latency when handling simultaneous tasks—such as inverse kinematics, sensor interrupts, and wireless communication.

Therefore, we engineered a system where critical subsystems operate on **independent processing cores**. This ensures that heavy computational tasks never block safety-critical functions like obstacle detection. We prioritized predictability over raw speed, using mathematical models (PID) and hardware interrupts to guarantee the robot behaves exactly the same way in every run.

1.3 Technical Implementation Strategies

1. **SARM: Dual-Core Decentralization**
To ensure zero latency during the manual operation phase, the Single Arm Robot (SARM) utilizes a split-architecture approach, as detailed in our firmware source files:

- **Base Controller (Mobility):** A dedicated ESP32 handles the Mecanum wheel kinematics. The code processes directional vectors (Forward, Backward, Strafe Left/Right) and maps them to the L298N drivers independently of the arm's status.
- **Arm Controller (Manipulation):** A second ESP32 manages the 5-DOF servo arm. We implemented **"State Macros"** such as `moveArmContracted()` and `moveArmExpanded()`. These macros allow the operator to trigger complex folding sequences with a single button press, ensuring the gripper geometry is always perfectly aligned for obstacle pickup.

2. **ALFR: Interrupt-Driven Safety Layer**
For the Autonomous Line Follower, we moved away from standard polling loops to achieve deterministic safety:

- **Hardware Interrupts:** We utilized a hardware Interrupt Service Routine (ISR) for the ultrasonic sensor (`echoISR`). This allows the `emergencyStop` flag to trigger in microseconds, instantly overriding the motor drivers the moment an object is detected, regardless of the main loop's cycle time.
- **PID Stabilization:** We implemented a Proportional-Derivative control loop to handle the arena's S-curves.

$$PID_{value} = (K_p \times Error) + (K_d \times (Error - PreviousError)) \quad (1)$$

With tuned constants ($K_p = 40, K_d = 6$), the robot adjusts motor differential speed dynamically based on the weighted average of the 8-channel sensor array.

3. **Arena: Non-Blocking Automation**
The Arena Controller serves as the central logic node. To satisfy the Gate 2 requirement without freezing the system, we utilized a `millis()` based non-blocking timer rather than `delay()`. This ensures the controller can still read the Gate 3 Load Cell sensors even while the Gate 2 LEDs are active.

2 Advanced Line Follower Robot (ALFR)

2.1 Mechanical Integration

The ALFR chassis is a custom 3D-printed assembly designed for low center-of-mass stability.

- **Sensor Placement:** The 8-channel analog line sensor array is mounted at the extreme anterior of the chassis. This forward offset provides a crucial "look-ahead" buffer, allowing the PID algorithm to react to the arena's sharp radii (R140mm and R150mm) before the drive wheels reach the curve.
- **Obstacle Detection:** An HC-SR04 ultrasonic sensor is mounted centrally at a height of 40mm, optimized to detect the standard obstacle blocks used in Stages 1, 2, and 3.

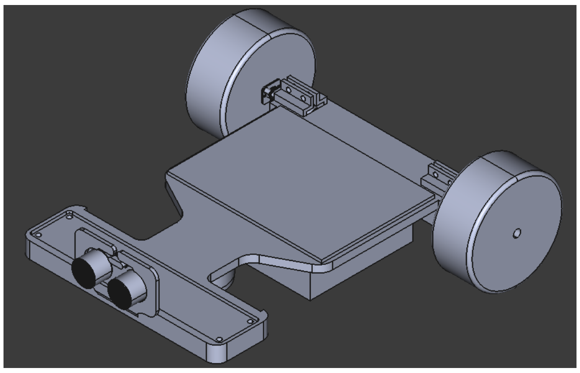

Figure 1: ALFR CAD

2.2 Control Algorithms

Unlike basic threshold-based followers, our ALFR utilizes a deterministic **PD (Proportional-Derivative)** control loop to ensure smooth trajectory maintenance.
1. PD Implementation
The core navigation logic calculates the positional error relative to the center of the sensor array. This error is processed through the following control equation:

$$Correction = (K_p \cdot Error) + (K_d \cdot (Error - Error_{prev})) \tag{2}$$

Based on our testing, we tuned the gain constants to:

- **Proportional Gain ($K_p = 40.0$):** Provides immediate reaction to track deviation.
- **Derivative Gain ($K_d = 6.0$):** Dampens oscillations, preventing the robot from "fishtailing" on straight sections.

This mathematical approach allows the ALFR to vary the motor differential speed dynamically ('dynBase'), maintaining a high base speed while slowing down only as much as necessary for turns.

2. Interrupt-Driven Obstacle Avoidance
To strictly adhere to the rule that the ALFR must "Wait if Obstacle Present" (Rulebook Page 4), we implemented a hardware-interrupt safety layer.

Instead of polling the sensor in the main loop (which can be delayed by PID calculations), we attached an **Interrupt Service Routine (ISR)** to the Ultrasonic Echo pin:

```
void echoISR() {
    // Calculates distance immediately upon signal return
    if (distance < 20) emergencyStop = true;
}
```

This architecture ensures that if an obstacle is placed within 20cm, the robot halts in microseconds, independent of the main processor load.

2.3 Code Uniqueness

- **Memory Buffer:** The system utilizes a `lastTurn` variable. If the robot completely loses the line (e.g., at a sharp acute angle), it recalls the direction of the last known error and performs a **rotational search** in that direction until the line is re-acquired.
- **Hardware-Level Pin State Check:** Rising/falling edges of the echo signal are detected by checking

  ```
  if (PIND & (1 << PIND3))
  ```
  This is faster and more precise than using digitalRead(), **giving microsecond accuracy** to the distance measurement.
- **Constrained Motor Control for Safe Behavior:** Motor outputs are bounded using `constrain()` to ensure safe acceleration and consistent turning behavior. This prevents **electrical and mechanical stress** on the motors during sharp PID corrections.

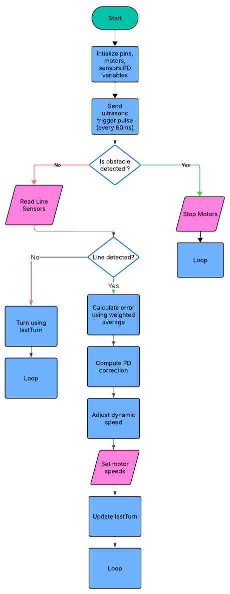

Figure 2: ALFR Algorithm flowchart

3 Single Arm Robot (SARM) Design

3.1 Control Architecture: Dual-Core Decentralization

To mitigate latency and ensure real-time responsiveness during the manual operation phase, we developed a **Decentralized Control System**. Unlike traditional single-board designs, our SARM separates mobility and manipulation into independent processing nodes:

- **Node A (Mobility Controller):** An ESP32 running `SARM_base_code.ino` is dedicated exclusively to the Mecanum wheel inverse kinematics. It parses directional vectors (Forward, Strafe, Rotate) via Bluetooth without servo signal interference.
- **Node B (Arm Controller):** A secondary ESP32 running `Final_SARM_arm_code.ino` manages the 5-DOF robotic arm. This separation ensures that the PWM signals for the servos remain stable even when the motors are drawing high current during rapid acceleration.

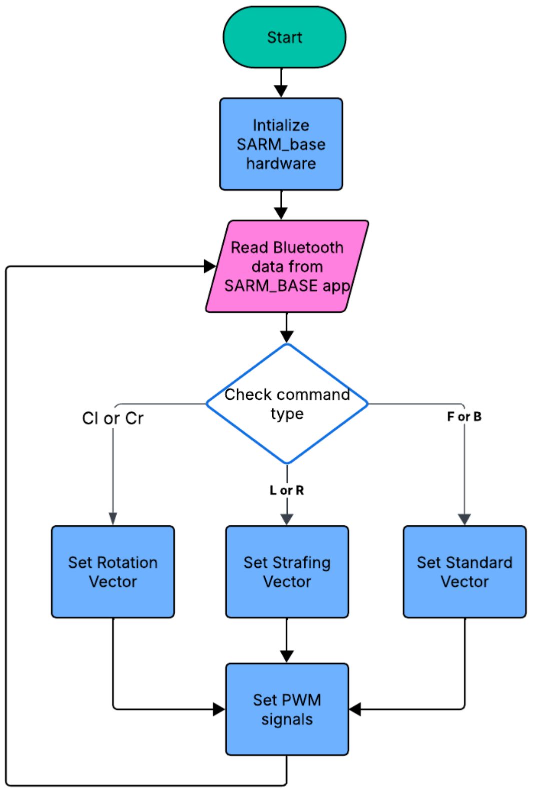

Figure 3: Node A Algorithm flowchart

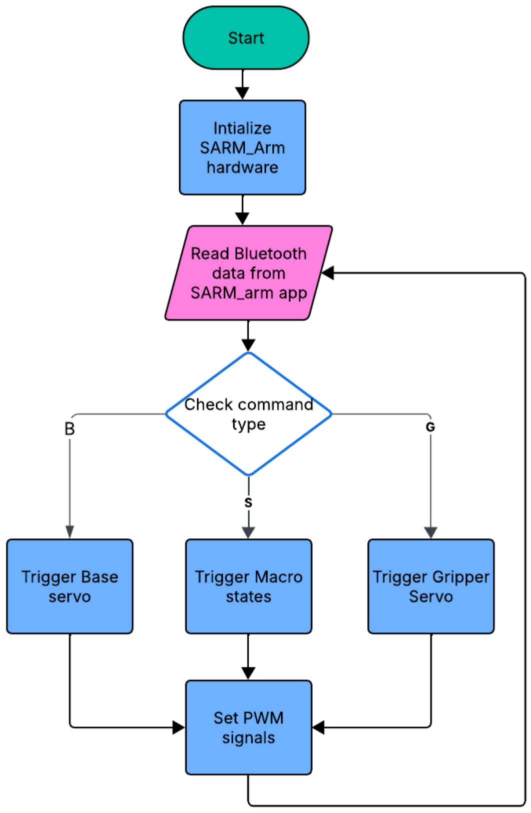

Figure 4: Node B Algorithm flowchart

3.2 Omni-Directional Mobility

The base utilizes four independently driven Mecanum wheels. The drive logic leverages vector cancellation to achieve holonomic movement. As implemented in our firmware function `void left()`:

```
// Vector Math for Left Strafe
Motor1 (Front-Left):  Backward
Motor2 (Front-Right): Forward
```

```
Motor3 (Rear-Right):  Forward
Motor4 (Rear-Left):   Backward
```

This configuration allows the operator to align with obstacles laterally without needing to perform complex multi-point turns, significantly reducing cycle time.

3.3 Automated Manipulation Logic

To minimize operator fatigue and error, we moved away from raw manual control for every joint. Instead, we implemented State Macros pre-calculated geometric configurations hard-coded into the firmware.

- **Contracted State (Safety):** Defined in code as `contracted_shoulder = 140`, this macro folds the arm entirely within the chassis footprint. This allows the robot to travel at maximum speed without the arm changing the center of gravity or colliding with arena walls.
- **Expanded/Grip State:** Defined as `expanded_elbow = 30`, this macro automatically positions the end-effector at the exact height of the obstacle blocks ($Z \approx 40mm$), removing the need for the operator to manually estimate depth.

3.4 Dual-Pilot Operational Strategy

Innovation: Two Apps, Two Pilots, Zero Lag

To overcome the inherent latency limitations of standard Bluetooth stacks and reduce the cognitive load on a single operator, we engineered a **Split-Control HMI (Human-Machine Interface)**.

Instead of a single app, we developed two distinct custom Android applications:

1. **Pilot App (Navigation):** Pairs exclusively with the Base ESP32 (“SARM_BASE”). This allows the primary pilot to focus solely on high-speed maneuvering and obstacle approach vectors.
2. **Co-Pilot App (Manipulation):** Pairs exclusively with the Arm ESP32 (“ESP32_ARM”). This allows the secondary pilot to trigger the gripper macros (Open/Close, Fold/Extend) independently.

Strategic Advantage: This ‘Rally Car’ style coordination (Driver + Navigator) allows the arm to begin unfolding *while* the base is still moving toward the target. This parallel execution effectively doubles our operational bandwidth and significantly reduces the total cycle time per obstacle compared to a single-pilot workflow.

3.5 Gripper Design & Modification

The standard gripper design provided in the reference kit was analyzed and found to lack sufficient friction for the arena obstacles. To address this, we engineered the following modifications:

1. **Adaptive Contact Geometry:** Unlike a rigid geometric modification, we retained the standard gripper structure but integrated a **compliant interface**. The compressible nature of the foam pads allows the jaw profile to dynamically deform and mold itself to the obstacle’s shape, effectively increasing the contact surface area upon closure.

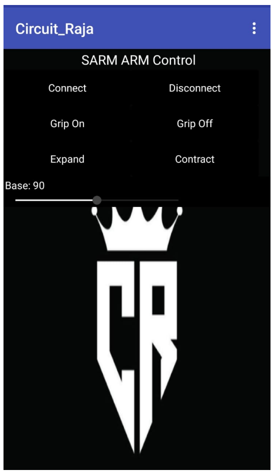

Figure 5: SARM Arm GUI

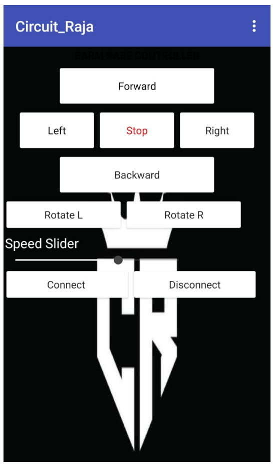

Figure 6: SARM Base GUI

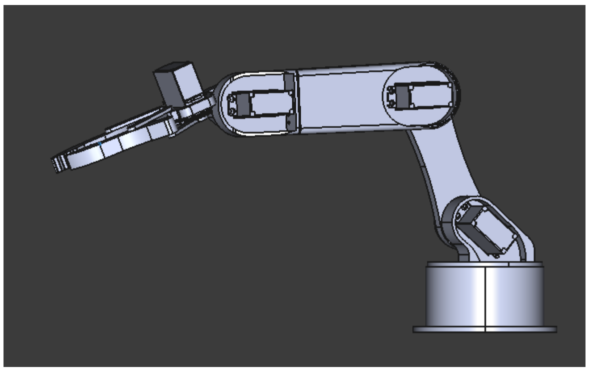

Figure 7: ARM CAD

2. **Surface Material Enhancement:** We applied high-density adhesive foam padding to the inner jaw faces. This modification significantly increases the **coefficient of friction** compared to bare PLA and provides mechanical damping, ensuring the obstacle does not slip or vibrate loose during high-speed acceleration.

3. **Kinematic Redesign (5-Bar to 4-Bar Conversion):** The original gripper design utilized a 5-bar linkage mechanism, which is under-actuated when driven by a single motor. To resolve this instability, we rigidly fixed two of the intermediate links at the pivot joint.

4 Arena Circuitry & Automation Design

4.1 Centralized Arena Control Architecture

To ensure synchronized game states across all three stages, we implemented a **Centralized Event-Driven Architecture**. A single Microcontroller Unit (MCU) monitors the state of all three gates simultaneously.

Unlike basic sequential systems, our firmware utilizes **Non-Blocking I/O** logic. This allows the system to manage the Gate 2 LED timers while simultaneously polling the Gate 3 Load Cell data, preventing any blind spots in the sensing loop.

4.2 Gate 1: Precision Fluid Delivery System

Objective: Dispense exactly 125ml of fluid upon ALFR detection.

- **Hardware Integration:** We utilized a 5V Relay Module (controlled via Pin 14) to switch the high-current DC Peristaltic Pump. The trigger is a digital IR sensor (Pin 16) configured as `INPUT_PULLUP` (Active LOW).
- **Flow Control Logic:** Instead of relying on variable voltage, we operate the pump at a fixed 12V to ensure a constant flow rate (Q). The dispense volume is controlled purely by time (t):

$$t_{dispense} = \frac{V_{target}}{Q_{pump}} \tag{3}$$

The code waits for the `sensorState == LOW` signal to close the relay circuit.

4.3 Gate 2: Timer-Based LED Activation

Objective: Illuminate the Lam LED array for 5 seconds without blocking system processes.

Innovation - Non-Blocking Timer: Standard `delay(5000)` functions would freeze the processor, preventing it from detecting the robot at Gate 3 if it moved quickly. To solve this, we implemented a `millis()` based timer in the code:

```
if (sensorState2 == LOW) {
    relay2Active = true;       // Set Flag
    relay2TimerStart = millis(); // Start Stopwatch
}
// Check time difference in every loop cycle
if (relay2Active && (millis() - relay2TimerStart >= 5000)) {
    turnOffLEDs();             // Auto-Reset
}
```

This ensures the visual effects (Gate 2) do not interfere with the critical data collection (Gate 3).

4.4 Gate 3: Validation Node (Load Cell & LCD)

Objective: Verify fluid weight and display Team Identity.

1. **Weight Sensing (HX711):** We interfaced a 5kg Load Cell via an HX711 24-bit ADC (Pins 32/33). To ensure accuracy despite vibration, the code takes a 10-sample moving average (`scale.get_units(10)`) before validating the success condition.

2. **Visual Feedback (ST7735):** Upon successful weight verification, the system drives an ST7735 TFT Display via SPI. As a unique team identifier, the display renders:

   - Team Handle: **"Circuit_RAJA"**
   - Custom Bitmap: `smile_bmp` (Visual success indicator)
   - Real-time Weight Value

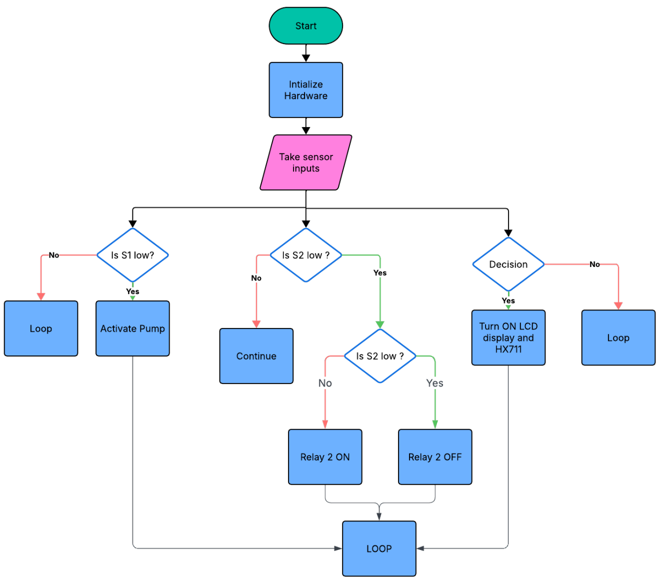

Figure 8: Arena Algorithm flowchart

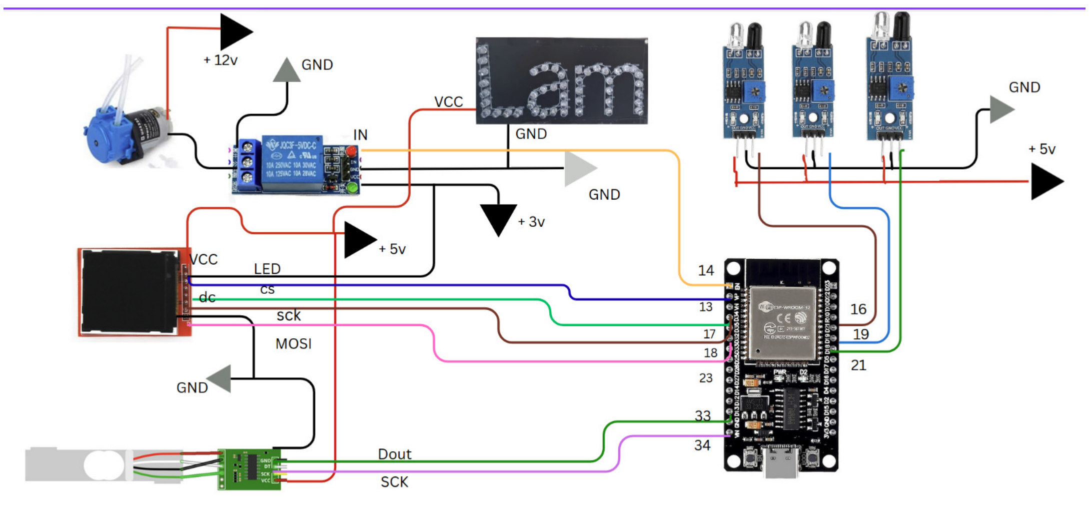

Figure 9: Arena Circuit Schematic

5 Peristaltic Pump Development

5.1 Mechanical Design Principle

To achieve the specific fluid delivery requirement of 125 mL at Gate 1, we engineered a custom positive displacement pump. The mechanism relies on a rotating three-roller configuration that progressively compresses a flexible silicone tube against a circular stator.

- **Stator Geometry:** Designed with a track radius (R) of 25 mm to support the tubing without over-compression.
- **Rotor Configuration:** A three-roller design was selected to ensure continuous flow and prevent backflow (siphoning) when the motor stops[cite: 273].
- **Coupling:** A custom-designed 3D-printed mount aligns the 600 RPM DC motor shaft directly with the rotor assembly[cite: 274, 288].

5.2 Flow Rate Mathematics

To strictly control the dispensed volume, we derived the theoretical flow rate based on the tubing geometry (2 mm Internal Diameter).

1. **Volume per Revolution (V_{rev}):** Using the tube cross-sectional area (A) and the path circumference (C):
$$V_{rev} = A \times C \approx 0.492 mL/rev \tag{4}$$
Derived from $A = 3.14 \times 10^{-6} m^2$ and $C = 0.157m$[cite: 280, 283, 286].

2. **Time Calculation (t):** With a motor speed of 600 RPM ($10 rev/s$) and an estimated mechanical efficiency of 90%, the actual flow rate is calculated as:
$$Q_{actual} \approx 4.43 mL/s \tag{5}$$
Therefore, the required activation time to dispense 125 mL is:
$$t = \frac{125mL}{4.43mL/s} \approx 28 seconds \tag{6}$$

This duration falls well within the competition limits.

5.3 Manufacturing & Calibration

The pump components (Housing and Rotor) were manufactured using FDM 3D printing with PLA/PETG filaments for structural rigidity.

Calibration Strategy: Although the theoretical calculation yields ~ 28 seconds, we implemented a software calibration routine in the Arena Controller:

1. Run pump for fixed interval ($10s$).
2. Measure dispensed mass.
3. Adjust the `relay1` active duration in the code based on the experimental flow rate to ensure $\pm 5\%$ accuracy.

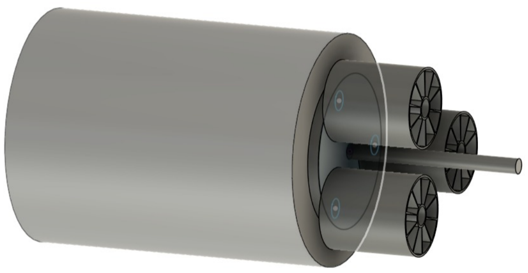
Figure 10: Side view of the pump

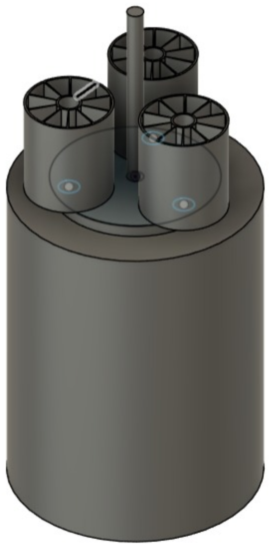
Figure 11: Isometric view of the pump