Analysis of Lateral Offset Invariance in Parallel Parking Maneuver: A Geometric Simulation Study

Kevin Luo and Hsinghan Meng

Sunny Hills High School, Fullerton, CA 92833, SAT Professionals, Diamond Bar, CA 91765

Abstract. This paper investigates whether the initial lateral offset of a vehicle from the parking space, denoted Δy, affects the vehicle's ability to successfully complete a parallel parking maneuver. Using the geometric mathematical model developed by Wahab et al. [1] and implemented via a simulation developed in a Java-based environment to allow for high-fidelity kinematic modelling and real-time geometric validation, a series of trials were conducted in which Δy was systematically varied while all other vehicle and parking space parameters were held constant. The results demonstrate that, under the constraints of this model, the parallel parking maneuver can be completed successfully across a wide range of Δy values. Specifically, varying Δy causes the geometric solution to self-adjust: the intermediate quantities a, c, y_1, and θ all recalculate such that a valid two-arc trajectory always exists. It is concluded that Δy does not prevent a vehicle from parking; rather, it only shifts the starting position and arc geometry while preserving the feasibility of the maneuver.

Keywords: parallel parking, path planning, lateral offset, Δy, simulation, Ackermann steering, bicycle model.

1 Introduction

Parallel parking remains one of the most challenging driving maneuvers for both human drivers and autonomous systems. The geometric complexity arises from the need to navigate a vehicle into a constrained space using a series of curved arcs dictated by the vehicle's minimum turning radius. Wahab et al. [1] developed a compact mathematical model for a 1:10 scale prototype vehicle that successfully reduced the required parking space to just 1.33 times the vehicle length — an improvement over prior approaches such as those of Jiang and Seneviratne [2] and Gupta et al. [3].

A key parameter in their formulation is Δy, illustrated in Figure 1. It is defined as the lateral gap between the near side of the approaching vehicle and the near edge of the adjacent parked car, i.e., the boundary of the open parking space. As shown in the figure, a vehicle travelling parallel to the parked cars does not typically drive flush against them; Δy captures that stand-off distance, which can be measured in real-time by a side-facing ultrasonic sensor mounted on the vehicle. A larger Δy means the vehicle is positioned farther from the parked cars, and a smaller Δy means it is closer. Intuitively, one might expect that a larger Δy would require more room to maneuver, or that a smaller Δy might cause the vehicle to collide with the parked cars. In this paper, a parallel parking simulation system was developed to test that intuition.

In this simulation study, Δy is parameterized across a wide range and the two-arc parking path is observed to recalculate cleanly in every case. This is argued to be a direct consequence of the closed-form geometric solution: Δy feeds directly into the derived geometric quantities (detailed in Section 3), and all subsequent quantities adjust continuously and smoothly.

David C. Wyld et al. (Eds): BMLI, CACIT, EDUAIT, InWES – 2026
pp. 81-93, 2026. CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.161807

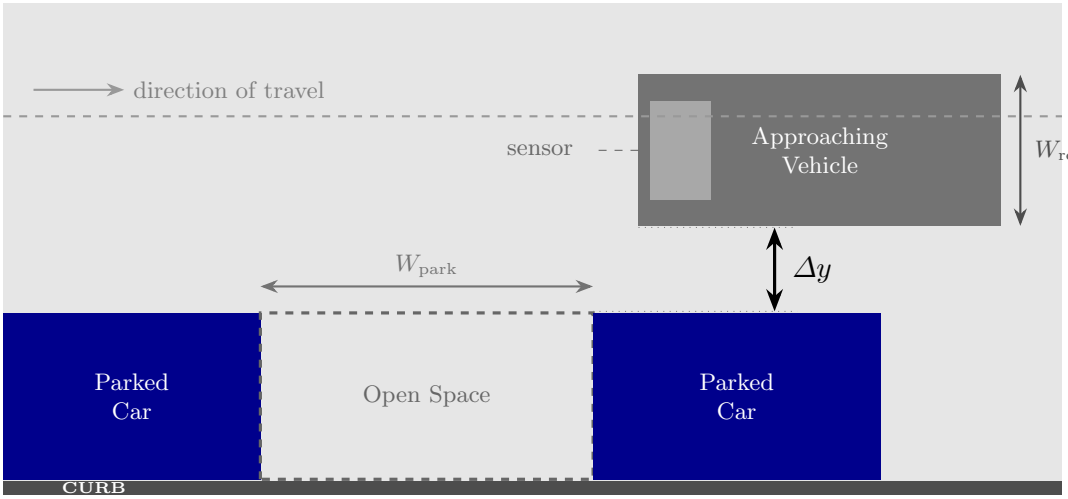

Fig. 1. Top-down schematic defining Δy.

2 Background and Related Work

Automated parallel parking systems can be categorized broadly into sensor-based maneuver (SBM) approaches and path-planning approaches. SBM methods use predefined sensor thresholds to trigger steering actions reactively [1, **?**]. Path-planning approaches compute an explicit geometric or polynomial trajectory prior to execution [4, **?**].

The present work is grounded in the SBM / geometric tradition. Wahab et al. [1] synthesize Ackermann steering geometry with a straightforward two-arc maneuver derived from circle geometry, yielding simple closed-form equations implementable on an Arduino MEGA 2560 microcontroller. Their prototype uses four HC-SR04 ultrasonic sensors to detect the parking space and measure relevant distances, including the lateral offset Δy from the parked car beside the space.

2.1 Assumptions of the Wahab et al. Model

The geometric model rests on three principal assumptions that are carried forward throughout this study:

1. **No-slip condition.** The wheels are assumed to roll without lateral slip. This is the standard kinematic (as opposed to dynamic) assumption and holds reasonably well at the low speeds typical of parking maneuvers.
2. **Perfect steering.** The steering angle — defined as the angle of the front wheels relative to the vehicle's longitudinal axis, with a maximum value $\delta_{\max}$ determined by the vehicle's mechanical steering limits — is assumed to be set instantaneously and held exactly at $\pm\delta_{\max}$ throughout each arc phase, with no steering lag or mechanical backlash.
3. **Flat ground.** The road surface is assumed to be perfectly level and horizontal, so that three-dimensional effects such as cross-slope or camber can be neglected.

Under these assumptions, the vehicle trajectory reduces to two tangentially joined circular arcs, which is the basis of the closed-form solution presented in Section 3.

Gupta et al. [3] similarly employed a four-wheeled Ackermann steered robot and derived

arc-based paths. Zhang et al. [4] used fifth-degree polynomial trajectories for smoother curvature profiles but required more computational overhead. For the purposes of this study, the simpler two-arc circular model of [1] is used, as its closed-form structure makes the role of each parameter — including Δy — analytically transparent.

3 Mathematical Model

A complete understanding of the parallel parking geometry is essential before proceeding. In the two-arc model, the vehicle begins travelling parallel to the curb, reverses while turning hard in one direction (Phase 1), then reverses while turning hard in the opposite direction (Phase 2), ending up aligned inside the parking space. The two arcs meet at a single inflection point where the steering direction switches. Figure 2 shows the full geometric arrangement including the two arc centres, the start point, turn point, and end point, as well as the key dimensions Δy, a, c, y_1, x_1, and x_s.

The key issues in the Wahab et al. framework are: (i) selecting the correct minimum turning radius $R_{\min}$ — the tightest circle the vehicle can follow at maximum steering lock, a property fixed by the vehicle's geometry — (ii) computing the sweep angle θ that ensures the vehicle clears all obstacles, and (iii) determining the minimum required starting offset x_s ahead of the space.

Table 1 summarizes the fixed vehicle and space parameters used throughout this study. These values are taken directly from Table 2 of Wahab et al. [1] and apply to their 1:10 scale prototype vehicle. All geometric quantities derived in this paper (a, c, y_1, θ, x_1, x_s) are computed from these same parameters, so the results here are directly comparable to the results reported by Wahab et al.

Table 1. Fixed vehicle and parking space parameters (from Table 2 of [1]).

Symbol	Description	Value
W_{park}	Width of empty parking space	220 mm
W_{robot}	Width of small mobile vehicle	200 mm
m	Safe distance, rear obstacle to vehicle	30 mm
n	Length from center/rear axle to rear	40 mm
L	Wheelbase (rear to front axle)	260 mm
D	Front bumper to rear axle	340 mm
$\delta_{\max}$	Maximum steering angle	41.3°

3.1 Minimum Turning Radius

The minimum turning radius $R_{\min}$ is a fundamental property of any steered vehicle: it is the radius of the tightest circular path the vehicle can follow, achieved when the front wheels are turned to their maximum angle $\delta_{\max}$. It is computed from $\delta_{\max}$ and the wheelbase L as:

$$R_{\min} = \frac{L}{\tan(\delta_{\max})} \approx 296 \text{ mm} \tag{1}$$

This follows from Ackermann steering geometry: when the front wheels are turned to their maximum angle, the instantaneous centre of curvature lies at a perpendicular

distance $R_{\min}$ from the rear axle (see Figure 2). A smaller wheelbase L or a larger maximum steering angle both reduce $R_{\min}$, improving manoeuvrability in tight spaces. For the 1:10 scale vehicle used here, $\delta_{\max} = 41.3°$ was determined experimentally from the physical steering mechanism of the prototype [1].

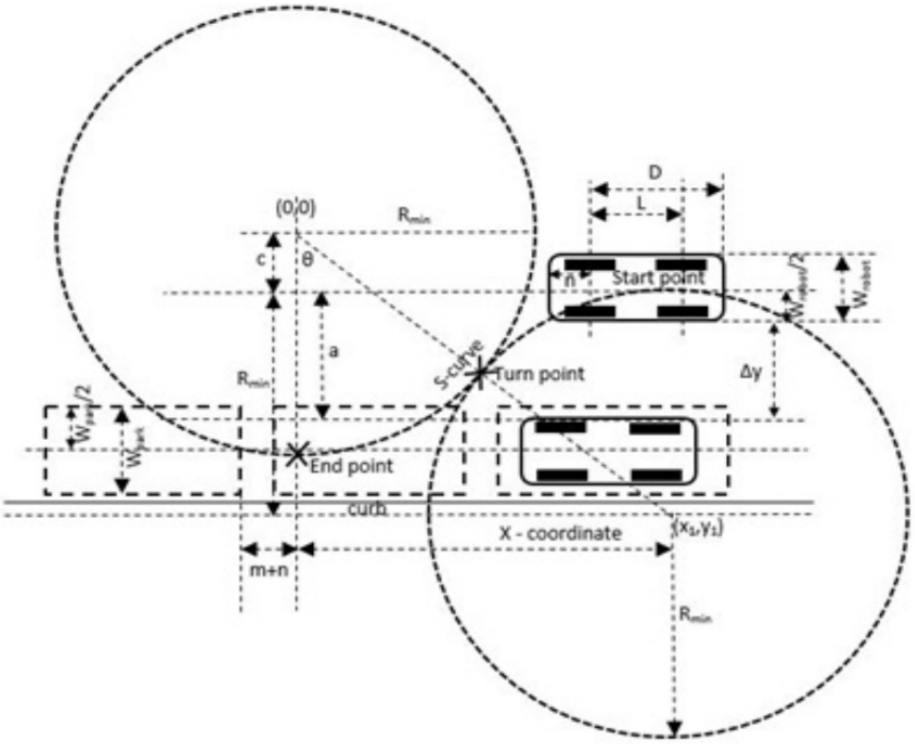

Fig. 2. Path-planning geometry for the two-arc parallel parking maneuver, adapted from Wahab et al. [1] (their Figure 3). The vehicle starts at the *Start point* (upper right), reverses along arc 1 to the *Turn point*, then continues along arc 2 to the *End point* inside the parking space. The two dashed circles are the turning circles of radius $R_{\min}$. Key dimensions a, c, y_1, x_1, and Δy correspond directly to Equations 2–6.

3.2 Geometric Relationships

The core geometry follows equations (5) through (12) of Wahab et al. [1], and the quantities below can be traced directly to the labelled dimensions in Figure 2. Beginning from the lateral offset parameter Δy:

$$a = \Delta y + \frac{W_{\text{robot}}}{2} \tag{2}$$

The quantity a is the lateral distance from the centreline of the vehicle to the near edge of the parking space. It combines Δy (measured from the vehicle's side to the parked car edge) with half the vehicle width, translating the sensor reading into a distance measured from the vehicle's geometric centre.

$$c = R_{\min} - a - \frac{W_{\text{park}}}{2} \tag{3}$$

The quantity c is the lateral distance from the far edge of the parking space to the centre of the lower (first) turning circle. When $c > 0$, the turning circle centre lies outside the parking lane. When $c < 0$ (which occurs for larger values of Δy, as seen in Table 2), the turning circle centre is located inside the lateral boundary of the parking lane — meaning the arc sweeps through a wider angle to compensate for the greater initial offset.

$$y_1 = R_{\min} + c = 2R_{\min} - \left(\Delta y + \frac{W_{\text{robot}}}{2} + \frac{W_{\text{park}}}{2} \right) \tag{4}$$

The quantity y_1 is the lateral distance between the centres of the two turning circles. As Δy increases, y_1 decreases, which in turn increases the sweep angle θ in the next equation.

$$\theta = \cos^{-1}\left(\frac{y_1}{2R_{\min}} \right) \tag{5}$$

The angle θ is the arc sweep angle subtended by each of the two phases. It is the angle through which the vehicle's heading rotates during Phase 1 (and again during Phase 2). A larger θ means the vehicle must follow a longer arc before switching steering direction.

$$x_1 = 2R_{\min} \sin\theta \tag{6}$$

The quantity x_1 is the horizontal distance between the two turning circle centres, which determines how far forward the vehicle must be positioned at the start. The starting position of the front axle is then:

$$x_s = x_1 + m + n \tag{7}$$

where m is a safe clearance distance from the rear obstacle and n is the distance from the rear axle to the rear bumper. Each arc has length:

$$S = R_{\min} \cdot \theta \tag{8}$$

The two arcs are symmetric ($S_1 = S_2$), and the maneuver is completed in exactly two phases of equal angular sweep θ, regardless of the specific value of Δy. It should be noted that θ (the arc sweep angle), δ (the steering angle at the front wheels), and ψ (the vehicle heading angle) are distinct quantities and are never used interchangeably in this analysis.

3.3 Coordinate System and Geometric Layout

Following Figure 3 of [1] (reproduced here as Figure 2), the origin is placed at the centre of the upper turning circle, which coincides with the end-point target for the front axle. The $+x$ direction points toward the starting position (away from the space), and $+y$ points toward the curb. This coordinate system ties directly to the dimensions in Table 1 and to the equations in Section 3.2.

4 Simulation Implementation

4.1 Overview

The simulation was developed in a Java-based environment (Processing 3) to allow for high-fidelity kinematic modelling and real-time geometric validation. The system renders a top-down view of the road scene — including the parked cars, the open parking space, the vehicle, and its turning circles — and animates the full two-phase parking maneuver. At

any point the user can adjust Δy interactively and immediately observe how the geometry and trajectory update. Figure 3 shows a screenshot of the simulation environment at $\Delta y = 60$ mm. The full source code is provided in the Appendix.

4.2 Bicycle Model

The vehicle dynamics follow the standard kinematic bicycle model. Under the no-slip and flat-ground assumptions, the state of the vehicle is fully described by the rear-axle position (x, y) and heading angle ψ (the direction the vehicle is pointing). At each simulation timestep these evolve as:

$$\dot{\psi} = \frac{v \tan(\delta)}{L}, \qquad \dot{x} = v \cos\psi, \qquad \dot{y} = v \sin\psi \tag{9}$$

where v is the vehicle speed (negative when reversing) and δ is the front-wheel steering angle. The controller sets $\delta = +\delta_{\max}$ (full lock in one direction) throughout Phase 1 and $\delta = -\delta_{\max}$ (full lock in the opposite direction) throughout Phase 2. Each phase ends when the total change in heading ψ equals the target sweep angle θ computed from the current Δy, at which point the vehicle has traced a circular arc of radius $R_{\min}$ and transitions to the next phase.

4.3 Interactive Parameter Control

The simulation allows the user to adjust Δy in 5 mm increments using keyboard input. After each adjustment the scene geometry is fully recomputed — the arc sweep angle θ, the starting position x_s, and the arc length S all update automatically according to Equations 2–8. The vehicle is repositioned at the new starting location and the maneuver can be replayed immediately. The simulation was tested with Δy ranging from 5 mm to approximately 500 mm.

5 Results and Discussion

5.1 Maneuver Feasibility Across Δy Values

Across all tested values of Δy from 5 mm to 500 mm, the simulated vehicle successfully completed the two-arc parallel parking maneuver without collision in every trial. The vehicle entered the parking space and the maneuver converged to the target pose in all cases. Figure 3 shows the simulation environment at $\Delta y = 60$ mm, illustrating the rear-axle path and the vehicle at its final parked position.

Table 2 shows computed geometric values for a representative selection of Δy settings.

Fig. 3. Simulation environment at $\Delta y = 60$ mm. The trail visible behind the vehicle is the rear-axle trajectory, composed of two smooth arcs. The information panel (top right) confirms the computed parameters: $a = 160$ mm, $c = 26.0$ mm, $y_1 = 321$ mm, $\theta = 57.1°$, $x_s = 496$ mm, arc $S = 294$ mm. The circular overlays show the two turning circles whose geometry is computed by Equations 2–8.

Table 2. Computed geometry for selected Δy values. All lengths in mm, angle in degrees.

Δy (mm)	a (mm)	c (mm)	y_1 (mm)	θ (deg)	x_s (mm)
20	120	66	362	52.4	469
40	140	46	342	54.7	481
60	160	26	322	57.1	496
80	180	6	302	59.4	510
100	200	−14	282	61.8	526
120	220	−34	262	64.2	535

5.2 Effect of Δy on Geometric Parameters

From Equations 2–5, the dependence of θ on Δy is monotonically increasing: as Δy grows, y_1 decreases, causing the sweep angle $\theta = \cos^{-1}(y_1/2R_{\min})$ to increase. This means the vehicle must follow a longer arc to complete the maneuver. Correspondingly, x_s increases, so the vehicle must begin the maneuver from a position farther ahead of the parking space.

Theoretical Operational Limit of the Wahab Model The theoretical operational limit of the Wahab model is defined by the condition that $y_1 \leq 2R_{\min}$, which ensures the argument of $\cos^{-1}$ in Equation 5 remains within $[-1, 1]$. This reduces to:

$$\Delta y \leq R_{\min} - \frac{W_{\text{robot}} + W_{\text{park}}}{2} \tag{10}$$

For the paper's parameters, this upper bound evaluates to $296 - 210 = 86$ mm. Beyond this value the lateral clearance c (the distance from the far edge of the parking space to the lower turning circle centre) becomes negative. The physical significance is that the lower turning circle centre has crossed inside the lateral boundary of the parking lane, implying a wider arc sweep is needed to complete the maneuver. Despite this geometric transition, the two-arc trajectory remains mathematically valid and the simulation continued to park successfully across the full tested range.

5.3 Interpretation

The key insight is that Δy is not a free disturbance that disrupts a fixed trajectory. Rather, it is an *input* to the path-planning module. For every value of Δy, the model generates a new, valid two-arc trajectory. The parking system is therefore robust to variations in Δy in the sense that:

1. The sensor measures Δy prior to the maneuver.
2. The geometric equations compute a customised sweep angle θ and starting position x_s for that Δy.
3. The controller executes the resulting arcs.
4. The maneuver converges to the target pose.

This is directly visible in the simulation: at every tested Δy, the rear-axle trail traces two smooth arcs that deposit the vehicle neatly inside the parking space.

6 Conclusion

Through simulation of the geometric model from Wahab et al. [1], it is demonstrated that the initial lateral offset Δy of the vehicle from the parking space does not prevent

successful parallel parking. Rather than degrading maneuver feasibility, varying Δy simply produces a different valid two-arc path with adjusted sweep angle θ and starting position x_s. The simulation software confirmed complete, collision-free parking across all tested Δy values from 5 mm to 500 mm, with the maneuver converging to the target pose in every case.

These findings have practical implications for autonomous parking system design: precise control of lateral lane position prior to initiating parking is not strictly necessary. Provided the sensor measurement of Δy is accurate and the parking space is geometrically sufficient, the path planner can accommodate a wide range of approach offsets. Future work could extend this analysis to include uncertainty in Δy measurement and its propagation through the geometric equations.

References

1. M. F. Wahab, A. L. Moe, A. Abu, Z. Yaacob, and A. Legowo, "Development of Automated Parallel Parking System in Small Mobile Vehicle," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 16, pp. 7107–7112, Sept. 2015.
2. K. Jiang and L. D. Seneviratne, "A Sensor Guided Autonomous Parking System for Nonholonomic Mobile Robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, Detroit, MI, May 1999, pp. 311–316.
3. A. Gupta, R. Divekar, and M. Agrawal, "Autonomous Parallel Parking System for Ackermann Steering Four Wheelers," in *Proc. Computational Intelligence and Computing Research*, 2010, pp. 1–2.
4. S. Zhang, M. Simkani, and H. M. Zadeh, "Automatic Vehicle Parallel Parking Design Using Fifth Degree Polynomial Path Planning," in *Proc. IEEE Vehicular Technology Conference (VTC Fall)*, 2011, pp. 1–4.
5. H. Vézien, S. Glaser, N. M. E., and S. Mammar, "Automatic Parallel Parking in Tiny Spots: Path Planning and Control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, Feb. 2015.

A Simulation Source Code

The simulation consists of two Processing 3 source files. The complete listings are included below for reproducibility.

`ParallelParking.pde`

Listing 1.1. ParallelParking.pde — main sketch

```
// ============================================================
// Parallel Parking Simulation  -  ARPN 2015 paper
//
// Paper Table-2 (mm):  W_park=220  W_robot=200  m=30  n=40
//                      L=260  D=340
//
// Coordinate system (Fig. 3):
//   Origin = centre of the UPPER circle = same x as the end-point.
//   +x -> right,  +y -> DOWN (screen).
//
//   End-point  (front axle target):  (0,  c + a + W_robot/2)
//     where  a = delta_y + W_robot/2
//            c = R_min - a - W_park/2
//
//   Lower circle centre:  (x1, y1)
//     y1    = R_min + c
//     theta = arccos( y1 / (2*R_min) )
```

```
//      x1    = 2*R_min*sin(theta)
//
//   Starting front-axle x:  x_s = x1   (paper eq.)
//   Arc length of each curve: S = R_min * theta
//
// Controls:
//   SPACE : start / re-run
//   R     : reset
//   + / - : change delta_y by 5 mm
// ============================================================

final float S_PX = 0.38;             // px per mm  (fits 1100x600)

// Paper values in px
final float W_park_px  = 220 * S_PX;
final float W_robot_px = 200 * S_PX;
final float m_px       =  30 * S_PX;
final float n_px       =  40 * S_PX;
final float L_px       = 260 * S_PX;
final float D_px       = 340 * S_PX;
final float kF_px      = D_px - L_px;   // front overhang 80 mm
final float carLen_px  = D_px + n_px;   // total 380 mm

// max steer -> R_min = L / tan(d) approx 600 mm
final float MAX_STEER  = radians(23.5);

// Parked car display rectangles
final float PARKED_LEN = 450 * S_PX;
final float PARKED_WID = W_park_px;

// ---- Adjustable experiment variable -----------------------
float delta_y_px = 60 * S_PX;

// ---- Scene globals ----------------------------------------
float curbY, spaceLeftX, spaceLen;

SimCar car;
boolean running  = false;
boolean finished = false;
ArrayList<PVector> trail = new ArrayList<PVector>();

PFont mono;

// ============================================================
void setup() {
  size(1100, 600);
  smooth(4);
  mono = createFont("Courier New", 12);
  textFont(mono);
  initScene();
}

void initScene() {
  running  = false;
  finished = false;
  trail.clear();

  curbY    = height - 120;
  spaceLen = carLen_px * 1.5;

  float R  = L_px / tan(MAX_STEER);
```

```
  float a  = delta_y_px + W_robot_px / 2.0;
  float c  = R - a - W_park_px / 2.0;
  float y1 = R + c;
  float th = acos(constrain(y1 / (2.0 * R), -1, 1));
  float x1 = 2.0 * R * sin(th);
  float xs = x1;

  float originScreenX = width * 0.28;
  float originScreenY = curbY - W_park_px - delta_y_px - W_robot_px / 2.0 - c;

  spaceLeftX = originScreenX - (m_px + n_px);

  float startFAX = originScreenX + xs;
  float startFAY = originScreenY+c;

  float startX = startFAX - L_px;
  float startY = startFAY;

  car = new SimCar(startX, startY, 0.0,
                   L_px, kF_px, n_px, W_robot_px, MAX_STEER);
}

void draw() {
  background(205, 210, 213);
  drawRoad();
  drawParkedCars();
  drawAnnotations();
  drawTrail();

  if (running && !finished) {
    car.stepAutoPark();
    trail.add(new PVector(car.x, car.y));
    if (car.phase == 3) finished = true;
  }

  car.draw();
  drawHUD();
}

void keyPressed() {
  if (key == ' ') {
    if (!running) {
      car.startPark();
      running = true; finished = false; trail.clear();
    } else if (finished) {
      initScene();
      car.startPark();
      running = true; finished = false; trail.clear();
    }
  }
  if (key == 'r' || key == 'R') initScene();
  if (key == '+' || key == '=') { delta_y_px = min(delta_y_px + 5 * S_PX, 500 * S_PX); initScene(); }
  if (key == '-' || key == '_') { delta_y_px = max(delta_y_px - 5 * S_PX, 5 * S_PX); initScene(); }
}

String mm(float px) { return "" + int(px / S_PX); }

void dashedLine(float x1, float y1, float x2, float y2, float d, float g) {
```

```
  float dx = x2 - x1, dy = y2 - y1, len = sqrt(dx * dx + dy * dy);
  float nx = dx / len, ny = dy / len, t = 0;
  while (t < len) {
    float t2 = min(t + d, len);
    line(x1 + nx * t, y1 + ny * t, x1 + nx * t2, y1 + ny * t2);
    t += d + g;
  }
}
```

SimCar.pde

Listing 1.2. SimCar.pde — vehicle class and controller

```
// ============================================================
// SimCar  -  bicycle model, rear-axle reference
// heading 0 = pointing RIGHT; y-down screen coords
// ============================================================
class SimCar {
  float x, y;           // rear axle (px)
  float heading;        // radians
  float steerAngle;
  float speed;

  float L, kF, kR, W, maxSteer;

  // phase: 0=idle  1=reverse+steer-right  2=reverse+steer-left  3=done
  int   phase = 0;
  float accum = 0;
  float tgt1  = 0;
  float tgt2  = 0;

  final float SPD = 1.4;   // px per frame

  SimCar(float x, float y, float hdg,
         float L, float kF, float kR, float W, float maxSteer) {
    this.x=x; this.y=y; this.heading=hdg;
    this.L=L; this.kF=kF; this.kR=kR; this.W=W; this.maxSteer=maxSteer;
    steerAngle=0; speed=0;
    calcTargets();
  }

  void calcTargets() {
    float R  = L / tan(maxSteer);
    float a  = delta_y_px + W / 2.0;
    float c  = R - a - W_park_px / 2.0;
    float y1 = R + c;
    float th = acos(constrain(y1 / (2.0 * R), -1, 1));
    tgt1 = th;
    tgt2 = th;
  }

  void startPark() {
    calcTargets();
    accum = 0;
    phase = 1;
    speed = 0;
  }

  void stepAutoPark() {
    if (phase==0||phase==3) { speed=0; steerAngle=0; return; }
```

```
48
49      if      (phase==1) { speed=-SPD; steerAngle=+maxSteer; }
50      else if (phase==2) { speed=-SPD; steerAngle=-maxSteer; }
51
52      float hdDot = speed * tan(steerAngle) / L;
53      accum += abs(hdDot);
54      heading += hdDot;
55      x += speed * cos(heading);
56      y += speed * sin(heading);
57
58      if (phase==1 && accum>=tgt1) { accum=0; phase=2; }
59      else if (phase==2 && accum>=tgt2) { accum=0; phase=3; speed=0;
            steerAngle=0; }
60    }
61
62    float totalLen() { return kR+L+kF; }
63
64    void draw() {
65      if (phase==1||phase==2) drawCircles();
66
67      pushMatrix();
68      float cx = x + (kR+L/2.0)*cos(heading);
69      float cy = y + (kR+L/2.0)*sin(heading);
70      translate(cx,cy); rotate(heading);
71
72      stroke(0,0,0,55); strokeWeight(1);
73      fill(215,75,55);
74      rectMode(CENTER);
75      rect(0,0,totalLen(),W);
76
77      fill(238,195,195);
78      rect(L/2.0 - kF*0.3, 0, totalLen()*0.54, W*0.6);
79
80      fill(255,155,0); noStroke();
81      rect(totalLen()/2.0-3, 0, 5, W);
82
83      fill(55,115,215);
84      rect(-totalLen()/2.0+3, 0, 5, W);
85
86      drawWheels();
87      popMatrix();
88
89      fill(255,220,0); noStroke();
90      ellipseMode(CENTER); ellipse(x,y,6,6);
91    }
92
93    void drawWheels() {
94      float wL=16, wW=6;
95      float rOff=-(L/2.0), fOff=L/2.0;
96      fill(35); noStroke();
97      pushMatrix(); translate(rOff,-W/2.0); rect(0,0,wL,wW); popMatrix();
98      pushMatrix(); translate(rOff, W/2.0); rect(0,0,wL,wW); popMatrix();
99      pushMatrix(); translate(fOff,-W/2.0); rotate(steerAngle); rect(0,0,wL,
            wW); popMatrix();
100     pushMatrix(); translate(fOff, W/2.0); rotate(steerAngle); rect(0,0,wL,
            wW); popMatrix();
101   }
102
103   void drawCircles() {
104     float R = L/tan(maxSteer);
105     float c1x = x - R*sin(heading),  c1y = y + R*cos(heading);
```

```
106     float c2x = x + R*sin(heading),  c2y = y - R*cos(heading);
107
108     noFill(); strokeWeight(1); ellipseMode(RADIUS);
109     if (phase==1) {
110       stroke(100,175,255,110); ellipse(c1x,c1y,R,R);
111       fill(100,175,255,160); noStroke(); ellipse(c1x,c1y,5,5);
112     } else {
113       stroke(175,100,255,110); ellipse(c2x,c2y,R,R);
114       fill(175,100,255,160); noStroke(); ellipse(c2x,c2y,5,5);
115     }
116   }
117 }
```

Authors

Kevin Luo is a student at Sunny Hills High School, Fullerton, CA. His research interests include autonomous systems, robotics, and geometric path planning.

Hsinghan Meng is affiliated with SAT Professionals, Diamond Bar, CA. His research interests include mathematical modelling, simulation, and autonomous vehicle systems.

©2026 By AIRCC Publishing Corporation . This article is published under the Creative Commons Attribution (CC BY) license.