ALU-Based Autonomous Vehicle Simulator

Complete Project Summary + 5-Day Work Distribution

Project Overview

Title: Custom ALU for IoT Microcontroller - Autonomous Vehicle Testbed

What it is: A real-time simulation of a self-driving car that uses a custom decision-making ALU to navigate around obstacles, with quantitative performance analysis across different driving modes.

Core concept: We simulate an IoT microcontroller that reads 4 proximity sensors, processes them through custom decision logic (ALU + state machine), and outputs motion commands—all running in a 100ms control loop.

System Architecture

Layer 1: Frontend (What You See)

- Web dashboard in browser
- 2D Canvas showing:
  - Blue car with orientation
  - Red obstacles (static + moving)
  - Green sensor rays showing detection
- Real-time display:
  - 4 sensor readings (FL, FR, BL, BR)
  - Current ALU decision and state
  - Speed, collisions, distance traveled
  - Hazard score bar (0-100%)
- Controls:
  - Start / Pause / Reset buttons
  - Mode selector (Cautious / Normal / Aggressive)
  - Scenario selector (Corridor / Random / Intersection / Dense)
  - Noise ON/OFF toggle
  - Filter ON/OFF toggle

Layer 2: Backend (The Brain - Python)

Physics Engine: - Car kinematics (position, angle, velocity, acceleration, friction, turning) - Obstacle dynamics (movement, bouncing, collision detection) - 500×500 world with boundaries

Virtual Sensors: - 4 raycast-based proximity sensors - Configurable range (default 100 units), cone angle (60°) - Distance calculation to nearest obstacle in each direction

Sensor Noise + Filtering (WOW Feature #1): - Optional Gaussian noise injection: `reading = true_distance + N(0, noise_std)` - Moving average filter (keeps last N readings, outputs average) - Smooth, stable values for ALU

Custom ALU + State Machine: - **Inputs:** 4 filtered distances, current mode, current state - **Processing:** - Compute hazard score: normalized measure of proximity danger - Check against mode-specific thresholds (danger/warning) - State machine with 5 states: - CRUISE (normal forward motion) - AVOID_LEFT (obstacle on right, turning left) - AVOID_RIGHT (obstacle on left, turning right) - EMERGENCY_BRAKE (imminent collision) - REVERSING (blocked, backing up) - Hysteresis to prevent rapid state oscillation - **Outputs:** Motion command (FORWARD / TURN_LEFT / TURN_RIGHT / REVERSE / BRAKE)

Time-to-Collision (TTC) Prediction (WOW Feature #2): - Estimates: `TTC = front_distance / forward_speed` - If TTC < threshold (e.g., 1.0 sec), override to EMERGENCY_BRAKE - Predictive safety, not just reactive

Data Logging: - CSV per run: timestamp, position, speed, 4 sensor values (raw + filtered), decision, state, hazard, collisions - Stored in `logs/` directory

Metrics Computation: - Collisions count - Distance traveled - Average speed - Time to first collision - Average hazard score

Layer 3: Data & Analysis

Hazard vs Time Chart: - Line graph showing hazard_score over simulation time - Compare shapes between modes (Cautious has lower average hazard)

Mode Comparison: - Run same scenario in all 3 modes - Generate table/chart: - Collisions per mode - Average speed - Distance traveled - Safety vs efficiency tradeoff

---

How the ALU Works (Simple Explanation)

```
EVERY 100ms:

1. READ SENSORS
     Get 4 distances (FL, FR, BL, BR)
     Apply noise (if enabled)
     Apply moving average filter (if enabled)

2. COMPUTE HAZARD
     Closer obstacles = higher hazard
     Formula: hazard = average( (max_range - distance) / max_range )
```

```
     Result: 0.0 (safe) to 1.0 (very dangerous)

3. CHECK TIME-TO-COLLISION
     If moving forward and TTC < threshold → EMERGENCY_BRAKE

4. STATE MACHINE DECIDES
     Current state + hazard + side distances → next state
     States: CRUISE / AVOID_LEFT / AVOID_RIGHT / EMERGENCY_BRAKE / REVERSING

5. OUTPUT ACTION
     State → motion command
     Apply to car physics (update velocity, angle)

6. LOG DATA
     Save everything to CSV

7. UPDATE DISPLAY
     Send JSON to frontend via /api/state
```

The 3 Driving Modes

Mode	Danger Threshold	Warning Threshold	Behavior	Result
Cautious	30 units	40 units	Reacts very early, keeps large safety margin	0-1 collisions, slower speed (1.5-1.8 m/s)
Normal	15 units	25 units	Balanced approach	1-2 collisions, medium speed (2.0-2.2 m/s)
Aggressive	10 units	15 units	Goes close to obstacles, prioritizes speed	3-5 collisions, faster speed (2.5-2.8 m/s)

Key insight: Same ALU architecture, different parameters → different safety/speed profiles.

Performance Metrics & Analysis

What We Measure

- **Safety:** Collision count (lower = better)
- **Efficiency:** Distance traveled / time (higher = better)
- **Robustness:** Same ALU handles multiple scenarios
- **Predictability:** Hazard score patterns match expected behavior

Example Results Table

```
Scenario      Mode        Collisions   Avg Speed    Distance

Corridor      Cautious    0            1.5 m/s      180 m
Corridor      Normal      0            1.9 m/s      228 m
Corridor      Aggressive  2            2.5 m/s      300 m

Random        Cautious    0            1.8 m/s      216 m
Random        Normal      1            2.2 m/s      264 m
Random        Aggressive  5            2.8 m/s      336 m

Intersection  Cautious    0            1.6 m/s      192 m
Intersection  Normal      1            2.0 m/s      240 m
Intersection  Aggressive  4            2.6 m/s      312 m
```

Conclusion: Clear safety-speed tradeoff proven quantitatively.

Why This Is a Strong Final-Year Project

Complexity Justification

Applied Digital Logic Design: - ALU as combinational logic block (inputs → thresholds → outputs) - State machine as sequential logic (state register + next-state logic) - Configuration registers (thresholds per mode) - Clocked system (100ms = clock period) - Hysteresis and stability techniques

IoT Microcontroller Context: - Sense-Decide-Act loop (exactly what embedded controllers do) - Real-time constraints (must decide within 100ms) - Sensor input processing (ADC-like readings) - Actuator output (motor commands) - Can directly port to Arduino/STM32

Systems Engineering: - Multi-layer architecture (frontend, backend, data) - Clean module separation - REST API design - Real-time data flow - Performance optimization

Testing & Validation: - Quantitative metrics, not just "it looks cool" - Multiple scenarios × multiple modes = thorough testing - Statistical analysis (average, variance) - Reproducible results

---

5-DAY WORK DISTRIBUTION (4-Person Team)

Day 0: Setup (Evening, ~1 hour each)

Everyone: - Install Python 3.8+, pip, Git - Install dependencies: `pip install flask flask-cors numpy` - Create GitHub repo, clone locally - Create folder structure: `iot-alu-project/ car_simulator.py physics.py sensors.py alu.py api.py logger.py config.json index.html scenarios/ logs/`

Day 1: Foundation & Infrastructure

Person 1 (Physics Engineer) - 6 hours - [ ] Implement car physics class: - Position (x, y), angle, velocity, acceleration - `update()` method: apply friction, boundaries - `accelerate()`, `turn()`, `brake()` methods - [ ] Implement obstacle class: - Static obstacles (circles/rectangles) - Moving obstacles (simple patterns) - [ ] Basic collision detection (car vs obstacles, car vs walls) - [ ] Create `config.json` with all parameters - [ ] Test: car moves, obstacles exist, collisions detected

Person 2 (ALU Engineer) - 6 hours - [ ] Design state machine diagram (on paper/whiteboard) - [ ] Implement basic ALU class: - `compute_hazard(distances)` method - Simple state machine (start with 3 states: CRUISE, AVOID, BRAKE) - `decide(sensors, mode)` → returns action - [ ] Implement 3 mode configurations - [ ] Test with dummy sensor values (hardcoded)

Person 3 (Backend Engineer) - 6 hours - [ ] Set up Flask app structure - [ ] Implement basic endpoints: - `/api/state` (GET) - returns dummy data initially - `/api/reset` (POST) - `/api/set_mode/<mode>` (POST) - [ ] Create logger class: - CSV writer - Columns: timestamp, x, y, speed, sensors, decision, collisions - [ ] Test: Flask runs, endpoints respond

Person 4 (Frontend Engineer) - 6 hours - [ ] Create basic HTML structure: - Canvas element (600×600) - Control panel (buttons, dropdowns) - Stats display area - [ ] Implement canvas drawing: - Draw car as blue rectangle - Draw obstacles as red circles - Draw sensor rays as green lines - [ ] Implement polling: fetch `/api/state` every 100ms - [ ] Test: can draw static car and obstacles

End of Day 1 Goal: Everyone has their basic module working independently.

Day 2: Integration & Core Features

Person 1 (Physics Engineer) - 7 hours - [ ] Implement virtual sensors: - Ray-cast algorithm (line-circle intersection) - 4 sensors at angles: -45°, +45°, -135°, +135° relative to car - Return distance to nearest obstacle (up to max_range) - [ ] Add sensor noise option: - Gaussian noise: `N(0, noise_std)` - [ ] Implement moving average filter: - Keep last 5 readings per sensor - Return average - [ ] Create 2 basic scenarios (JSON): - Random: obstacles at random positions - Corridor: narrow passage with walls - [ ] Test: sensors return correct distances

Person 2 (ALU Engineer) - 7 hours - [ ] Complete 5-state machine: - CRUISE, AVOID_LEFT, AVOID_RIGHT, EMERGENCY_BRAKE, REVERSING - [ ] Implement hysteresis: - State must be stable for 3 frames before transition - [ ] Add TTC prediction: `- ttc = front_distance / speed if speed > 0` - If `ttc < 1.0`, override to EMERGENCY_BRAKE - [ ] Test state transitions with various sensor patterns - [ ] Integrate with Person 1's sensors

Person 3 (Backend Engineer) - 7 hours - [ ] Integrate physics + sensors + ALU into main simulator loop - [ ] Implement `/api/scenario/<name>` endpoint - [ ] Implement `/api/metrics` endpoint: - Compute collisions, distance, avg_speed, avg_hazard - [ ] Add logging: - Create new CSV per run - Log every 100ms step - [ ] Test full backend: car moves, ALU decides, data logged

Person 4 (Frontend Engineer) - 7 hours - [ ] Implement real-time updates: - Parse JSON from `/api/state` - Update canvas every frame - Update stats display - [ ] Add control buttons: - Start, Pause, Reset (call backend APIs) - Mode selector dropdown - Scenario selector dropdown - [ ] Add sensor value display (4 numbers) - [ ] Add current state and decision display - [ ] Test: full interaction loop working

End of Day 2 Goal: Fully integrated system. Car moves, avoids obstacles, you can control it.

Day 3: WOW Features + Scenarios

Person 1 (Physics Engineer) - 6 hours - [ ] Create 2 more scenarios: - Intersection: crossroad with moving obstacles - Dense traffic: many obstacles - [ ] Fine-tune physics parameters (acceleration, friction, turn_speed) - [ ] Add dynamic obstacles: - Some obstacles move back-and-forth - Some bounce off walls - [ ] Test all 4 scenarios thoroughly

Person 2 (ALU Engineer) - 6 hours - [ ] Optimize state machine: - Fix any oscillation bugs - Fine-tune thresholds for all 3 modes - [ ] Add edge case handling: - All sensors showing max range (no obstacles) - All sensors showing

0 (completely stuck) - Single sensor failing (stuck value) - [ ] Document state transition diagram - [ ] Write unit tests for ALU logic

Person 3 (Backend Engineer) - 6 hours - [ ] Implement `/api/logs` endpoint: - Return list of available CSV files - [ ] Add CSV download functionality - [ ] Run full test matrix: - 4 scenarios × 3 modes = 12 runs - 2 minutes per run - Save all results - [ ] Compute comparison metrics: - Aggregate data across modes - Create comparison JSON

Person 4 (Frontend Engineer) - 6 hours - [ ] Add noise/filter toggles: - UI switches for ON/OFF - Show raw vs filtered sensor values side-by-side - [ ] Add hazard score bar: - Visual bar (0-100%) - Color: green → yellow → red - [ ] Implement basic hazard chart: - Plot hazard_score vs time after run - Use simple canvas line chart or Chart.js - [ ] Add metrics panel: - Display results after run ends - [ ] Polish UI styling

End of Day 3 Goal: All features implemented. System looks professional.

Day 4: Testing, Metrics & Charts

Person 1 (Physics Engineer) - 5 hours - [ ] Run extensive testing: - Test each scenario 3 times per mode - Note any bugs or unrealistic behavior - [ ] Adjust physics if needed - [ ] Create scenario documentation: - Description of each scenario - Expected behavior - [ ] Help with integration bugs

Person 2 (ALU Engineer) - 5 hours - [ ] Final ALU optimization: - Ensure modes show clear behavioral differences - Verify TTC-based braking works - [ ] Create ALU logic flowchart: - State diagram - Decision tree - [ ] Write technical documentation: - How ALU works - Threshold meanings - State transitions - [ ] Prepare viva answers

Person 3 (Backend Engineer) - 5 hours - [ ] Implement mode comparison feature: - Backend API: `/api/compare_modes/<scenario>` - Runs same scenario in all 3 modes - Returns comparison data - [ ] Generate final results table: - CSV with all 12 configurations - Summary statistics - [ ] Write API documentation: - All endpoints - Request/response formats - [ ] Performance testing: - Ensure 100ms loop is stable

Person 4 (Frontend Engineer) - 5 hours - [ ] Implement mode comparison UI: - "Compare Modes" button - Shows 3-column comparison table - Bar chart: collisions per mode - [ ] Polish hazard chart: - Add axis labels - Add legend - Multiple runs on same chart (optional) - [ ] Add download button for CSV logs - [ ] Final UI polish: - Consistent styling - Responsive layout - Loading indicators

End of Day 4 Goal: Complete, tested, polished system. Ready for demo.

Day 5: Documentation, Demo Prep & Presentation

Person 1 (Physics Engineer) - 4 hours - [ ] Write README.md: - Installation instructions - How to run - Project structure - [ ] Create quick-start guide - [ ] Record demo video (optional): - 2-minute showcase - [ ] Prepare your demo talking points: - "I built the physics engine and sensors" - Explain raycast, collision detection

Person 2 (ALU Engineer) - 4 hours - [ ] Write ARCHITECTURE.md: - ALU design explanation - State machine documentation - Digital logic mapping - [ ] Create state diagram image/flowchart - [ ] Prepare demo script: - "Our ALU uses a 5-state FSM…" - Explain modes and thresholds - [ ] Prepare viva Q&A: - "What was hardest?" → hysteresis - "How does it map to hardware?" → FSM

Person 3 (Backend Engineer) - 4 hours - [ ] Write API.md: - All endpoints documented - Example requests/responses - [ ] Create testing report: - Results table (4 scenarios × 3 modes) - Analysis and conclusions - [ ] Prepare demo script: - "Backend handles physics, sensors, ALU, logging" - Show metrics endpoint live - [ ] Backup all code and data: - Zip project folder - Upload to Google Drive / GitHub

Person 4 (Frontend Engineer) - 4 hours - [ ] Create UI documentation: - Screenshots of all features - User guide (how to use dashboard) - [ ] Final testing on different browsers - [ ] Prepare demo script: - "Frontend shows real-time visualization" - Demo all features live - [ ] Create presentation slides (10-15 slides): - Title, problem, solution, architecture - Demo screenshots - Results table - Conclusion

Team Meeting (2 hours together): - [ ] Full rehearsal of demo (5 minutes) - [ ] Run through viva Q&A - [ ] Assign presentation roles: - Intro (Person 1) - Architecture (Person 2) - Demo (Person 3 drives, Person 4 narrates) - Results (Person 1) - Q&A (all) - [ ] Final polish and fixes

End of Day 5 Goal: Presentation-ready. Demo rehearsed. Documentation complete.

Daily Sync Schedule

Every day at 9 PM IST (30 minutes): 1. Each person reports: - What I completed today - What I'm working on tomorrow - Any blockers or questions 2. Quick integration check: - Does Person 3 have latest code from everyone? - Any merge conflicts? 3. Adjust plan if needed

Use: Slack/Discord for quick questions throughout the day

Success Criteria

By end of Day 5, you should have:

Working simulator: - Car navigates around obstacles - 3 modes show different behavior - Multiple scenarios work

WOW features: - Sensor noise + filtering (toggleable) - TTC-based predictive braking - Hazard vs time chart - Mode comparison dashboard

Data & metrics: - 12+ CSV logs (4 scenarios × 3 modes) - Results table with analysis - Performance comparison

Complete documentation: - README, ARCHITECTURE, API docs - State diagrams, flowcharts - Demo script prepared

Presentation: - 5-minute live demo - Viva Q&A prep - All team members confident

---

5-Minute Demo Script

0:00-0:45 - Introduction (Person 1) > "We built a simulator for testing custom ALU-based decision logic for autonomous vehicles. Our system has a 2D car with 4 virtual proximity sensors that feed into a custom ALU, which decides motion commands every 100ms."

Show architecture diagram.

0:45-1:30 - ALU Explanation (Person 2) > "The ALU implements a 5-state finite state machine with hazard-based decision making. We have 3 modes—Cautious, Normal, Aggressive—that use different safety thresholds."

Show state diagram briefly.

1:30-3:30 - Live Demo (Person 3 operates, Person 4 narrates)

Start simulation in Normal mode: > "Here's the car (blue), obstacles (red), and sensor rays (green). Watch the sensor values update in real-time."

Let it run for 20 seconds, point out: - ALU decision changing (CRUISE → AVOID_LEFT → CRUISE) - Hazard bar fluctuating - Car successfully avoiding obstacles

Switch to Aggressive mode: > "Now we switch to Aggressive mode—notice it goes much closer to obstacles, higher speed, but also more collisions."

Let run 15 seconds.

Toggle noise + filter: > "We can add sensor noise and show how filtering stabilizes the readings."

Show raw vs filtered values briefly.

3:30-4:30 - Results (Person 1)

Show results table on screen: > "We tested 4 scenarios across 3 modes. You can see Cautious has zero collisions but slower speed, while Aggressive is fastest but has 3-5 collisions per run."

Show hazard chart: > "This chart shows hazard score over time—Cautious stays in the safe zone, Aggressive frequently spikes into danger."

4:30-5:00 - Conclusion (Person 2) > "This demonstrates a complete ALU-based IoT control system: realistic sensors, custom decision logic, real-time operation, and quantitative validation. The logic can directly port to hardware like Arduino or STM32 for a real robot car."

Viva Q&A Prep (Key Points)

Q: How does this relate to digital logic design? A: "Our ALU is a combinational logic block: inputs (sensor values) → comparisons → state machine logic → output (action). The state machine is sequential logic with a state register. The 100ms loop is our clock period."

Q: Why different modes? A: "To prove the same ALU architecture can exhibit different behaviors via configuration, like how FPGAs/microcontrollers use programmable thresholds."

Q: What was the hardest part? A: "Preventing state oscillation. The car would rapidly flip between AVOID_LEFT and AVOID_RIGHT. We added hysteresis—a state must hold for multiple cycles before transitioning."

Q: How is this IoT-related? A: "This is exactly the sense-decide-act loop an IoT microcontroller executes: read ADC from sensors, run decision logic, output PWM to motors. We're simulating it safely before deploying to hardware."

Q: Can you add more features? A: "Yes: machine learning to optimize thresholds, multi-car coordination, camera-based prediction, or actual hardware integration with ROS/Gazebo."

Final Checklist

Code: - [ ] All modules implemented and integrated - [ ] No crashes or major bugs - [ ] Config file well-documented - [ ] Code commented

Features: - [ ] 3 modes working with clear differences - [ ] 4 scenarios available - [ ] Noise + filter working - [ ] TTC-based braking working - [ ] Charts and metrics displaying

Documentation: - [ ] README.md complete - [ ] ARCHITECTURE.md written - [ ] API.md documented - [ ] State diagram created

Testing: - [ ] 12+ runs completed - [ ] Results table generated - [ ] No obvious bugs remaining

Demo: - [ ] Rehearsed at least twice - [ ] 5-minute timing confirmed - [ ] All team members know their parts - [ ] Backup plan if live demo fails (video)

Presentation: - [ ] Slides created (10-15 slides) - [ ] Viva Q&A reviewed - [ ] Each person can explain their contribution

---

Motivation

This is totally doable in 5 days if: - Everyone commits ~5-7 hours per day - You do daily syncs - You help each other with blockers - You focus on functionality first, polish second

By Day 5, you'll have: - A working, impressive demo - Real data proving your design works - Clear documentation - Confidence to present and answer questions

You got this!

Document Version: 2.0
Date: January 30, 2026
Status: Ready for 5-day sprint
Team Size: 4 people
Estimated Total Hours: ~120 hours (30 hours/person)