Adaptive Cruise Control Algorithm and System Modeling

Authors: Enrique Garcia, Jacob Tenorio, Luis Garcia, Okorie Agwu

Date: April 17, 2025

1 ABSTRACT

This project develops a MATLAB® simulation of an adaptive cruise control (ACC) system that automatically adjusts a follower vehicle's speed to maintain a safe distance from a lead car. A PID controller regulates the follower's acceleration based on real-time gap measurements, with system dynamics integrated first using Euler's method for initial validation and then MATLAB's ODE45 solver for high-fidelity scenario simulations. Three key scenarios are analyzed: steady-state cruising, sudden lead vehicle braking, and gradual speed variation. A time step sensitivity study verifies numerical accuracy, and the effects of controller parameters on system stability and responsiveness are explored. Results demonstrate that ODE45 provides smoother gap tracking and acceleration profiles than Euler's method, particularly under dynamic conditions. Overall, the simulation reinforces core principles of numerical methods and control theory while offering practical insights into dynamic vehicle-following systems.

2 INTRODUCTION

Adaptive cruise control (ACC) systems represent a significant evolution in automotive safety and automation, allowing vehicles to autonomously adjust their speed based on the relative position and velocity of nearby traffic. Unlike traditional cruise control, which maintains a constant speed regardless of conditions, ACC continuously regulates acceleration and braking to preserve a safe following distance. These systems enhance driver comfort, reduce fatigue, and contribute to safer traffic flow, especially in congested or variable-speed environments [1].

This project models a simplified one-dimensional ACC system in MATLAB®, focusing on the dynamic interaction between a lead vehicle and a follower vehicle. The follower's acceleration is governed by a proportional-integral-derivative (PID) controller [2], which responds to real-time gap measurements to maintain a desired distance. Three representative driving scenarios are simulated—steady-state cruising, emergency braking, and gradual speed variation—to evaluate controller performance under varying conditions. As an additional scope, this work focuses on reformulating the system dynamics into ordinary differential equations (ODEs) solvable by MATLAB's ODE45, enabling high-fidelity numerical integration of vehicle behavior. Euler's method is also implemented for conceptual validation and time step sensitivity analysis.

The sections that follow present the mathematical modeling of the system, the control algorithm, and the numerical solvers used. Simulation results compare system responses under different PID parameters and numerical schemes, highlighting the trade-offs in accuracy, responsiveness, and stability. The report concludes with a discussion of key findings, practical implications for ACC design, and suggestions for further refinement.

3 METHODS

A one-dimensional adaptive cruise control (ACC) system was developed in MATLAB®, simulating interactions between a lead vehicle and a follower vehicle. A PID controller regulates the follower's acceleration to maintain safe following distances. Numerical integration was performed using both Euler's method (for verification and time step sensitivity studies) and MATLAB's `ODE45` solver [3] (for primary scenario simulations).

3.1 Lead Vehicle Modeling

The lead vehicle's motion was defined by three regimes:

$$v_l(t) = \begin{cases} v_0 & \text{(constant speed)} \\ v_0 + \alpha t & \text{(acceleration)} \\ v_0 - \beta t & \text{(emergency braking)} \end{cases} \tag{1}$$

where:

- $v_0 = 80.685\,\text{ft/s}$ (55 mph) represents highway cruising speed,
- $\alpha = 2.5\,\text{m/s}^2$ models moderate acceleration,
- $\beta = 8\,\text{m/s}^2$ models emergency braking at approximately $0.8g$.

Position was updated at each time step by:

$$x_l(t + \Delta t) = x_l(t) + v_l(t)\Delta t \tag{2}$$

where Δt is the simulation time step.

3.2 Follower Vehicle Modeling

The follower aims to maintain a safe, speed-dependent gap defined by:

$$gap_{\text{target}}(v_f) = \max(gap_{\min}, \tau v_f) \tag{3}$$

with parameters:

- $gap_{\min} = 5\,\text{m}$ (fixed minimum gap),

- $\tau = 2\,\mathrm{s}$ (2-second rule for safe following time),
- v_f (follower vehicle velocity in ft/s).

Mode switching was implemented:

$$\text{Mode} = \begin{cases} \text{Target speed tracking} & \text{if } gap_{\text{measured}} \geq gap_{\text{target}} \\ \text{Gap maintenance} & \text{if } gap_{\text{measured}} < gap_{\text{target}} \end{cases} \tag{4}$$

where $gap_{\text{measured}} = x_l(t) - x_f(t)$.

3.2.1 PID Control Algorithm

The follower's acceleration is governed by a PID controller:

$$a(t) = K_p e(t) + K_d \frac{de}{dt} + K_i \int_0^t e(\tau)\, d\tau \tag{5}$$

where:

- $e(t) = gap_{\text{measured}} - gap_{\text{target}}$: Gap error, driving corrective actions.
- $K_p = 0.5$: Proportional gain, weighting immediate error correction.
- $K_d = 0.8$: Derivative gain, damping oscillations from rapid error changes.
- $K_i = 0$: Integral gain (disabled in this implementation), which would address persistent steady-state errors.

The derivative term was approximated by backward finite difference:

$$\frac{de}{dt} \approx \frac{e(t) - e(t - \Delta t)}{\Delta t} \tag{6}$$

and acceleration bounds were applied:

$$-17.6\,\mathrm{ft/s^2} \leq a(t) \leq 11.2\,\mathrm{ft/s^2} \tag{7}$$

to reflect realistic passenger vehicle limits.

3.3 Numerical Methods

Euler's Method (for Verification). The follower's velocity and position were updated at each fixed time step:

$$v_f^{n+1} = v_f^n + \Delta t\, a^n, \quad x_f^{n+1} = x_f^n + \Delta t\, v_f^n \tag{8}$$

with Δt varying during time step sensitivity studies.

ODE45 Solver (for Scenario Simulations). The primary simulations used MATLAB's adaptive `ODE45` solver to integrate:

$$\frac{d}{dt}\begin{bmatrix} x_f \\ v_f \end{bmatrix} = \begin{bmatrix} v_f \\ a(t) \end{bmatrix} \tag{9}$$

with a relative tolerance of $\epsilon_{\text{rel}} = 10^{-3}$ for error control.

3.4 Convergence Validation

A time step sensitivity study using Euler's method verified numerical accuracy. Convergence was achieved for $\Delta t \leq 0.1$ s, and results compared to ODE45 showed less than 0.1% deviation in gap tracking and follower velocity across all tested scenarios.

4 Results

4.1 Solver Comparison: Euler vs. ODE45

To validate the model, both Euler's method and `ODE45` were applied to simulate the follower's response to a constant-speed lead vehicle. Figure 1 compares the resulting gap and acceleration profiles. ODE45 produces smoother acceleration and more stable gap tracking, especially during transient events, demonstrating its improved accuracy and dynamic time-stepping capabilities compared to the fixed-step Euler method.

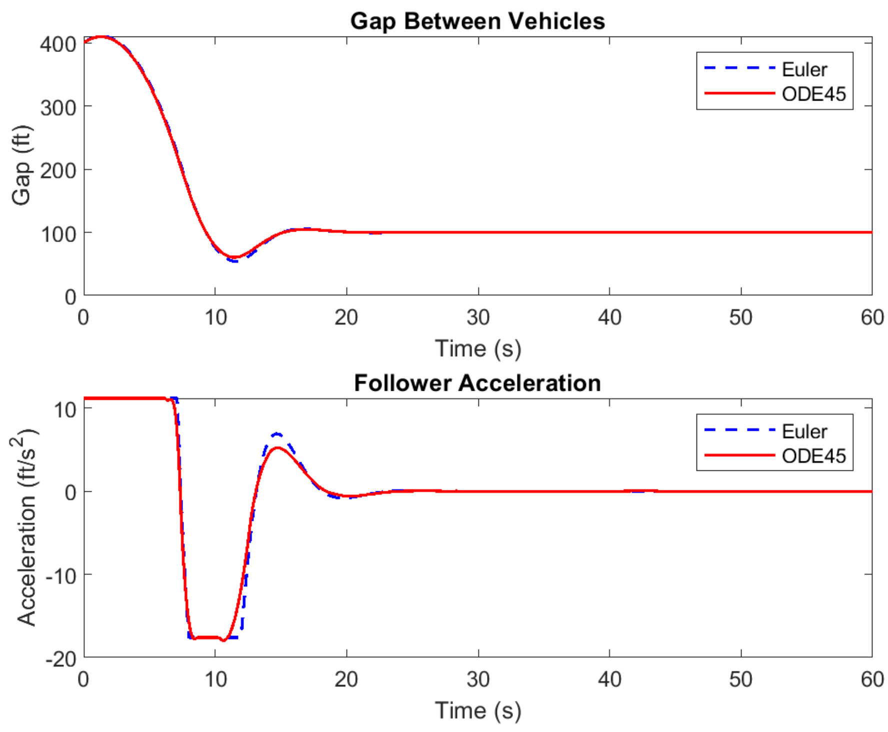

Figure 1 Gap and acceleration using Euler vs. ODE45.

4.2 Scenario Simulations Using ODE45

Scenario 1: Closing the Gap. In the first scenario, the follower begins at a higher speed than the lead vehicle and decelerates to establish the desired gap. As shown in Figure 2, the gap converges smoothly to the target value, with minimal overshoot, illustrating stable speed and position tracking under steady-state highway cruising conditions.

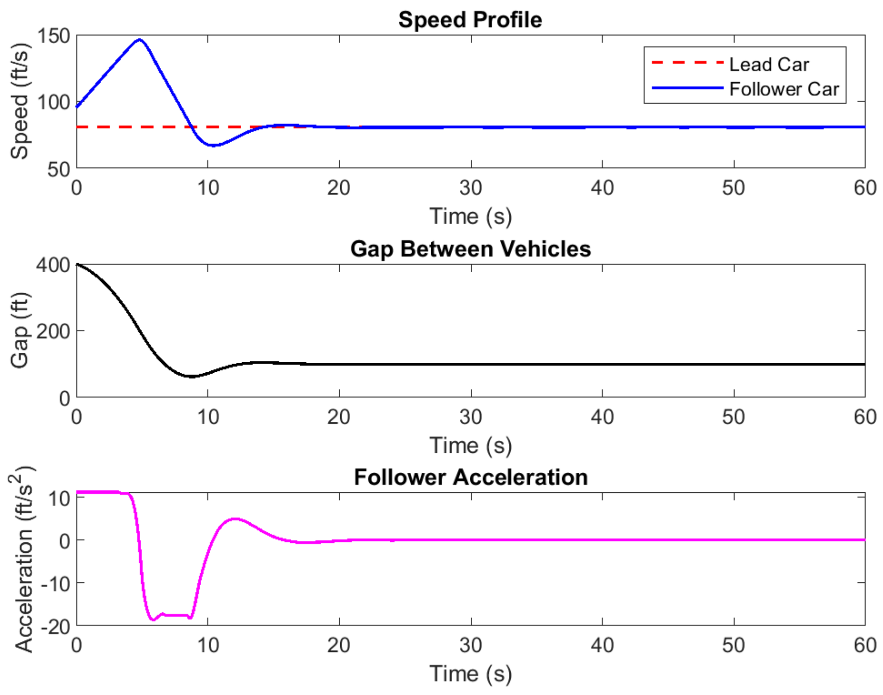

Figure 2 Scenario 1 – Gap convergence and speed matching.

Scenario 2: Emergency Braking. In this case, the lead car abruptly decelerates to simulate an obstacle on the road. Figure 3 shows the follower's rapid response, successfully maintaining a safe distance without collision. The follower's acceleration profile remains within realistic physical limits, highlighting effective emergency response under PID control.

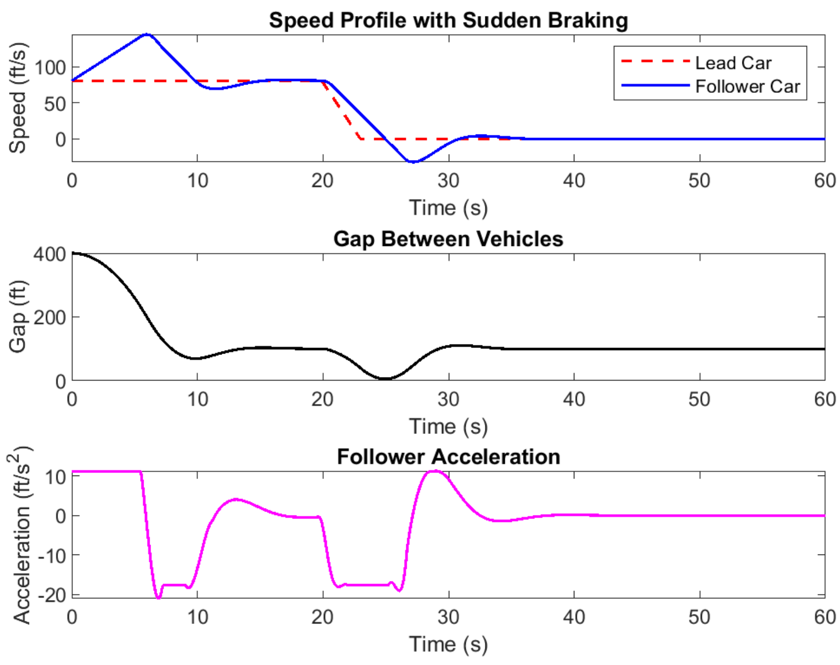

Figure 3 Scenario 2 – Follower reacts to sudden braking.

Scenario 3: Gradual Speed Variation. Here, the lead vehicle smoothly accelerates and decelerates over time. Figure 4 illustrates the follower's ability to track these changes with minimal lag, maintaining the desired gap despite continuous speed variations.

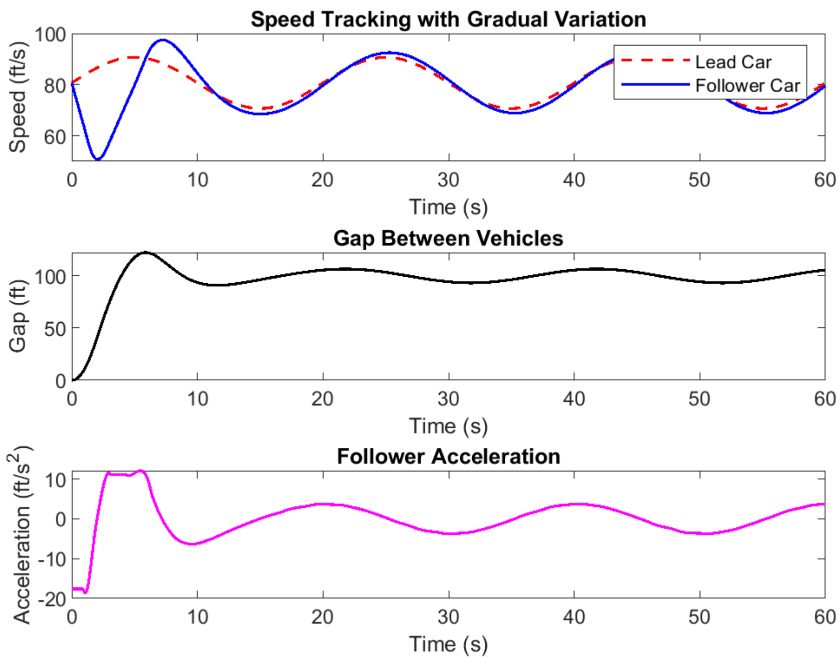

Figure 4 Scenario 3 – Follower tracks gradual speed changes.

4.3 Time Step Sensitivity (Euler)

To verify numerical accuracy, Euler simulations were repeated using progressively smaller time steps. As shown in Figure 5, reducing the time step significantly improves solution stability and smoothness, confirming that $\Delta t = 0.1$ s or smaller is necessary to avoid noticeable numerical errors.

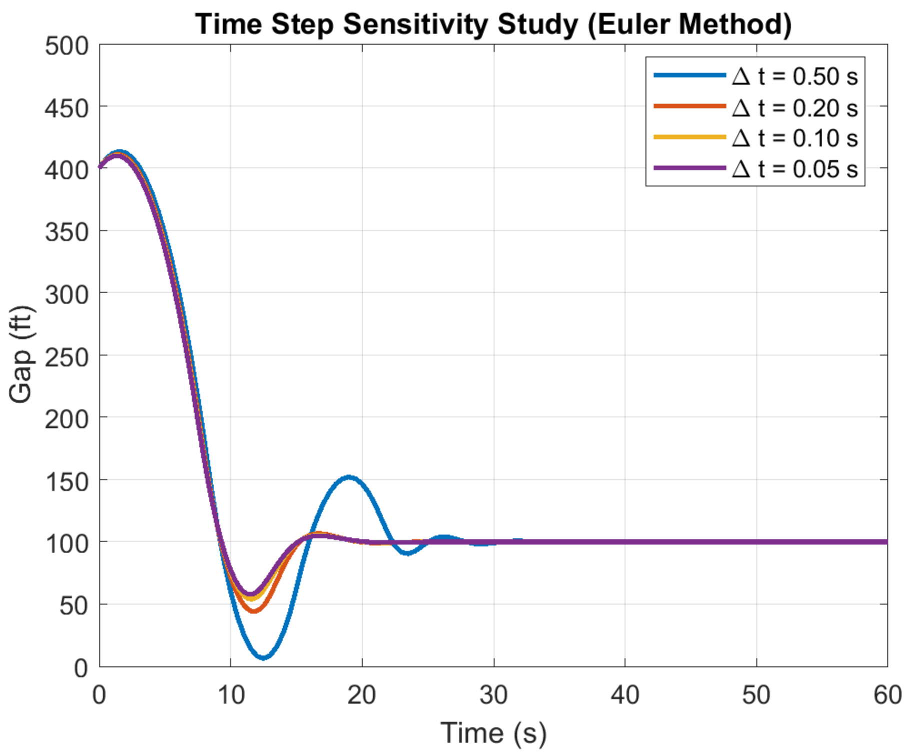

Figure 5 Gap vs. time for varying Euler time steps. Smaller time steps yield smoother, more accurate solutions.

4.4 Parameter Effects

Proportional Gain (K_p). The proportional gain affects how aggressively the follower responds to gap errors. Figure 6 shows that low K_p leads to slow convergence to the target gap, while high K_p results in faster responses but introduces overshoot and mild oscillations.

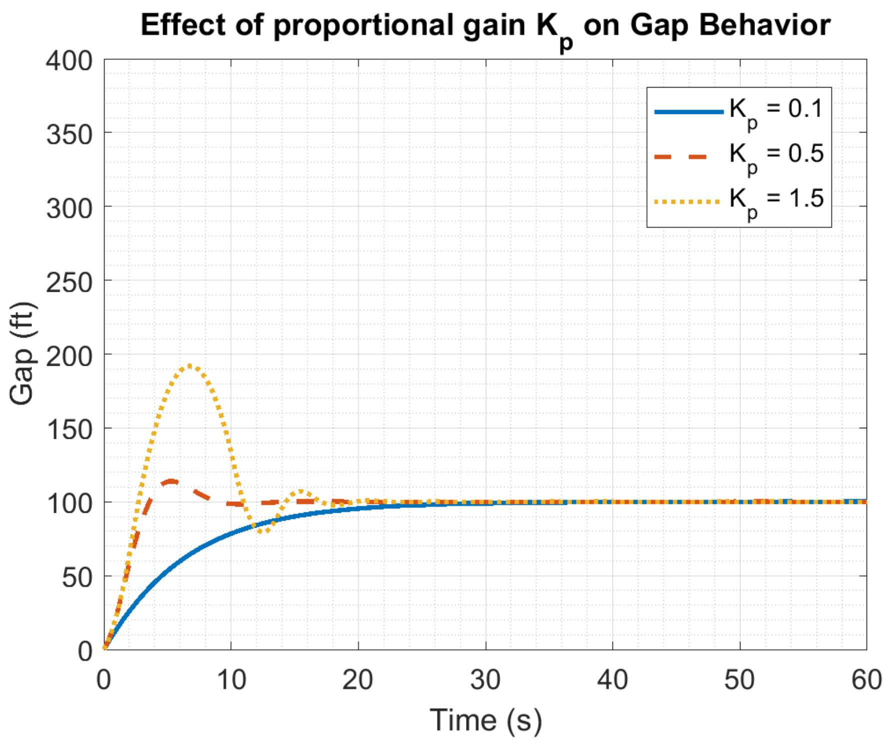

Figure 6 **Effect of K_p on gap behavior.**

Derivative Gain (K_d). The derivative gain acts to dampen the system's response by penalizing rapid changes in error. As seen in Figure 7, low K_d values allow oscillations, while higher K_d values smooth the follower's motion and suppress oscillations, though overly high K_d can slightly slow response time.

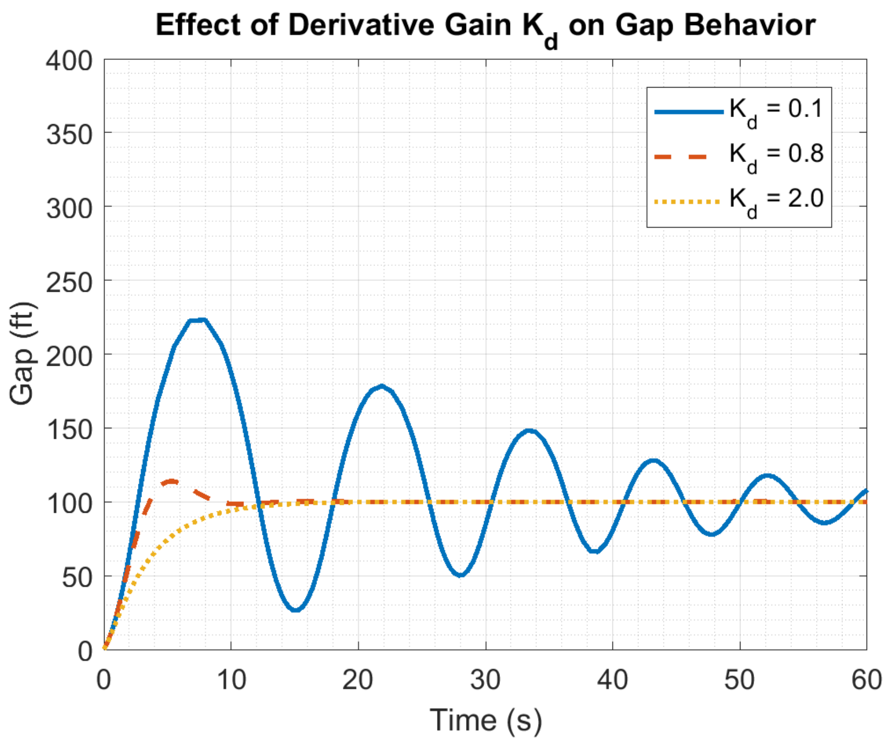

Figure 7 **Effect of K_d on system damping.**

5 Discussion

The simulation results demonstrate that a PID-controlled adaptive cruise control (ACC) system can maintain safe following distances under a range of dynamic driving scenarios. In the steady-state case, the follower vehicle gradually converged toward the desired gap without major overshoot, while in emergency braking, the system successfully decelerated to avoid collision within realistic acceleration limits. During gradual lead vehicle speed changes, the follower closely tracked the variations with minor lag. Solver comparison showed that ODE45 produced smoother gap tracking and acceleration profiles compared to Euler's method, especially during rapid changes in lead vehicle behavior. These findings validate the use of adaptive solvers for accurate and stable simulation of real-time vehicle dynamics.

The parameter studies provide further insights into system behavior. Low proportional gain (K_p) values resulted in sluggish responses and delayed gap closure, while high K_p values accelerated the response but introduced overshoot and oscillations. Increasing derivative gain (K_d) improved system damping, reducing oscillations and stabilizing the gap, although overly high K_d could slightly slow the system's responsiveness. These trends align with classical control theory, highlighting important design trade-offs between responsiveness, stability, and comfort. Additionally, the time step sensitivity study indicated that Euler's method requires sufficiently small time steps ($\Delta t \leq 0.1\,\mathrm{s}$) to avoid numerical instability, whereas ODE45 adaptively manages accuracy with minimal computational overhead.

Sources of error in the simulation include discretization errors from Euler's method, idealized assumptions of perfect sensors and actuators, and omission of external factors such as road grade, aerodynamic drag, or inter-vehicle communication delays. These limitations mean that while the model captures the essential behavior of ACC systems, it simplifies the complexity of real-world driving. Future work could improve realism by introducing sensor noise, time delay modeling, actuator saturation effects, and multi-vehicle interaction scenarios. Additionally, exploring more advanced control strategies such as model predictive control (MPC), adaptive PID tuning, or machine-learning-enhanced feedback could further improve system robustness and make the simulations more representative of practical ACC designs.

6 Conclusion

This project developed a MATLAB-based simulation of an adaptive cruise control (ACC) system using PID control and numerical integration methods. By comparing Euler's method and ODE45, we demonstrated that adaptive solvers provide superior accuracy and stability for dynamic vehicle modeling. Scenario testing and PID parameter studies confirmed the importance of careful controller tuning to balance responsiveness and stability. Overall, the results validate the feasibility of PID-based ACC systems while highlighting opportunities for future refinement through more advanced modeling and control strategies.

References

[1] Marcus P.S. de Abreu, F. O. S., Fúlvia S.S. de Oliveira, 2024, “An improved adaptive cruise control law,” Science Direct, accessed 2025, https://www.sciencedirect.com/science/article/pii/S0921889024000629
[2] Douglas, B., 2025, “PID Control - A brief introduction,” accessed 2025, https://youtu.be/UR0hOmjaHp0
[3] “ode45 documentation,” MathWorks, accessed 2025, https://www.mathworks.com/help/matlab/ref/ode45.html

7 APPENDIX

MATLAB Code

```
%% Solver Comparison: Euler vs. ODE45

% --- Time settings ---
dt = 0.1;
T = 60;
time = 0:dt:T;

% --- Lead vehicle (constant speed) ---
x_lead = zeros(size(time));
v_lead = 55 * 1.467 * ones(size(time)); % 55 mph in ft/s
x_lead(1) = 400; % Initial position

for i = 2:length(time)
    x_lead(i) = x_lead(i-1) + v_lead(i-1) * dt;
end

% --- Follower vehicle (Euler method) ---
x_follower = zeros(size(time));
v_follower = zeros(size(time));
a_follower = zeros(size(time));

x_follower(1) = 0;
v_follower(1) = 45 * 1.467; % 45 mph

% PID gains
Kp = 0.5;
Kd = 0.8;
Ki = 0;
integral_error = 0;
target_gap = 100;

for i = 1:length(time)-1
    gap = x_lead(i) - x_follower(i);
    error = gap - target_gap;

    if i == 1
        d_error = 0;
    else
        d_error = (error - (x_lead(i-1) - x_follower(i-1) - target_gap)) / dt;
    end

    integral_error = integral_error + error * dt;
    a_follower(i) = Kp * error + Kd * d_error + Ki * integral_error;
    a_follower(i) = max(min(a_follower(i), 11.2), -17.6);

    v_follower(i+1) = v_follower(i) + a_follower(i) * dt;
    x_follower(i+1) = x_follower(i) + v_follower(i) * dt;
end

% --- ODE45 simulation ---
params.Kp = Kp;
params.Kd = Kd;
params.Ki = Ki;
params.target_gap = target_gap;
params.v_lead = v_lead;
params.x_lead = x_lead;
params.time = time;

y0 = [0; 45 * 1.467; 0]; % Initial [x_follower; v_follower; integral_error]
[t_ode, y_ode] = ode45(@(t, y) follower_ode(t, y, params), time, y0);

x_lead_interp = interp1(time, x_lead, t_ode, 'linear', 'extrap');
gap_ode = x_lead_interp - y_ode(:,1);
accel_ode = diff(y_ode(:,2)) ./ diff(t_ode);
accel_ode(end+1) = accel_ode(end);

% --- Plot: Euler vs ODE45 ---
figure;
subplot(2,1,1);
plot(time, x_lead - x_follower, 'b--', t_ode, gap_ode, 'r', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Gap (ft)');
legend('Euler', 'ODE45');
title('Gap Between Vehicles');

subplot(2,1,2);
plot(time, a_follower, 'b--', t_ode, accel_ode, 'r', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Acceleration (ft/s^2)');
legend('Euler', 'ODE45');
title('Follower Acceleration');

% --- ODE function definition ---
function dydt = follower_ode(t, y, params)
    x_f = y(1);
    v_f = y(2);
    int_err = y(3);

    x_lead_t = interp1(params.time, params.x_lead, t, 'linear', 'extrap');
    v_lead_t = interp1(params.time, params.v_lead, t, 'linear', 'extrap');

    gap = x_lead_t - x_f;
    error = gap - params.target_gap;
    d_error = v_lead_t - v_f;

    a = params.Kp * error + params.Kd * d_error + params.Ki * int_err;
    a = max(min(a, 11.2), -17.6);

    dydt = [v_f; a; error];
end
```

```
%% Scenario 1: Closing the Gap

% --- Time settings ---
dt = 0.1;
T = 60;
time = 0:dt:T;

% --- Lead vehicle (constant speed) ---
x_lead = zeros(size(time));
v_lead = 55 * 1.467 * ones(size(time)); % 55 mph -> ft/s
x_lead(1) = 400; % Initial position

for i = 2:length(time)
    x_lead(i) = x_lead(i-1) + v_lead(i-1) * dt;
end

% --- PID Controller Parameters ---
Kp = 0.5;
Kd = 0.8;
Ki = 0; % Not used
target_gap = 100; % Target gap (ft)

% --- Pack parameters for ODE45 ---
params.Kp = Kp;
params.Kd = Kd;
params.Ki = Ki;
params.target_gap = target_gap;
params.v_lead = v_lead;
params.x_lead = x_lead;
params.time = time;

% --- Initial Conditions ---
y0 = [0; 65 * 1.467; 0]; % Start faster: 65 mph (follower), [position; ...
    velocity; integral error]

% --- Solve ODE ---
[t_ode, y_ode] = ode45(@(t, y) follower_ode(t, y, params), time, y0);

% --- Process Results ---
x_follower = y_ode(:,1);
v_follower = y_ode(:,2);

x_lead_interp = interp1(time, x_lead, t_ode, 'linear', 'extrap');
gap = x_lead_interp - x_follower;

accel = diff(v_follower) ./ diff(t_ode);
accel(end+1) = accel(end); % Pad to match vector size

% --- Plot Scenario 1 ---
figure;

subplot(3,1,1);
plot(t_ode, v_lead(1)*ones(size(t_ode)), 'r--', t_ode, v_follower, 'b', ...
   'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Speed (ft/s)');
legend('Lead Car', 'Follower Car');
title('Speed Profile');

subplot(3,1,2);
plot(t_ode, gap, 'k', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Gap (ft)');
title('Gap Between Vehicles');

subplot(3,1,3);
plot(t_ode, accel, 'm', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Acceleration (ft/s^2)');
title('Follower Acceleration');

% --- ODE function for follower dynamics ---
function dydt = follower_ode(t, y, params)
    x_f = y(1);
    v_f = y(2);
    int_err = y(3);

    x_lead_t = interp1(params.time, params.x_lead, t, 'linear', 'extrap');
    v_lead_t = interp1(params.time, params.v_lead, t, 'linear', 'extrap');

    gap = x_lead_t - x_f;
    error = gap - params.target_gap;
    d_error = v_lead_t - v_f;

    a = params.Kp * error + params.Kd * d_error + params.Ki * int_err;
    a = max(min(a, 11.2), -17.6); % Apply acceleration limits

    dydt = [v_f; a; error];
end
```

```
%% Scenario 2: Emergency Braking

% --- Time settings ---
dt = 0.1;
T = 60;
time = 0:dt:T;

% --- Lead vehicle (constant speed until sudden braking) ---
x_lead = zeros(size(time));
v_lead = 55 * 1.467 * ones(size(time)); % 55 mph -> ft/s
x_lead(1) = 400; % Initial position

% Parameters for sudden braking
brake_start_time = 20; % Lead car starts braking at 20 seconds
brake_decel = 8; % m/s^2 -> 26.24 ft/s^2
brake_decel_ft = brake_decel * 3.281; % convert m/s^2 to ft/s^2

for i = 2:length(time)
    if time(i) < brake_start_time
```

```
        v_lead(i) = v_lead(i-1); % constant speed
    else
        v_lead(i) = max(0, v_lead(i-1) - brake_decel_ft * dt); % decelerate
    end
    x_lead(i) = x_lead(i-1) + v_lead(i-1) * dt;
end

% --- PID Controller Parameters ---
Kp = 0.5;
Kd = 0.8;
Ki = 0; % Not used
target_gap = 100; % Target gap (ft)

% --- Pack parameters for ODE45 ---
params.Kp = Kp;
params.Kd = Kd;
params.Ki = Ki;
params.target_gap = target_gap;
params.v_lead = v_lead;
params.x_lead = x_lead;
params.time = time;

% --- Initial Conditions ---
y0 = [0; 55 * 1.467; 0]; % Follower initially matches lead car (55 mph)

% --- Solve ODE ---
[t_ode, y_ode] = ode45(@(t, y) follower_ode(t, y, params), time, y0);

% --- Process Results ---
x_follower = y_ode(:,1);
v_follower = y_ode(:,2);

x_lead_interp = interp1(time, x_lead, t_ode, 'linear', 'extrap');
gap = x_lead_interp - x_follower;

accel = diff(v_follower) ./ diff(t_ode);
accel(end+1) = accel(end); % Pad to match vector size

% --- Plot Scenario 2 ---
figure;

subplot(3,1,1);
plot(t_ode, interp1(time, v_lead, t_ode), 'r--', t_ode, v_follower, 'b', ...
    'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Speed (ft/s)');
legend('Lead Car', 'Follower Car');
title('Speed Profile with Sudden Braking');

subplot(3,1,2);
plot(t_ode, gap, 'k', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Gap (ft)');
title('Gap Between Vehicles');

subplot(3,1,3);
plot(t_ode, accel, 'm', 'LineWidth', 1.2);
```

```
xlabel('Time (s)');
ylabel('Acceleration (ft/s^2)');
title('Follower Acceleration');

% --- ODE function for follower dynamics ---
function dydt = follower_ode(t, y, params)
    x_f = y(1);
    v_f = y(2);
    int_err = y(3);

    x_lead_t = interp1(params.time, params.x_lead, t, 'linear', 'extrap');
    v_lead_t = interp1(params.time, params.v_lead, t, 'linear', 'extrap');

    gap = x_lead_t - x_f;
    error = gap - params.target_gap;
    d_error = v_lead_t - v_f;

    a = params.Kp * error + params.Kd * d_error + params.Ki * int_err;
    a = max(min(a, 11.2), -17.6); % Apply acceleration limits

    dydt = [v_f; a; error];
end
```

```
%% Scenario 3: Gradual Speed Variation

% --- Time settings ---
dt = 0.1;
T = 60;
time = 0:dt:T;

% --- Lead vehicle (gradual speed variation: sinusoidal profile) ---
v_base = 55 * 1.467;           % Base speed: 55 mph in ft/s
amp = 10;                      % Amplitude of variation (ft/s)
freq = 0.05;                   % Frequency (Hz), or cycles/sec

v_lead = v_base + amp * sin(2*pi*freq*time);  % Smooth sinusoidal speed profile
x_lead = cumtrapz(time, v_lead);              % Integrate to get position

% --- PID Controller Parameters ---
Kp = 0.5;
Kd = 0.8;
Ki = 0;
target_gap = 100;

% --- Pack parameters for ODE45 ---
params.Kp = Kp;
params.Kd = Kd;
params.Ki = Ki;
params.target_gap = target_gap;
params.v_lead = v_lead;
params.x_lead = x_lead;
params.time = time;

% --- Initial Conditions ---
y0 = [0; 55 * 1.467; 0]; % Initial follower state [x, v, int_err]

```

```
% --- Solve ODE ---
[t_ode, y_ode] = ode45(@(t, y) follower_ode(t, y, params), time, y0);

% --- Post-processing ---
x_follower = y_ode(:,1);
v_follower = y_ode(:,2);
x_lead_interp = interp1(time, x_lead, t_ode, 'linear', 'extrap');
v_lead_interp = interp1(time, v_lead, t_ode, 'linear', 'extrap');
gap = x_lead_interp - x_follower;
accel = diff(v_follower) ./ diff(t_ode);
accel(end+1) = accel(end);

% --- Plot Scenario 3 ---
figure;

subplot(3,1,1);
plot(t_ode, v_lead_interp, 'r--', t_ode, v_follower, 'b', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Speed (ft/s)');
legend('Lead Car', 'Follower Car');
title('Speed Tracking with Gradual Variation');

subplot(3,1,2);
plot(t_ode, gap, 'k', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Gap (ft)');
title('Gap Between Vehicles');

subplot(3,1,3);
plot(t_ode, accel, 'm', 'LineWidth', 1.2);
xlabel('Time (s)');
ylabel('Acceleration (ft/s^2)');
title('Follower Acceleration');

% --- ODE function ---
function dydt = follower_ode(t, y, params)
    x_f = y(1);
    v_f = y(2);
    int_err = y(3);

    x_lead_t = interp1(params.time, params.x_lead, t, 'linear', 'extrap');
    v_lead_t = interp1(params.time, params.v_lead, t, 'linear', 'extrap');

    gap = x_lead_t - x_f;
    error = gap - params.target_gap;
    d_error = v_lead_t - v_f;

    a = params.Kp * error + params.Kd * d_error + params.Ki * int_err;
    a = max(min(a, 11.2), -17.6);

    dydt = [v_f; a; error];
end
```

```
%% Time Step Sensitivity (Euler)

% --- Simulation settings ---
```

```
T = 60; % total simulation time (seconds)
dt_values = [0.5, 0.2, 0.1, 0.05]; % different time steps to test

% --- Lead car parameters (constant speed) ---
v_lead_const = 55 * 1.467; % 55 mph in ft/s
x_lead_init = 400; % Initial position

% --- Follower car initial conditions ---
v_follower_init = 45 * 1.467; % 45 mph in ft/s

% --- PID parameters ---
Kp = 0.5;
Kd = 0.8;
Ki = 0;
target_gap = 100; % ft

% --- Prepare figure ---
figure;
hold on;
colors = lines(length(dt_values)); % Automatically assign distinct colors

for j = 1:length(dt_values)
    dt = dt_values(j);
    time = 0:dt:T;

    % Preallocate arrays
    x_lead = zeros(size(time));
    v_lead = v_lead_const * ones(size(time));
    x_follower = zeros(size(time));
    v_follower = zeros(size(time));
    a_follower = zeros(size(time));
    integral_error = 0;

    % Initial conditions
    x_lead(1) = x_lead_init;
    x_follower(1) = 0;
    v_follower(1) = v_follower_init;

    for i = 1:length(time)-1
        % Lead car (constant speed)
        x_lead(i+1) = x_lead(i) + v_lead(i) * dt;

        % Gap and error
        gap = x_lead(i) - x_follower(i);
        error = gap - target_gap;

        if i == 1
            d_error = 0;
        else
            d_error = (error - (x_lead(i-1) - x_follower(i-1) - target_gap)) / dt;
        end
        integral_error = integral_error + error * dt;

        % PID control for acceleration
        a = Kp * error + Kd * d_error + Ki * integral_error;
        a = max(min(a, 11.2), -17.6); % acceleration bounds

        % Euler integration for follower
        v_follower(i+1) = v_follower(i) + a * dt;
        x_follower(i+1) = x_follower(i) + v_follower(i) * dt;
    end

    % Plot gap vs. time for this dt
    plot(time, x_lead - x_follower, 'LineWidth', 2, 'Color', colors(j,:), ...
         'DisplayName', sprintf('\\Delta t = %.2f s', dt));
end

xlabel('Time (s)', 'FontSize', 12);
ylabel('Gap (ft)', 'FontSize', 12);
title('Time Step Sensitivity Study (Euler Method)', 'FontSize', 14, ...
    'FontWeight', 'bold');
legend('Location','best', 'FontSize', 10);
grid on;
set(gca, 'FontSize', 11);
xlim([0 T]);
ylim([0 500]);
box on;
hold off;
```

```
%% PID Parameter Effects K_p

% --- Time settings ---
dt = 0.1;
T = 60;
time = 0:dt:T;

% --- Lead vehicle (constant speed) ---
v_lead_const = 55 * 1.467; % 55 mph in ft/s
v_lead = v_lead_const * ones(size(time));
x_lead = cumtrapz(time, v_lead); % Integrate to get position

% --- Follower car settings ---
Kd = 0.8; % Derivative gain (fixed)
Ki = 0;   % Integral gain (fixed)
target_gap = 100; % Target gap (ft)

% --- Range of Kp values to study ---
Kp_values = [0.1, 0.5, 1.5]; % Low, medium, high

% --- Prepare plot ---
figure;
hold on;
colors = lines(length(Kp_values));
styles = {'-', '--', ':'};

for j = 1:length(Kp_values)
    % Pack parameters for ODE
    params.Kp = Kp_values(j);
    params.Kd = Kd;
    params.Ki = Ki;
    params.target_gap = target_gap;
    params.v_lead = v_lead;
    params.x_lead = x_lead;
    params.time = time;

    % Initial conditions
    y0 = [0; 45 * 1.467; 0]; % Start slower than lead car

    % ODE45 simulation
    [t_ode, y_ode] = ode45(@(t, y) follower_ode(t, y, params), time, y0);

    % Process gap
    x_follower = y_ode(:,1);
    x_lead_interp = interp1(time, x_lead, t_ode, 'linear', 'extrap');
    gap = x_lead_interp - x_follower;

    % Plot
    plot(t_ode, gap, 'LineWidth', 2, 'Color', colors(j,:), ...
         'LineStyle', styles{j}, 'DisplayName', sprintf('K_p = %.1f', ...
            Kp_values(j)));
end

xlabel('Time (s)', 'FontSize', 13);
ylabel('Gap (ft)', 'FontSize', 13);
title('Effect of proportional gain K_p on Gap Behavior', 'FontSize', 15, ...
    'FontWeight', 'bold');
legend('Location','best', 'FontSize', 11);
grid on;
grid minor;
set(gca, 'FontSize', 12);
xlim([0 T]);
ylim([0 400]);
box on;
hold off;

% --- ODE function ---
function dydt = follower_ode(t, y, params)
    x_f = y(1);
    v_f = y(2);
    int_err = y(3);

    x_lead_t = interp1(params.time, params.x_lead, t, 'linear', 'extrap');
    v_lead_t = interp1(params.time, params.v_lead, t, 'linear', 'extrap');

    gap = x_lead_t - x_f;
    error = gap - params.target_gap;
    d_error = v_lead_t - v_f;

    a = params.Kp * error + params.Kd * d_error + params.Ki * int_err;
    a = max(min(a, 11.2), -17.6);

    dydt = [v_f; a; error];
end
```

```
%% PID Parameter Effects K_d

% --- Time settings ---
dt = 0.1;
T = 60;
```

```
time = 0:dt:T;

% --- Lead vehicle (constant speed) ---
v_lead_const = 55 * 1.467; % 55 mph in ft/s
v_lead = v_lead_const * ones(size(time));
x_lead = cumtrapz(time, v_lead); % Integrate to get position

% --- Follower car settings ---
Kp = 0.5; % Proportional gain (fixed now)
Ki = 0;   % Integral gain (fixed)
target_gap = 100; % Target gap (ft)

% --- Range of Kd values to study ---
Kd_values = [0.1, 0.8, 2.0]; % Low, medium, high

% --- Prepare plot ---
figure;
hold on;
colors = lines(length(Kd_values));
styles = {'-', '--', ':'};

for j = 1:length(Kd_values)
    % Pack parameters for ODE
    params.Kp = Kp;
    params.Kd = Kd_values(j);
    params.Ki = Ki;
    params.target_gap = target_gap;
    params.v_lead = v_lead;
    params.x_lead = x_lead;
    params.time = time;

    % Initial conditions
    y0 = [0; 45 * 1.467; 0]; % Start slower than lead car

    % ODE45 simulation
    [t_ode, y_ode] = ode45(@(t, y) follower_ode(t, y, params), [0 T], y0);

    % Process gap
    x_follower = y_ode(:,1);
    x_lead_interp = interp1(time, x_lead, t_ode, 'linear', 'extrap');
    gap = x_lead_interp - x_follower;

    % Plot
    plot(t_ode, gap, 'LineWidth', 2, 'Color', colors(j,:), ...
         'LineStyle', styles{j}, 'DisplayName', sprintf('K_d = %.1f', ...
            Kd_values(j)));
end

xlabel('Time (s)', 'FontSize', 13);
ylabel('Gap (ft)', 'FontSize', 13);
title('Effect of Derivative Gain K_d on Gap Behavior', 'FontSize', 15, ...
    'FontWeight', 'bold');
legend('Location','best', 'FontSize', 11);
grid on;
grid minor;
set(gca, 'FontSize', 12);
xlim([0 T]);
```

```
ylim([0 400]);
box on;
hold off;

% --- ODE function ---
function dydt = follower_ode(t, y, params)
    x_f = y(1);
    v_f = y(2);
    int_err = y(3);

    x_lead_t = interp1(params.time, params.x_lead, t, 'linear', 'extrap');
    v_lead_t = interp1(params.time, params.v_lead, t, 'linear', 'extrap');

    gap = x_lead_t - x_f;
    error = gap - params.target_gap;
    d_error = v_lead_t - v_f;

    a = params.Kp * error + params.Kd * d_error + params.Ki * int_err;
    a = max(min(a, 11.2), -17.6);

    dydt = [v_f; a; error];
end
```