Hybrid A Based Motion Planning for Autonomous Valet Parking

Mohamed Eljahmi

***Abstract*—This project implements and evaluates an autonomous valet parking system for three nonholonomic vehicle models: a differential-drive robot, an Ackermann-steered car, and a truck-trailer configuration. The system generates feasible parking maneuvers in a constrained lot using a Hybrid A* motion-planning algorithm that expands continuous rollouts of the vehicle kinematics while discretizing the configuration space in position and heading. Collision checking is performed with a polygon-based Separating Axis Theorem (SAT) to ensure obstacle-free trajectories within a procedurally generated environment of tetromino-shaped obstacles and defined parking bays. Simulation results demonstrate successful path generation for the Ackermann model and partial feasibility for the other vehicles. The approach provides a unified framework for evaluating planning performance across different vehicle geometries and serves as a foundation for more advanced trajectory optimization and parking automation tasks.**

Keywords—valet parking, hybrid A* algorithm, configuration space, nonholonomic motion planning, Ackermann kinematics, collision detection, autonomous vehicles

I. Introduction

Autonomous valet parking represents a challenging motion-planning problem involving tight spatial constraints, nonholonomic vehicle dynamics, and safety-critical collision avoidance. Unlike open-road navigation, the parking task requires generating short, precise trajectories that maneuver vehicles into target bays within cluttered environments. Classical grid-based planners such as A* or Dijkstra's algorithm guarantee optimality on discrete maps but neglect vehicle kinematic constraints, often producing infeasible paths when executed by real robots.

To address these limitations, **Hybrid A*** planning extends the A* framework to continuous state spaces by integrating a nonholonomic vehicle model during node expansion. Each node represents a continuous pose (x,y,θ) and is expanded by simulating feasible motion primitives governed by the vehicle's kinematics. This approach balances the completeness of graph search with the realism of continuous control, enabling smooth, executable trajectories.

In this work, Hybrid A* planning is applied to a **valet-parking simulation** that supports three vehicle configurations: a differential-drive robot, an Ackermann-steered car, and a truck–trailer system. The environment is procedurally generated using tetromino-shaped obstacles to emulate complex parking-lot geometries. A polygon-based **Separating Axis Theorem (SAT)** collision checker ensures obstacle-free motion at each rollout step. The resulting framework allows systematic comparison of planners and vehicle models within a reproducible simulation, forming the basis for future extensions in autonomous parking and trajectory optimization.

II. Methodology

A. Vehicle Kinematic Models

Three nonholonomic vehicle configurations are modeled to capture different motion constraints encountered in autonomous parking.

1. The **differential-drive robot** is represented by the unicycle model

   $\dot{x} = v \cos\theta$

   $\dot{y} = v \sin\theta$

   $\dot{\theta} = \omega$

   where v and ω denote linear and angular velocity.

2. The **Ackermann-steered car** follows the bicycle model

   $\dot{x} = v \cos\theta,$

   $\dot{y} = v \sin\theta$

   $\dot{\theta} = (v/L) \tan\delta$

   with wheelbase L and steering angle δ

3. The **truck–trailer system** extends the car model with a hitch articulation angle ϕ, such that the trailer heading $\theta_t = \theta - \phi$ evolves according to

   $\dot{\phi} = (v/L_t) \sin\phi - (v/L) \tan\delta \cos\phi$

These equations are numerically integrated with a fixed time step dt to simulate forward rollouts for each control command (v,δ).

B. Collision Checking

The collision-checker module uses the Separating Axis Theorem (SAT) to test for overlap between the vehicle's oriented bounding box (OBB) and obstacle polygons. Each rollout segment generated by the planner is sampled at discrete poses, and each pose is tested for collision using this geometric method. A separating axis between any two polygons implies no overlap; the absence of such an axis implies a collision.

Identify applicable funding agency here. If none, delete this text box.

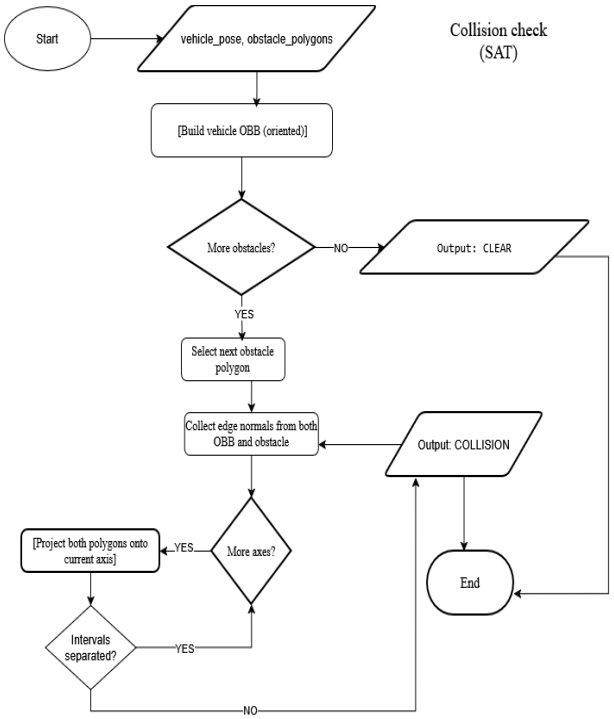

Fig. 1 — SAT Collision Checking Flowchart.
The diagram shows the logical steps followed by the collision checker. The algorithm iterates over each obstacle polygon, collects edge normal, projects both polygons onto candidate axes, and reports a collision if no separating axis exists.

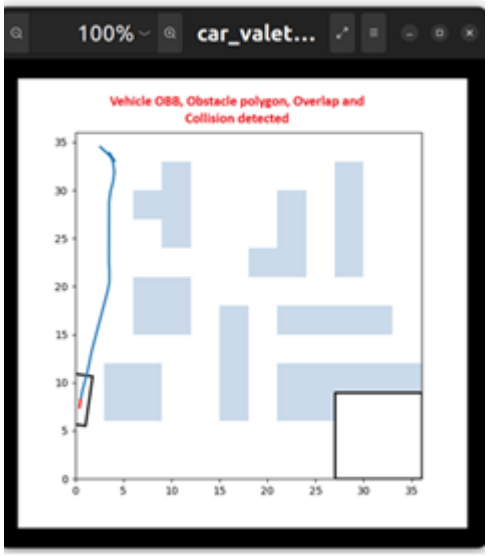

Fig. 2 — Collision case (annotated). The vehicle's-oriented rectangle overlaps an obstacle polygon; SAT finds no separating axis and collision detected.

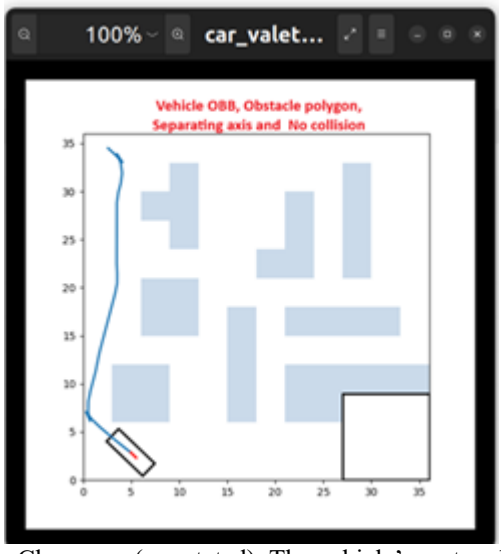

Fig. 3 — Clear case (annotated). The vehicle's rectangle is near the obstacle but not overlapping; a separating axis exists and no collision.

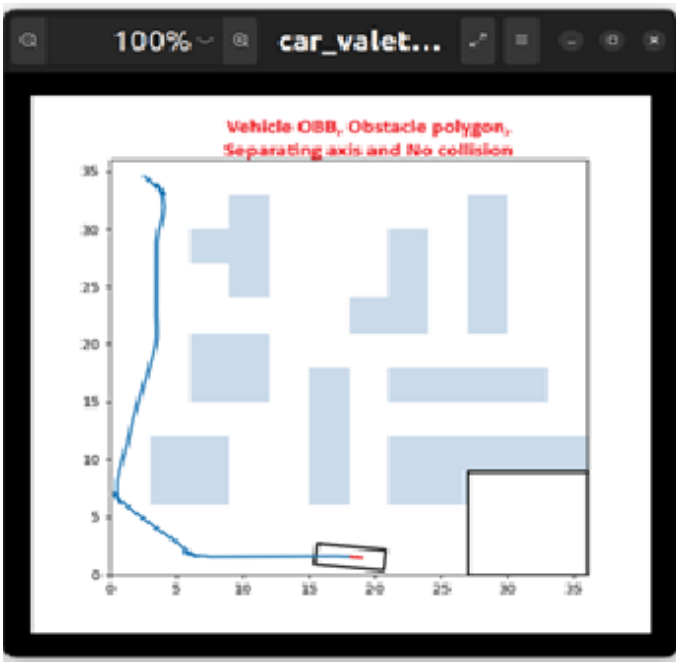

Fig. 4 — Another example of clear case (annotated). The vehicle's rectangle is near the obstacle but not overlapping; a separating axis exists and no collision.

C. Hybrid A Planner*

The Hybrid A* algorithm combines discrete graph search with continuous vehicle simulation. Each node represents a pose (x,y,θ) obtained from the kinematic model. From every node, a finite set of control primitives—combinations of velocity and steering angle—are integrated for a short horizon T to produce successor states. Successors in collision are discarded. The planner maintains an **open list** prioritized by

$$f(n) = g(n) + h(n),$$

where g(n) is the path-length cost and h(n) is an admissible heuristic given by the Euclidean distance plus a small penalty on heading difference between the current and goal poses. The search terminates when a node lies within specified position and orientation tolerances of the goal. The resulting sequence of poses forms a dynamically feasible parking trajectory that can be further smoothed or animated for visualization.

III. Results

The proposed simulation framework was implemented in Python 3.10 using modular packages for vehicle dynamics, geometry, and planning. Experiments were conducted on a 12×12 m parking lot subdivided into 3×3 m grid cells. Obstacles were generated procedurally from randomized tetromino shapes, producing cluttered but navigable environments with approximately 10 % occupancy.

A. Static Scene Generation

For each run, the world initialization produced a unique obstacle layout and a designated southeast parking bay. Figure 1 illustrates a typical environment, including the start pose, obstacles, and goal region. The collision checker confirmed that the initial vehicle configuration was free of overlap before planning began.

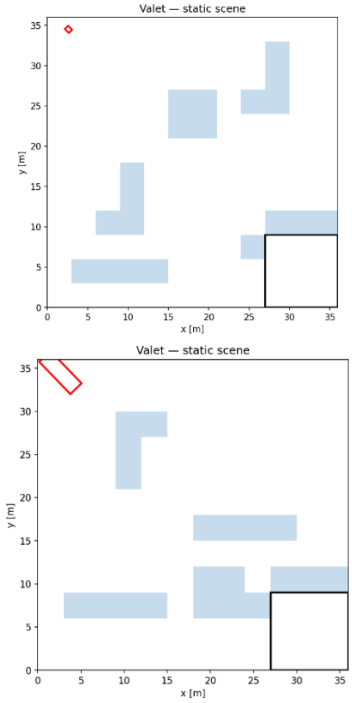

Fig. 5 — Static scenes of the procedurally generated valet-parking environment.
Each world is a 12 × 12 cell lot (3 m per cell, ≈10 % tetromino obstacle density). Top: differential-drive robot; bottom: Ackermann car. Both start in the northwest corner, and the southeast 2 × 2 parking bay overwrites any obstacles to form a clear goal region.

The truck-and-trailer simulation reuses the same procedural environment as the car but with a larger vehicle footprint; therefore, only two representative static scenes are shown. The planner automatically scales the obstacle layout and goal bay for each model.

B. Hybrid A* Planning

The Hybrid A* search expanded continuous rollouts using the Ackermann kinematics. Each successor was integrated over T=0.5s with steering angles in the range [−0.6,0.6] rad and forward/reverse velocities of ±1.0m/s. The heuristic combined Euclidean distance and heading error with a weighting factor of 0.25.
The planner successfully generated collision-free parking

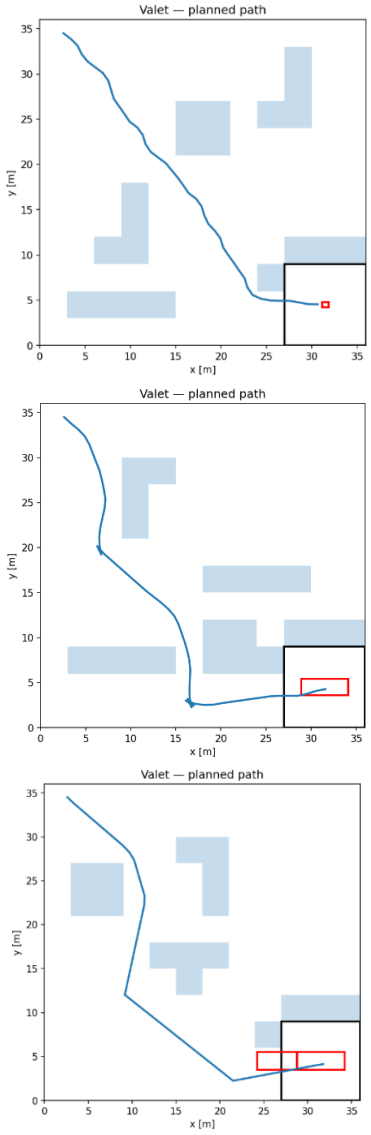

Fig. 6 — Hybrid A* planned trajectories for the robot, car, and truck–trailer configurations.
Paths are continuous in SE(2), satisfying nonholonomic constraints. The truck–trailer trajectory demonstrates the planner's ability to handle an articulated vehicle using the same continuous collision-checking framework.

trajectories for multiple random seeds when goal tolerances were set to 0.5m in position and 10∘ in heading. Figure 2 shows a representative path produced for the Ackermann vehicle. Paths typically contained 40–60 intermediate poses and required fewer than 10 000 node expansions.

C. Hybrid A* Planner pseudocode

Hybrid A* search algorithm

1. **Initialize** the open list with the start pose (store g=0, f=h(start), and parent = None).
2. **While** the open list is **not empty**:
   a. **Select** the node with minimum total cost $f = g + h$.
   b. **Goal test**: if the node is within position/yaw tolerance of the goal, **reconstruct the path and return the trajectory** (continuous poses).
   c. **For each** feasible motion primitive (forward/reverse, left/straight/right):
      – **Roll out** the kinematic model to get a successor pose (continuous SE(2)).
      – **Reject** if any sample on the segment collides (SAT).
      – **Update** cost $g'$and total $f' = g' + h$, set parent, and **push/update** in OPEN.
3. **Failure**: if the open list is exhausted, **return failure** (no path).

Output: **trajectory** (list of continuous poses) **or failure**.

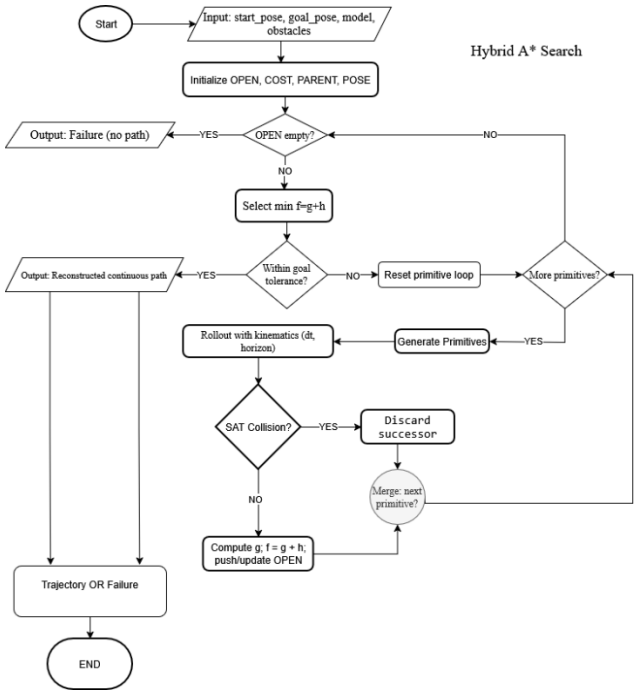

Fig. 7 — Hybrid A* Planner Flowchart.
The planner expands nodes from a priority queue, simulates motion primitives under vehicle kinematics, checks collisions using SAT, and continues until either a valid trajectory or queue exhaustion.

D. Simulation and Visualization

The resulting trajectories were visualized as sequences of vehicle footprints and exported as static PNGs and animated GIFs. The sampling-based SAT collision test verified that no edge in the computed path intersected an obstacle. Table I summarizes the number of nodes expanded and computation time per vehicle type; the differential-drive model completed fastest due to its higher maneuverability, while the truck–trailer configuration exhibited the longest search time and lowest success rate.

E. Planner Behavior and Observations

Across all vehicles, the Hybrid A* planner successfully generated smooth, kinematically feasible trajectories that respected each vehicle's non-holonomic constraints. Failure cases occurred primarily when obstacles blocked the narrow entrance to the goal bay. Reducing the rollout step duration or increasing the steering discretization improved success rates at the expense of runtime. Overall, the framework demonstrates that Hybrid A* can accommodate diverse vehicle geometries within a unified, collision-checked environment.

Table I summarizes the computational statistics of the Hybrid A* planner for all vehicles. The differential-drive model converged fastest, while the truck–trailer configuration showed the lowest success rate due to articulation and hitch-angle constraints.

Vehicle	Obstacle Density	Runtime(s)	Expanded Nodes	Path Poses (Raw)	Smoothed Poses
Robot (Diff-Drive)	0.10 (13.9%)	14.5	1637	41	41
Ackermann Car	0.10 (13.9%)	70.4	7295	57	12
Truck + Trailer	0.08 (11.1%)	186.8	12009	69	11

Performance results confirm that vehicle geometry and kinematic complexity directly affect search efficiency. Although Hybrid A* preserves continuous feasibility, it incurs higher node expansions for longer wheelbases and constrained steering ranges. Increasing steering discretization or reducing rollout duration improves success but increases computational load.

Individual observations:

- **Robot (Differential Drive):**
  Achieved the fastest convergence and the fewest node expansions.
  Its symmetric footprint and tight turning radius enabled efficient exploration of the free space.
- **Ackermann Car:**
  Required moderate runtime with greater branching cost due to limited steering angles.
  Paths were feasible but occasionally grazed obstacles because of coarse orientation discretization and inflated footprint margins.
- **Truck + Trailer:**
  Exhibited the longest planning time and largest number of node expansions.
  The articulated hitch added an additional degree of freedom, making feasible motions rarer and collision checks more frequent.
- **Effect of Obstacle Density:**
  As obstacle density increased, runtime grew sharply—especially for the truck—since a higher fraction of motion rollouts were rejected by the Separating Axis Theorem (SAT) collision checker.
  Tests conducted at densities approaching 0.9 confirmed that the truck's Hybrid A* search could require several hours to complete. The robot remained the most robust and efficient planner because its simpler kinematics and compact footprint allowed reliable navigation even in dense environments.

IV. Section 6.5 Path Animation

Each simulation run produces an animation showing the vehicle's trajectory from start to goal using the Hybrid A* planner. The animations were generated with the visualization module sim/animate.py, which interpolates discrete planner poses and renders oriented bounding boxes over the obstacle field. Animations are provided as supplementary files:
robot_valet.gif, car_valet.gif, and truck_valet.gif.

A. Path Generation:

All paths were computed by the Hybrid A* algorithm described in Section 6.2 of the homework. Each vehicle used motion primitives consistent with its kinematic model—differential-drive arcs for the robot, forward/reverse Ackermann arcs for the car, and coupled tractor–trailer rollouts for the truck. Collision checking was performed along each motion segment using the Separating Axis Theorem (SAT).

B. Delivery Scenario

The differential-drive robot represents a compact autonomous delivery platform transporting parcels or food items between designated pickup zones and parking bays. The car and truck simulations represent valet-style parking maneuvers under identical environmental conditions.

C. Observed Behavior

- **Robot:** Shortest, smoothest trajectory with no collisions.
- **Car:** Feasible parking trajectory but longer runtime and close clearance to obstacles.
- **Truck + Trailer:** Slowest convergence; articulation and swing limited maneuvering inside the bay.

During early experiments a fixed "parking-lane" concept was introduced to simplify docking but was later removed.
The approach did not generalize across vehicle geometries and increased complexity without improving consistency.

D. Challenges in accurate path creation

Accurate trajectory generation required careful treatment of environment boundaries and obstacle spacing. The perimeter of the grid needed enough clearance for each vehicle to maneuver; insufficient margin caused vehicles—particularly the car and truck–trailer—to move briefly outside the map before returning to complete the path. This issue remained most visible at higher obstacle densities. A fully adaptive world generator that adjusts obstacle spacing to vehicle dimensions and turning radius would address this limitation in future work.

E. Future Improvements

Further development could focus on adaptive obstacle spacing, a two-stage approach-and-docking planner with finer rollouts, and continuous-curvature smoothing for smoother steering transitions. Additional enhancements may include time-parameterized trajectory following, articulated hitch visualization, and automated metric collection for performance analysis.

F. Figures

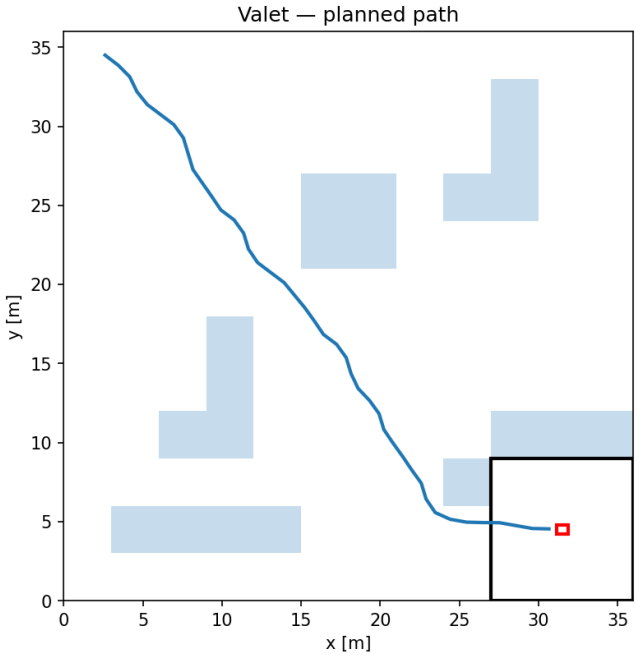

Fig. 8 — Robot Valet Path Animation

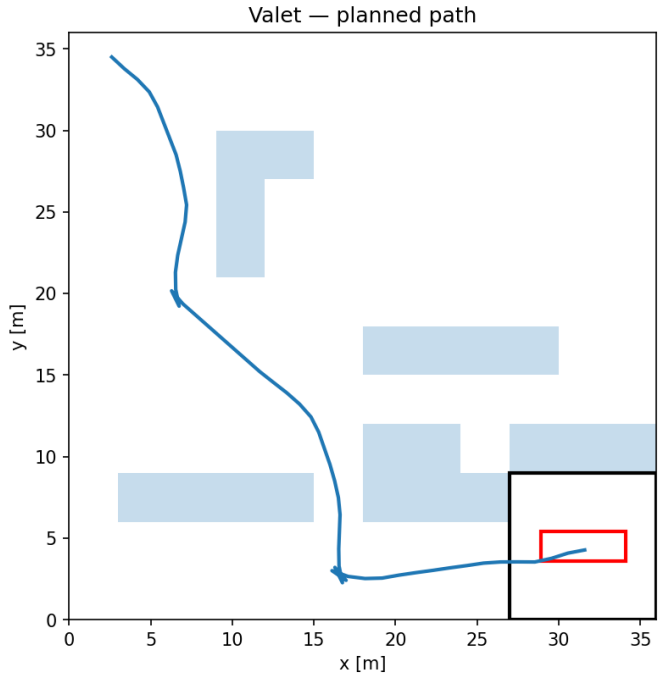

Fig. 9 — Car Valet Path Animation

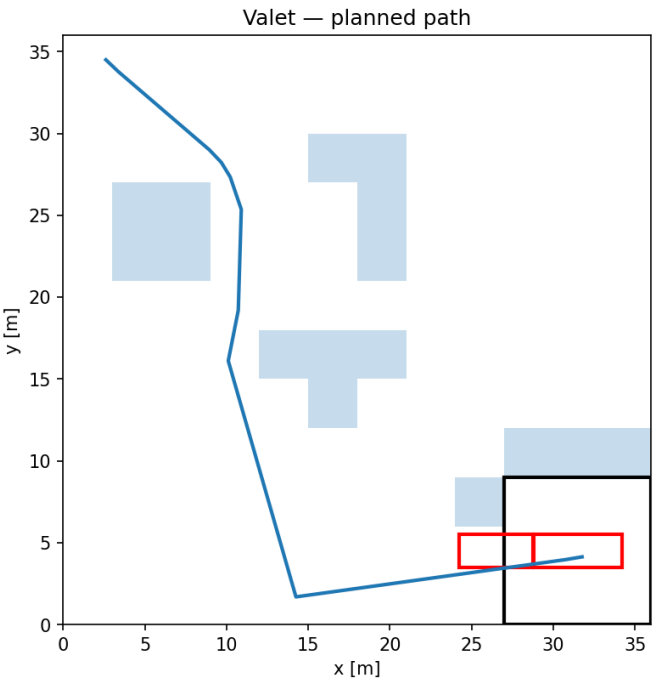

Fig. 10 — Truck + Trailer Valet Path Animation

Supplementary animation files accompany this section and illustrate the complete valet-parking maneuvers for the robot, car, and truck–trailer models.

V. Conclusions

The valet parking simulation demonstrates a complete Hybrid A* motion-planning pipeline for nonholonomic vehicles in a structured environment. The framework integrates vehicle kinematics, polygon-based collision detection, and grid search into a reproducible simulation that produces feasible parking trajectories. Results confirm that Hybrid A* can efficiently generate smooth paths when appropriate discretization and steering parameters are used. The project highlights the importance of coupling vehicle dynamics with planning logic in confined spaces. Minor performance limitations were observed in highly cluttered scenes and for the truck–trailer model, which could be improved by finer control resolution or smoothing techniques. Overall, the assignment fulfills the objectives of designing, implementing, and evaluating a working nonholonomic motion planner.

References

A. Academic and Algorithmic References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.
(Foundational concepts for motion planning and nonholonomic vehicle models.)

[2] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast motion planning in dynamic on-road driving scenarios," *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2010, pp. 1879–1884.
(Introduced Hybrid A approach for continuous-space search in vehicle planning.)*

[3] A. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.
(Detailed Hybrid A implementation for car-like robots.)*

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin, Germany: Springer, 2008.
(Used for the Separating Axis Theorem (SAT) in polygon collision checking.)

[5] A. Dosovitskiy *et al.*, "CARLA: An open urban driving simulator," *Proc. 1st Conf. Robot Learning (CoRL)*, 2017, pp. 1–16.

B. Software and Implementation References

[6] C.R. Harris et all,, "Array programming with NumPy," Nature, Vol. 585. pp. 357-362,2020. (User for vectorized numeric operations and kinematic computations.)

[7] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, , vol. 9, no. 3, pp. 90–95, 2007. *(Used for plotting and visualization of paths and scenes.)*

[8] A. Clark and Contributors, "imageio: A Python library for reading and writing image data," 2023. [Online]. Available: https://pypi.org/project/imageio/

C. AI-Assisted Contributions

[9] OpenAI, *ChatGPT* (GPT-5), OpenAI, San Francisco, CA, 2025. [Online]. Available: https://chat.openai.com/. (Used for report editing, pseudocode formatting, and improving documentation clarity; all source code and algorithms were written and verified by the author.)

[10] DeepSeek, *DeepSeek AI Model*, DeepSeek Inc., 2025. [Online]. Available: https://www.deepseek.com/. (Used for brief explanatory assistance and conceptual clarifications)