STM JOURNALS

Journal of
Mechatronics and Automation

ISSN: 2455-1988
Volume 13, Issue 2, 2026
May–August
DOI (Journal): 10.37591/JoMA

https://journals.stmjournals.com/joma

Research JoMA

A Reproducible Low-cost Adaptive Cruise Control Platform with Lidar–Ultrasonic Sensor Fusion and Timer-conflict-free Servo-Scanned Blind-spot Detection on Arduino UNO

Bibhuti Bhusan Nayak[1,*], Md Sahariar Hossain[2], Tamanna Gautam[2]

Abstract

Road traffic deaths caused by driver distraction and inadequate surroundings remain a big problem for preventable deaths. This paper talks about designing and testing an affordable, easy-to-copy Adaptive Cruise Control (ACC) system. It uses LiDAR-ultrasonic sensors to figure out how far things are in front of the car, and a servo-actuated, dynamically scanned blind spot detector. All of this runs on a single Arduino Uno.

To measure front distance, they use both a Benewake TF-Mini LiDAR and an HC-SR04 ultrasonic sensor. The team combines the data from these using the minimum-selection rule and filter out implausible readings. This info is then mapped to motor drive using a simple control law where speed adjusts based on distance.

A cool part of the project was figuring out how to work around the Timer 1 issue with libraries for controlling motors and servos. By creating a custom routine for the servos, they managed to run all four-wheel motors and back scanners at the same time. Their tests show a speed command error of just 2.4 PWM units, which is only about 1% off. Also, it detects blind spots perfectly, without any false alarms, and works great on different surfaces too. Best of all, they did it for under $50 extra.

Keywords: Adaptive cruise control, sensor fusion, LiDAR, ultrasonic ranging, blind-spot detection, bit-banged PWM, Arduino, embedded systems, ADAS.

INTRODUCTION

Adaptive Cruise Control (ACC) keeps your car at a set speed while auto-adjusting to stay safe behind other vehicles. This makes it one of the more advanced helper-tech features we have now. Yet, ACC systems aren't cheap and are locked into car brands, which makes them tough to look into for studies or cheap mock-ups. Road accidents happen often due to sleepy or distracted drivers. In fact, around 1.19 million people die each year in crashes, according to the World Health Organization. Thus, having real-time awareness with automatic speed control can help not just make rides comfy, but keep folks safer too. This research tackles three main issues: first, making an open and doable ACC model for classes and study; second, giving back support to detect spots behind you that normal sensors miss; and third, doing all this on affordable, easy-to-find parts [1-5].

The key focus here is showing how front-object sensing, merging info from different sensors, fixing

***Author for Correspondence**
Bibhuti Bhusan Nayak
E-mail: bibhuti@nitsikkim.ac.in

[1]Assistant Professor, Department of Mechanical Engineering, National Institute of Technology Sikkim, Ravangla, South Sikkim, India
[2]Student, Department of Mechanical Engineering, National Institute of Technology Sikkim, Ravangla, South Sikkim, India

Received Date: July 17, 2026
Accepted Date: August 11, 2026
Published Date: August 27, 2026

Citation: Bibhuti Bhusan Nayak, Md Sahariar Hossain, Tamanna Gautam. A Reproducible Low-cost Adaptive Cruise Control Platform with Lidar–Ultrasonic Sensor Fusion and Timer-conflict-free Servo-Scanned Blind-spot Detection on Arduino UNO. Journal of Mechatronics and Automation. 2026; 13(2): 58–70p.

speeds as needed, and carefully checking blind spots can all work on just an Arduino Uno. In fact, they explain how a LiDAR-ultrasonic mix helps pick out actual obstacles better and avoid fake readings. Plus, they solved a tech issue letting a motor and servo run together on one tiny computer board. There's even a part-mover that scans all angles in your blind spot, not just one fixed angle.

Also important, when creating tech for super crucial safety stuff, how it's built matters as much as what math you use. They chose ways to set timings, like quick non-stop check-ins, safe data handling, and smooth display updates, all shown in full detail so others can copy exactly [6-9].

The authors offer a practical way to cut down big crashes using affordable parts, explained well enough for others to copy it. The paper is structured like this: Section II covers ACC controls, sensor blending, and blind-spot detection. Section III dives into their two-node setup and control levels. In Section IV, they talk about the hardware and timing issues that cause main problems. Section V includes all the firmware details for controls and actions, listed out completely. Then, Section VI shows experiment results, Section VII talks about limits of their work, and finally, Section VIII wraps things up [10,11].

RELATED WORK

Adaptive Cruise Control

Early cruise control just kept your car at a steady speed, no matter what was going on around you. But to make it adaptive, engineers added a sensor that looks ahead and a control system that switches between modes. In one mode, it keeps the car at a speed set by the driver. In another, it makes sure the car is a safe distance from the one in front.

Shladover's work [1] laid the groundwork for this in intelligent transportation systems. His headway-time based control is still used today [12]. According to the SAE's driving automation levels [13], Adaptive Cruise Control only partially automates driving. It still falls on the driver to handle the driving task, but the system takes care of the car's speed and distance from other vehicles.

From the late '90s on, commercial adaptive cruise control (ACC) systems mainly used millimeter-wave radar for forward sensing. This technology handles rain, fog, and low-light well. Researchers later looked into cheaper sensor options. Moon et al. showed that LiDAR could offer precise full-range ACC and collision avoidance, all tested across 5 to 150 meters. In our smaller setup, which ranges under 150 cm, a budget-friendly time-of-flight sensor, like the TF-Mini, works just as well for relative accuracy.

Most ACCs use one of three control methods: PID control, model predictive control (MPC), and fuzzy logic. MPC is good for dealing with constraints and offering preview features, but it needs a lot of processing power. Fuzzy logic relies on expert guesses yet requires tweaking. The prototype we're dealing with is for low speeds and fairly simple dynamics. In situations like this, basic proportional control connects distance to speed directly. Studies back this up [3].

Since our system operates slowly – where things like actuator lag and tire dynamics don't affect much – the benefits of integral or derivative terms aren’t needed. As a result, a simple proportional map linking detected distance to motor duty cycle does the job nicely and is easy to manage within the microcontroller’s control loop *B.*

LiDAR–Ultrasonic Sensor Fusion

Ultrasonic HC-SR04 sensors figure out distance using the time-of-flight of an acoustic pulse. These sensors are affordable and do a great job for flat, straight-on surfaces. Still, they don't handle angled, rough, or absorbent surfaces well because the returning echo gets redirected.

Time-of-flight LiDAR works differently—it sends out a collimated optical pulse. This method isn’t as bothered by the shape of the target, offering better angular detail and a further useful range. Yet, it

faces issues of its own on reflective and super dark surfaces, like glass, because it picks up too little light. The neat thing is that these systems usually mess up in different situations, making their combo pretty powerful. Engineers have come up with various ways to merge their data. For instance, there's the minimum-selection rule, which opts for the closer of the two distances detected. If either finds something nearby, they trigger a caution right away. This approach is easy on processing power and doesn't need any adjustments, making it ideal for controllers with fewer capabilities. There are also more complex fusions using filters like the Kalman, which aim to perfect stats and manage movement models smoothly. But they require tweaking and managing extra info, complicating things. In this case, researchers improved the basic approach by removing unlikely readings beforehand—anything less than 2 cm or more than 400 cm for the sonar, and beyond 1200 cm for LiDAR gets ditched. Also, there's a backup to keep everything moving if both sensors goof up simultaneously, stopping any unexpected halts.

Blind-Spot Detection

Commercial blind-spot monitoring typically uses 24 GHz or 77 GHz short-range radar mounted in the rear bumper, and is governed by lane-change decision-aid performance standards. In academic and prototype work, fixed-angle ultrasonic sensors are a common low-cost substitute, but a fixed mount provides only a point measurement at a single predetermined direction; a vehicle executing a lane change requires continuous angular coverage that a single static sensor cannot supply, leaving blind regions between sensor axes. A servo-actuated sensor that sweeps the rear sector addresses this directly by trading instantaneous coverage for time-multiplexed angular coverage. Alonso et al. [5] demonstrated a rotating ultrasonic assembly for indoor robot obstacle avoidance with sub-100 ms sweep cycles at 5° resolution, establishing the feasibility of the approach. Elfes [4] earlier formalized the use of sonar range data for occupancy mapping, underpinning the interpretation of swept range measurements. The scanner in the present work sweeps a 60°–120° arc sized for a scaled prototype.

SYSTEM ARCHITECTURE

The integrated platform is partitioned into two computational nodes connected by a UART serial link, as shown in Fig. 1. The Arduino Uno is the real-time control node, responsible for sensor polling, sensor fusion, proportional speed regulation, servo actuation, and LCD display management. A Raspberry Pi 4 acts as a perception node; for the purposes of this paper it is treated as an external source of a single-byte driver-status command, which the control node honors at the highest priority. At each control cycle the Arduino reads the LiDAR and both ultrasonic sensors, applies the fusion rule, polls the serial buffer for a driver-status byte, and resolves a motor command by the priority hierarchy: (1) if the driver-status byte indicates a stop condition, the vehicle halts unconditionally; (2) if the fused front distance is within the safe-stop threshold, the vehicle halts; (3) if the fused distance lies between the stop threshold and the maximum detection range, speed is mapped proportionally to distance; (4) otherwise the vehicle cruises at maximum speed. Concurrently, the servo sweep runs on a non-blocking timer and the LCD refreshes at 300 ms intervals.

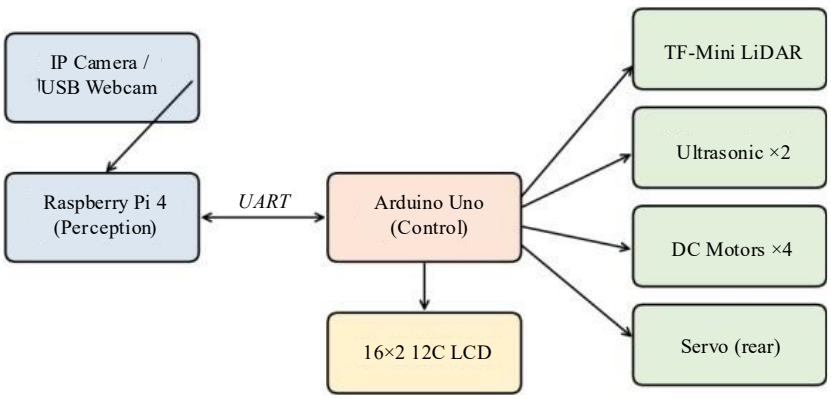

Figure 1. Two-node architecture and peripheral connections.

HARDWARE DESCRIPTION

Table 1 summarizes the principal components. The platform is a four-wheel-drive plywood chassis (25×20 cm) driven by four TT gear motors through an Adafruit AFMotor (L293D) shield, powered from a 7.4 V Li-Po pack with regulated 5 V rails.

Table 1. Principal Components

Component	Qty	Key specification
Arduino Uno R3	1	ATmega328P, 16 MHz
Raspberry Pi 4 (1 GB)	1	Cortex-A72, Wi-Fi
Benewake TF-Mini LiDAR	1	0.3–12 m, ±1%, 100 Hz
HC-SR04 ultrasonic	2	2–400 cm, 5 V
AFMotor shield (L293D)	1	4-ch DC, PWM Tmr1/2
TT gear motor 1:48	4	3–6 V, 120–200 RPM
SG90 micro servo	1	0–180°, 1.8 kg·cm
16×2 I2C LCD	1	PCF8574, addr 0x27

Front Ranging Sensors

The TF-Mini transmits a 9-byte UART frame at 115200 baud with a fixed 0x59 0x59 synchronization header, two little-endian distance bytes (cm), two signal-strength bytes, and a checksum, at a 100 Hz update rate that comfortably exceeds the 10–20 Hz control-loop rate. Two HC-SR04 modules are deployed: one forward, alongside the LiDAR, for redundant front detection, and one rear-mounted on the servo for blind-spot scanning. Both share a single trigger pin and use separate echo pins; distance is obtained from the echo pulse duration t (μs) and the speed of sound as

$$d = t \times 0.034 / 2, \tag{1}$$

where the factor of one half accounts for the round-trip path. Echo durations corresponding to ranges below 2 cm or above 400 cm are rejected as invalid.

Motor Drive and the Timer Allocation

The AFMotor shield uses two L293D H-bridges. PWM for motor channels 1–2 is generated on Timer 1 and for channels 3–4 on Timer 2. The ATmega328P provides only three hardware timers: Timer 0 serves the Arduino runtime, leaving Timer 1 (16-bit) and Timer 2 (8-bit) for libraries. The standard Servo.h library also claims the 16-bit Timer 1 for its 50 Hz refresh, creating a direct resource conflict whose resolution is described in Section V-D.

Pin Assignment

Table 2 documents the complete pin mapping so the platform can be rebuilt without ambiguity. The two ultrasonic sensors share a single trigger line and use independent echo pins; the LiDAR uses a SoftwareSerial pair on the analog header; and the servo signal occupies the digital pin freed by removing Servo.h.

Table 2. Arduino Pin Assignment

Pin	Connection	Function
A0 / A1	TF-Mini TX / RX	LiDAR SoftwareSerial
A2	HC-SR04 TRIG (both)	Shared trigger pulse
A3	HC-SR04 echo (front)	Front echo return
2	HC-SR04 echo (rear)	Rear echo return
10	SG90 servo signal	Bit-banged PWM
A4 / A5	LCD SDA / SCL	I2C bus
3,5,6,9,11	AFMotor shield	Motor channels 1–4
USB	Raspberry Pi	UART command link

CONTROL AND ACTUATION SOFTWARE

Non-Blocking Firmware Architecture

The firmware executes all tasks within a single non-blocking main loop using millis() and micros() timers; the delay() function is never used in the loop. This is essential because any blocking call would prevent the serial port from being polled for an incoming stop command and would suspend sensor polling, potentially missing a rapidly closing obstacle. LiDAR parsing, ultrasonic polling, serial reading, fusion, and speed control execute every iteration; servo pulses fire every 20 ms, the servo position advances every 40 ms, and the LCD updates every 300 ms, each gated by an elapsed-time comparison. Listing 1 shows the structure of the loop, in which every branch is non-blocking and the most time-critical task—reading the driver-status command and the front sensors—executes first.

```
  int lD = readLidar();    // -1 if invalid int fD =
  getDistance(ECHO_F);
  int rD = getDistance(ECHO_R); // blind spot int fused;
  if (lD>0 && fD>0) fused = min(lD,fD); else if (lD>0)
                 fused = lD;
  else if (fD>0) fused = fD;
  else           fused = MAX_DETECT_DIST+1; int spd = 0;
  if (driverStatus=='D' ||
      (fused>0 && fused<=SAFE_STOP_DIST)) { stopCar(); spd = 0;
  } else {
    spd = (fused<MAX_DETECT_DIST) ?
      map(fused,SAFE_STOP_DIST,MAX_DETECT_DIST,
          MIN_SPEED,MAX_SPEED) : MAX_SPEED;
    moveForward(constrain(spd,0,MAX_SPEED));
  }
  handleServo();           // bit-banged PWM if (millis()-lastLCD >=
  300) {
    lastLCD = millis(); updateLCD(fused,rD,spd,driverStatus);
  }
}
```

Listing 1. Non-blocking main control loop (priority: drowsy stop → obstacle stop → proportional → cruise).

The consequence of any blocking call in this loop is concrete and severe. A single delay(100) would prevent the serial port from being checked for 100 ms—a window in which the perception node might transmit a stop command that goes unacted—and, more critically, would suspend sensor polling, potentially missing a rapidly closing obstacle over the same interval. The non-blocking design guarantees that each task is serviced at its required rate independent of the others: the only blocking primitives that remain are the few microseconds of the ultrasonic trigger and the bounded 25 ms pulseIn() timeout, both of which are accounted for in the worst-case loop bound.

```
void readPiCommand(){
  if (Serial.available() > 0){
    String cmd =
    Serial.readStringUntil('\n');
    cmd.trim();
    if (cmd.length() >= 11){ //
      'D:W:060:150' driverStatus =
      cmd.charAt(0);         // N or D
      weatherMode  = cmd.charAt(2);    //
      N/W/I/F int g =
      cmd.substring(4,7).toInt();
      int s = cmd.substring(8,11).toInt();
      if (g >= 20) SAFE_STOP_DIST = g;
      // reject if (s >= 50) MAX_SPEED =
      s; // reject
    }
  }
}
```

Listing 1b. Fixed-position command parser with field validation.

The structured command received from the perception node is parsed by the routine in Listing 1b, which reads a complete newline-terminated line only when one is available and validates each field by fixed character position. Implausible values—a safety gap below 20 cm or a maximum speed below the stall floor—are silently rejected, so a corrupted packet preserves the last known-good parameters rather than commanding an unsafe state.

Sensor Fusion with Validity Fallback

The fusion rule selects the minimum of the two valid readings, falls back to whichever single sensor remains valid, and defaults to a beyond-range value only if both fail—so the vehicle maintains motion rather than stopping spuriously when both sensors momentarily time out:

Equivalently, writing d_L and d_U for the validity-filtered LiDAR and ultrasonic readings, the fused distance is

$$d\_f = min(d\_L, d\_U) \text{ for valid } d\_L, d\_U, \tag{2}$$

with single-sensor fallback when only one reading is valid, and a beyond-range sentinel when both fail.

Proportional Speed Control

Outside the stop zone, speed is a linear map from fused distance to PWM between an empirically determined minimum (85 PWM, the lowest duty cycle that reliably overcomes static friction on the loaded chassis) and the maximum cruise PWM:

$$u = u\_min + (d\_f - d\_stop)(u\_max - u\_min)/(d\_max - d\_stop), \tag{3}$$

clamped to [0, u_max]. The resulting state machine is summarized in Table 3. The proportional law yields a smooth, continuous deceleration profile rather than the abrupt step response of a threshold-only controller.

Table 3. ACC State-Machine Zones

Distance range	PWM	Behaviour
0–stop dist.	0	Emergency stop
stop–50 cm	85–103	Creep at min speed
50–80 cm	103–140	Reduced speed
80–150 cm	140–200	Speed ∝ distance
> 150 cm	200	Cruise at max

Timer-Conflict Resolution: Bit-Banged Servo PWM

When both AFMotor and Servo.h are present, both initialize Timer 1 with conflicting prescaler and compare-match settings. The servo then receives an incorrect pulse width and remains stationary or twitches, while the Timer 1 motor channels become unreliable. The adopted solution removes the Servo.h dependency entirely and replaces it with a manually bit-banged routine: a micros()-gated timer fires a HIGH pulse every 20 ms (50 Hz), holding it for a width linearly interpolated between the SG90 minimum (544 µs) and maximum (2400 µs) for 0–180°:

$$t\_pw = 544 + (\theta / 180)(2400 - 544) \text{ [µs]} \tag{4}$$

The 2.4 ms worst-case pulse occupies only 12% of the 20 ms refresh window, leaving the remaining 17.6 ms free for sensing and serial communication, so the firmware retains its non-blocking character. This frees Timer 1 for exclusive motor-driver use and resolves the conflict. Listing 2 gives the routine: a single short blocking interval per pulse is acceptable precisely because it recurs only once per 20 ms window.


```
void sendServoPulse(int angle){
  int pw = map(angle,0,180,544,2400);
  // us digitalWrite(SERVO_PIN,HIGH);
  delayMicroseconds(pw);
  digitalWrite(SERVO_PIN,LOW);
}
void handleServo(){
  if (micros()-lastPulse >= 20000){
                                  // 50
    Hz lastPulse = micros();
    sendServoPulse(servoPos);
  }
  if (millis()-lastMove >= 40){// 2
    deg/40ms lastMove = millis();
    servoPos += servoDir;
    if (servoPos<=60 || servoPos>=120)
      servoDir = -servoDir;      // reverse
  }
}
if (lD>0 && fD>0) fused = min(lD,fD);
else if (lD>0) fused = lD;
else if (fD>0) fused = fD;
else fused = MAX_DETECT_DIST + 1;
void loop() {              // non-
  readPiCommand            blocking
```

Listing 2. Bit-banged servo PWM replacing Servo.h.

Servo-Scanned Blind-Spot Detection

The rear HC-SR04 is mounted on the SG90 servo, which sweeps a 60°–120° arc in 2° steps at 40 ms intervals, giving a full sweep in roughly 1.2 s (50°/s). A three-tier proximity policy maps the measured rear distance to alert levels reported on the I2C LCD and an audible buzzer: SAFE (>60 cm), WARNING (30–60 cm), and DANGER (<30 cm).

Robust Parsing and Display Management

Two further mechanisms preserve real-time behavior. The LiDAR parser (Listing 3) scans the SoftwareSerial buffer for the 0x59 0x59 header before extracting the little-endian distance field, and aborts after a bounded number of attempts so that a corrupted or partially received frame degrades gracefully to a single-sensor cycle rather than blocking the loop. The bounded-attempt guard is important because a SoftwareSerial buffer overflow during a long task can otherwise leave the parser scanning indefinitely.

```
int readLidar(){ int
  attempts = 0;
  while (tfSerial.available() >= 9){
    if (tfSerial.read()==0x59 &&
       tfSerial.peek()==0x59){
      tfSerial.read();        // 2nd header
      uint8_t d[7];
      for (int i=0;i<7;i++)
      d[i]=tfSerial.read(); int dist = d[0] +
      d[1]*256; // little-endian if (dist>0
      && dist<=1200) return dist;
    }
    if (++attempts > 20) break;  // anti-
    overflow
  }
  return -1;               // no valid frame
}
```

Listing 3. Header-synchronized TF-Mini frame parser.

The ultrasonic routine (Listing 4) issues a 10 µs trigger and times the echo with a 25 ms pulseIn() timeout, corresponding to a maximum measurable range near 4.25 m; an out-of-range reading returns the invalid sentinel so the fusion stage can fall back cleanly.

```
long getDistance(int echoPin){
  digitalWrite(TRIG,LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG,HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG,LOW);
  long dur = pulseIn(echoPin,HIGH,25000);
  // 25 ms if (dur==0) return -1;
  long cm = dur * 0.034 / 2;
  return (cm<2 || cm>400) ? -1 : cm;
}
```

Listing 4. Ultrasonic ranging with timeout and validity bounds.

The 16×2 I2C LCD, whose full two-line refresh costs 5–15 ms over the 100 kHz bus, is managed by a double-buffering scheme: each line is rebuilt into a local array and written only when its content changes, and updates are gated to a 300 ms interval. This eliminates visible flicker and bounds I2C bus utilization without consuming loop time.

EXPERIMENTAL RESULTS

Distance-to-Speed Mapping

The proportional law was characterized by placing a flat target at fixed distances and recording the issued PWM over ten trials per distance (Table 4). The mean absolute error across all distances was 2.4 PWM units, about 1.0% of the full 0–255 range. The stop boundary (≤30 cm → 0) and cruise boundary (≥150 cm → 200) were reproduced exactly; the largest deviations occurred at intermediate distances where reflection geometry and surface reflectivity perturb the readings (Figure 2).

Table 4. Speed Command vs. Distance

Dist (cm)	Exp. PWM	Mean PWM	Err (%)
20	0	0	0.0
30	85	85	0.0
50	111	109	1.8
80	148	146	1.4
100	168	165	1.8
120	188	185	1.6

Sensor-Fusion Reliability

Each sensor was tested against four surfaces at a fixed 80 cm range (100 readings per condition); the miss rate is the fraction of readings outside ±10 cm of truth (Table 5). The fused miss rate equals the lower of the two individual rates in every case, confirming that minimum-selection exploits the complementary failure modes—ultrasonic dominates on specular glass while LiDAR dominates on low-reflectivity foam (figure 3).

favors one modality, and the proportional law produces the smooth speed decay rather than a stepwise drop.

Blind-Spot Detection

The servo completed consistent 60°–120° sweeps during a 10-min ute continuous test, with reliable position control via the bit-banged routine. Measured rear distances over a representative scan are listed

in Table 6 and plotted in Fig. 5, illustrating the three-tier alert mapping. All three tiers behaved correctly: SAFE produced no alert, WARNING a single beep, and DANGER a continuous 2 kHz tone. Zero false-positive alerts were observed across five separate 10-minute runs, indicating that the threshold logic is well calibrated for the test environment.

Table 5. Miss Rate by Surface (%)

Surface	LiDAR	Ultra.	Fused
White cardboard	0	2	0
Black foam	3	14	3
Human hand	1	5	1
Glass panel	8	12	8

Table 6. Blind-Spot Scan and Alert Status

Angle	Sector	Dist (cm)	Alert
0°	Left extreme	70	SAFE
30°	Left side	70	SAFE
60°	Left rear	55	WARN
90°	Rear centre	40	WARN
120°	Right rear	60	WARN
150°	Right side	30	DANGER
180°	Right extreme	75	SAFE

Figure 4 shows a representative combined trace in which the fused front distance falls as an obstacle approaches and the motor command decreases accordingly, illustrating the closed coupling between perception and actuation. The LiDAR and ultrasonic traces diverge where surface geometry

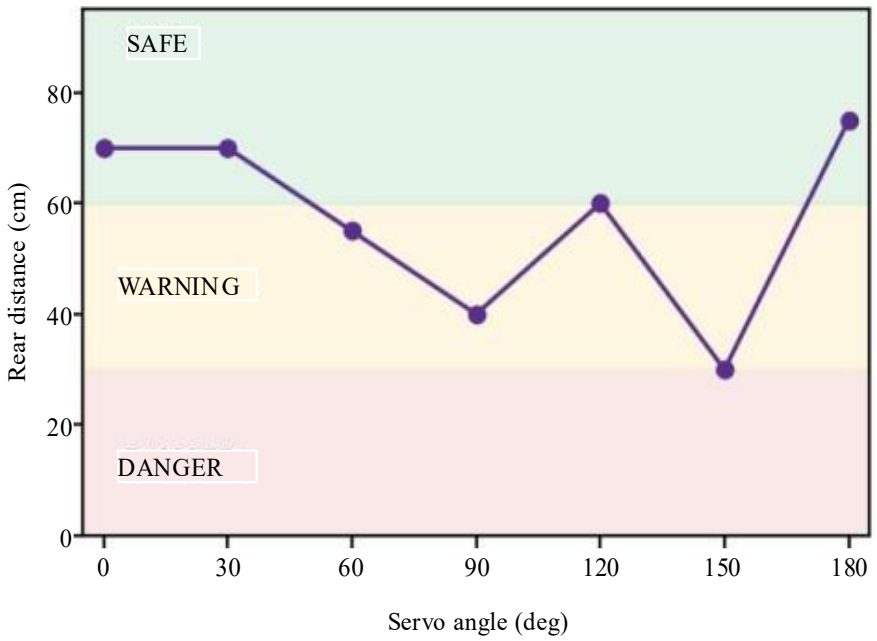

Figure 5. Rear distance vs. servo angle with the three alert bands.

Integrated Operation

Table 7 traces a representative integration run, recording the fused distance, the active mode, and the resulting vehicle action as an obstacle is introduced, approached, and removed. The transitions confirm

end-to-end coupling: the proportional zone produces a graded slow-down, the stop threshold halts the vehicle, and clearing the obstacle restores cruise, all within the per-iteration latency budget.

Table 7. Representative Integrated-Operation Timeline

t (s)	Event	Vehicle action
0	Clear road (>150 cm)	Cruise, PWM 200
3	Obstacle at 80 cm	Slow to PWM 146
5	Obstacle at 25 cm	Stop, PWM 0
7	Obstacle removed	Resume cruise 200
10	Rear object at 30 cm	DANGER alert (buzzer)
13	Rear object clears	Alert cleared

Response Latency and Cost

The fused-sensing-to-actuation path within the Arduino completes in well under 100 ms per control iteration; the worst-case loop bound is set by the 25 ms ultrasonic timeout when both echo channels return no echo. The complete control and blind-spot subsystem is realized for under USD 50 in incremental hardware cost, dominated by the TF-Mini LiDAR.

Comparison with Prior Work

Table 8 positions the platform against a representative low-cost academic prototype [6] and a typical ultrasonic-only Arduino ACC. The present system adds true two-sensor fusion, dynamically scanned blind-spot coverage, and a resolved motor/servo timer conflict at comparable cost, while remaining fully reproducible from commodity parts.

Table 8. Comparison with Prior Work

Feature	This work	Prior [6]
LiDAR+US fusion	Yes	No
Validity filtering	Yes	No
Blind-spot scan	Servo 60–120°	No
Timer-1 conflict	Resolved	N/A
Cruise control law	Proportional	On/off
Cost (USD)	< 50	< 40

LIMITATIONS AND THREATS TO VALIDITY

The reported results should be read against the following limitations. First, all characterization was conducted indoors under controlled, stable illumination and on a flat, smooth surface at low speed (below roughly 0.5 m/s); the control law, fusion parameters, and blind-spot thresholds have not been validated on inclines, uneven terrain, with moving obstacles, or in the higher-speed regime where actuator and tyre dynamics become significant. Second, the proportional gain and the empirical minimum-speed PWM are tuned to this specific chassis, motor set, and battery state; transferring the platform would require re-identification of these parameters. Third, the sensor-reliability figures are drawn from 100 readings per surface at a single 80 cm range, so they characterize miss rate at that operating point rather than across the full range. Fourth, the single-byte serial protocol from the perception node is validated only by length and is not error-checked; electrical noise on the USB link could in principle corrupt a command, and a checksum field would harden the link at modest cost. Finally, the minimum-selection rule is deliberately conservative and does not estimate closing velocity, so it cannot anticipate a rapidly decelerating lead obstacle; this motivates the Kalman-filter extension discussed below.

CONCLUSION

A low-cost, reproducible ACC platform with LiDAR–ultrasonic fusion and servo-scanned blind-spot detection has been implemented and characterized on a single Arduino Uno. The fusion strategy attained a fused miss rate equal to the better constituent sensor on every surface, the proportional control law reproduced the commanded mapping to within ~1% mean absolute error, and the blind-spot scanner achieved zero false positives over repeated runs. The principal engineering contribution—resolving the AFMotor/Servo.h Timer 1 conflict via a bit-banged PWM routine—enables simultaneous motor drive and servo actuation that is otherwise unreliable on this platform. Future work includes outdoor and higher-speed validation, a probabilistic (Kalman) fusion that estimates closing velocity for anticipatory braking, and migration from USB UART to a CAN-bus interface for full-scale deployment.

REFERENCES

1. S. E. Shladover, "Review of the state of development of advanced vehicle control systems," Veh. Syst. Dyn., vol. 24, no. 6–7, pp. 551–595, 1995.
2. S. Moon, I. Moon, and K. Yi, "Design, tuning, and evaluation of a full-range adaptive cruise control system with collision avoidance," Control Eng. Pract., vol. 17, no. 4, pp. 442–455, 2009.
3. R. Rajamani, Vehicle Dynamics and Control, 2nd ed. New York, NY, USA: Springer, 2012.
4. A. Elfes, "Sonar-based real-world mapping and navigation," IEEE J. Robot. Autom., vol. 3, no. 3, pp. 249–265, 1987.
5. J. Alonso et al., "Ultrasonic sensor-based obstacle avoidance for indoor mobile robots," Sensors, vol. 14, no. 9, pp. 16486–16507, 2014.
6. S. Mehta, R. Singh, and A. Sharma, "Low-cost adaptive cruise control with drowsiness alert using Arduino and Raspberry Pi," Int. J. Eng. Res. Technol., vol. 8, no. 5, pp. 112–118, 2019.
7. Benewake Co. Ltd., TF-Mini LiDAR Module Product Manual, ver. 1.3.6, 2019.
8. Pathak A, Pathan RK, Tutul AU, Tousi NT, Rubaba AS, Bithi NY. Line follower robot for industrial manufacturing process. International Journal of Engineering Inventions. 2017 Oct;6(10):10-7.
9. World Health Organization, Road Traffic Injuries Fact Sheet, 2023.
10. ISO 17387:2008, Intelligent Transport Systems—Lane Change Decision Aid Systems (LCDAS)—Performance Requirements and Test Procedures.
11. Arduino LLC, Arduino Reference Documentation, 2024.
12. ISO 15622:2018, Intelligent Transport Systems—Adaptive Cruise Control Systems—Performance Requirements and Test Procedures.
13. SAE International, J3016: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, 2021.

APPENDIX A
Firmware Declarations and Setup

For reproducibility, the global declarations and initialization are reproduced below. Mutable safety parameters (stopping distance and maximum speed) are runtime variables so the perception node can adjust them without reflashing; the minimum speed and detection range are compile-time constants.

```
const int SERVO_PIN = 10, TRIG = A2;
const int ECHO_F = A3, ECHO_R = 2; int
servoPos = 90, servoDir = 2;
unsigned long lastPulse=0, lastMove=0, lastLCD=0;

void setup(){ Serial.begin(9600);
  tfSerial.begin(115200);
  lcd.init(); lcd.backlight();
  pinMode(SERVO_PIN,OUTPUT); pinMode(TRIG,OUTPUT);
  pinMode(ECHO_F,INPUT); pinMode(ECHO_R,INPUT);
  for (int i=0;i<30;i++){ sendServoPulse(90); delay(20); }
  lcd.clear(); lcd.print("SYSTEM READY");
}
```

```
#include <AFMotor.h> #include
<SoftwareSerial.h> #include
<Wire.h>
#include <LiquidCrystal_I2C.h>
// Servo.h deliberately excluded (Timer 1 conflict)

SoftwareSerial tfSerial(A0, A1);      // TF-
Mini LiquidCrystal_I2C lcd(0x27, 16, 2);
AF_DCMotor m1(1), m2(2), m3(3), m4(4);

int  SAFE_STOP_DIST = 30;      // runtime
(cm) int  MAX_SPEED    = 200;  // runtime
(PWM) char weatherMode = 'N';
char driverStatus    = 'N';  // N=normal,
D=drowsy const int MIN_SPEED         = 85;   //
stall floor const int MAX_DETECT_DIST = 150;
```

Listing 5. Library includes, parameter declarations, and setup(). The complete loop and helper routines appear as Listings 1–4.

```
void moveForward(int s){
  m1.setSpeed(s); m2.setSpeed(s);
  m3.setSpeed(s); m4.setSpeed(s);
  m1.run(FORWARD); m2.run(FORWARD);
  m3.run(FORWARD); m4.run(FORWARD);
}
void stopCar(){
  m1.run(RELEASE); m2.run(RELEASE);
  m3.run(RELEASE); m4.run(RELEASE);
}
void updateLCD(int f,int r,int spd,char
  d){ char line0[17];
  int pct = map(spd,0,MAX_SPEED,0,100);
  if (d=='D') strcpy(line0,"!!DRIVER
  ASLEEP!"); else sprintf(line0,"F:%3d
  S:%3d%%",f,pct);
  if (strncmp(line0,prevLine0,16)!=0){
    lcd.setCursor(0,0); lcd.print(line0);
    strcpy(prevLine0,line0);
  }
}
```

Listing 6. Motor actuation and anti-flicker LCD helpers.

The actuation and display helpers complete the listing. The motor helpers wrap the AFMotor channel calls so that all four wheels are driven symmetrically, and the display helper applies the double-buffering described in Section V-F.

APPENDIX B
Component Specifications

The principal sensing and actuation components are summarized below from manufacturer datasheets; ±10% variation is typical at this price point.

Table 9. Benewake TF-Mini LiDAR

Parameter	Value
Wavelength	850 nm (near-IR)
Range	0.3 – 12 m
Accuracy	±1% or ±1 cm
Update rate	100 Hz
Interface	UART, 115200 baud, 8N1
Frame	9 bytes (0x59 0x59 + 7)
Supply	5 V, < 120 mA

Table 10. HC-SR04 Ultrasonic Sensor

Parameter	Value
Range	2 – 400 cm
Resolution	0.3 cm
Beam angle	~15° cone
Trigger	10 µs HIGH pulse
Formula	d = t × 0.034 / 2 (cm)
Supply	5 V, 15–30 mA

Table 11. SG90 Micro Servo

Parameter	Value
Control	PWM, 50 Hz
0° / 90° / 180°	544 / 1472 / 2400 µs
Voltage	4.8 – 6 V
Stall torque	1.8 kg·cm at 4.8 V
Speed	0.1 s / 60°
Range	0 – 180°

