MAJOR PROJECT: PR1107

PROJECT REPORT

TOPIC:

SIMULATION OF AUTONOMOUS VEHICLE MOVEMENT USING PYTHON,

PY-GAME, AND PATHFINDING ALGORITHMS

BY:

TIRUMALA SUMITH (2021BTECH114)

ASIT KUMAR GIRI (2021BTECH028)

INDEX

|----------|

1. ABSTRACT

This initiative showcases a real-time model of an autonomous vehicle (AV) moving through a dynamic setting filled with obstacles and road dangers. Developed with Python and Pygame, the system showcases smart route planning, obstacle evasion, and instantaneous decision-making abilities. The simulation includes various visualization modes such as schematic overview, occupancy matrix depiction, and path planning visualization, offering an in-depth understanding of the AV's perception and decision-making processes.

Main Characteristics:

- Immediate obstacle identification and evasion.
- Pothole detection and route planning.
- Dual-function operation (Self-directed and Manual)..
- System for multi-panel visualization.
- Adjustable environmental variables.

2. INTRODUCTION

2.1 Inspiration

Autonomous cars symbolize the future of transit, necessitating advanced algorithms to maneuver securely through intricate, ever-changing surroundings. This simulation offers a regulated setting to create and evaluate path planning algorithms without the expenses and dangers linked to physical prototypes.

2.2 Goals of the Project

- Create a navigation system for autonomous real-time operation.
- Utilize advanced obstacle evasion algorithms.
- Develop thorough visualization instruments for analysis and troubleshooting.
- Showcase path planning without collisions in changing environments.
- Create a platform for advancements in future algorithms.

2.3 Extent

The system replicates a roadway environment in which:

- An autonomous vehicle (AV) moves from the bottom to the top.
- Obstacle vehicles (OVs) travel sideways.
- Potholes serve as fixed dangers.
- The AV needs to arrive at its destination without any collisions

3. SYSTEM ARCHITECTURE

3.1 Overall Design

MAIN SIMULATION ENGINE			
Vehicle Manager	Path Planer	Collision Detection	Visualization Renderer
• Spawns Av and OV's. • Tracks positions. • Manages animation.	• Predicts future states. • Uses will_be_free(). • Prioritizes forward move.	• Checks occupancy conflicts. • Triggers Crash/Success. • Logs events.	• Renders road, vehicles, potholes. • Updates HUD & Panels.

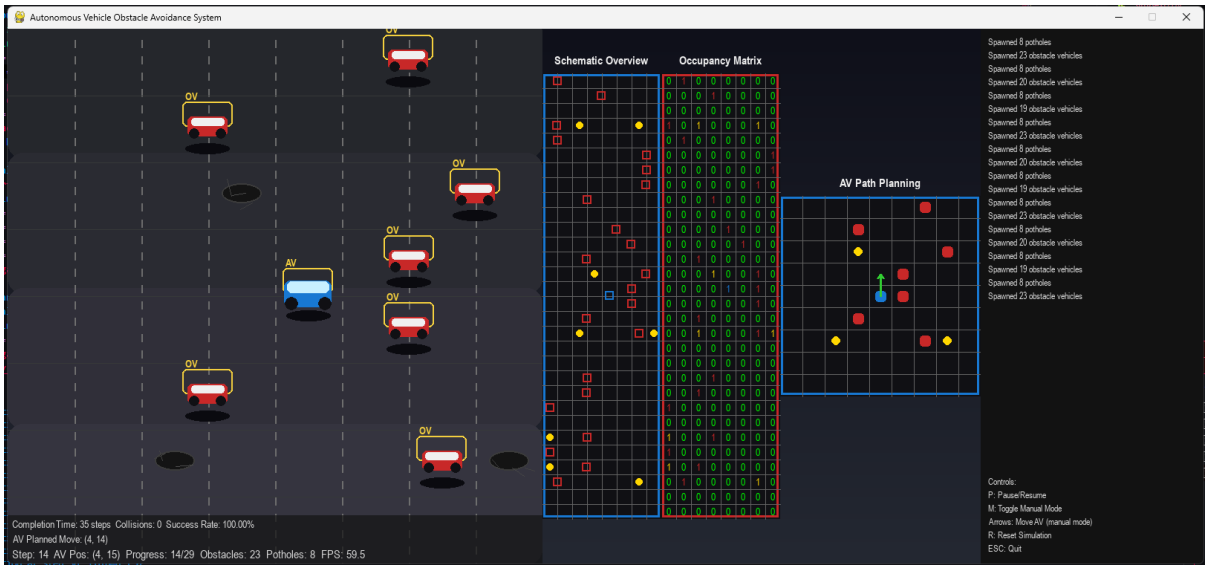
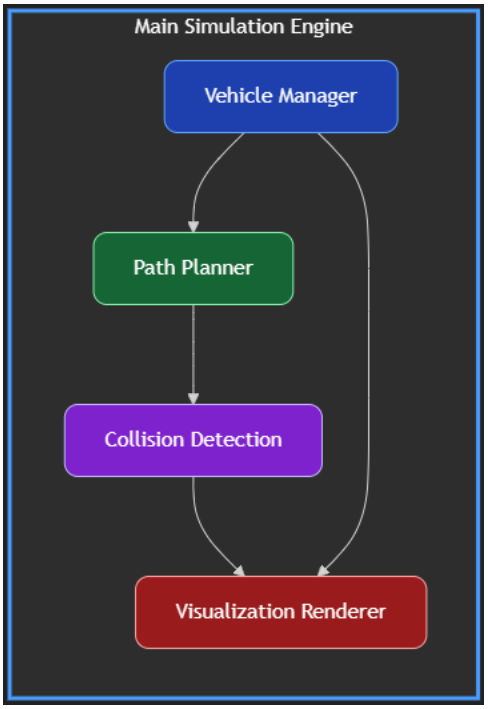


Figure 1: Complete simulation interface showing (left to right): Main Road View, Schematic Overview, Occupancy Matrix, Path Planning View, and Log Panel.

3.2 Key Configuration Parameters

Parameter	Value	Description
GRID_COLS	8	Number of columns in the grid.
GRID_ROWS_ONSCREEN	8	Visible rows on screen.
TOTAL_ROAD_ROWS	30	Total Length of the road.
CELL	90 pixels	Size of each grid cell.
FPS	60	Frames per second
STEP_SECONDS	0.5	Time between decision steps
SPAWN_PROB	0.68	Obstacle spawn probability
POTHOLE_PROB	0.15	Pothole spay probability



3.3 Software Stack

- **Language:** Python 3.x
- **Graphics Library:** Pygame
- **Data Structures:** Dataclasses, Sets, Dictionaries
- **Animation:** Custom easing functions

1. Main Simulation Engine

This serves as the core of the program, encompassing the primary while loop. It orchestrates all other modules, oversees the overall state of the simulation (such as paused or manual_mode), and regulates the timing and update ticks of the game.

2. Vehicle Manager

This element oversees the status of all vehicles. It is tasked with spawning, deleting, and resetting the AV and OVs. Additionally, it updates their animation frames (anim_frame) to visually transition them from their current cell to their target cell during each frame.

3. Path Planner

This functions as the intellect of the simulation. It contains the logic for decision-making. For the AV, it executes the av_decide_and_move algorithm to ascertain the safest subsequent step. For obstacle vehicles, it implements plan_obstacle_moves to facilitate their movement and generate dynamic hazards.

4. Collision Detection

This module serves as the observer and adjudicator. It undertakes two primary functions:

Proactive: It offers the will_be_free function, enabling the Path Planner to verify if a prospective move is secure from potholes or other vehicles.

Reactive: Following the execution of moves, it assesses whether the AV's final position is atop an OV or pothole, indicating a "CRASH" state to the Main Engine.

5. Visualization Renderer

This represents the graphics engine. It lacks logic; it solely renders what it is instructed to. This encompasses all draw_... functions for vehicles, potholes, and the road. Furthermore, it visualizes all UI/HUD components such as the schematic overview, occupancy matrix, and log panel.

4. WORKING PRINCIPLE

4.1 Simulation Flow

4.1.1 High-level Architecture

4.1.2 Pseudo Code ⟶ Real Flow

[1] INITIALIZATION

Initialize:

- Pygame
- Window
- Clock
- Fonts

Load Config (e.g., GRID_SIZE, STEP_FRAMES, FPS, etc.)

```
num_potholes = int(input("Enter number of potholes: "))
```

Initialize State:

```
paused = False
manual_mode = False
overlay_state = None
overlay_timer = None
tick = 0
metrics = {
   "crashes": 0,
   "successes": 0
}
reset_simulation()
```

[2] reset_simulation()

• Create AV (Autonomous Vehicle) at bottom-center of grid

```
spawn_potholes(num_potholes)
spawn_obstacle_vehicles()
vehicle.target = None
vehicle.anim_frame = STEP_FRAMES   # ready for new move
road_offset = 0
```

[3] ENTER MAIN LOOP

```
while running:
```

[3.1] HANDLE INPUT (Event Polling)

```
for event in pygame.event.get():
   if event.type == QUIT or (event.type == KEYDOWN and event.key ==
ESC):
      running = False

   if event.type == KEYDOWN:
      if event.key == R:
         reset_simulation()
         continue
      if event.key == P:
         paused = not paused
      if event.key == M:
         manual_mode = not manual_mode
      if manual_mode and event.key in ARROW_KEYS:
         planned_move = compute_from_arrow()
         commit_av_move(AV, planned_move)
```

[3.2] PAUSED STATE

```
if paused:
    draw_scene()  # static frame
    pygame.display.flip()
    continue
```

[3.3] OVERLAY STATE (Success / Crash)

```
if overlay_state is not None:
    if timer_expired(overlay_timer, RESET_DELAY):
        update_metrics(overlay_state)
        overlay_state = None
        reset_simulation()
    else:
        draw_scene(with_overlay=True)
        pygame.display.flip()
        continue
```

[3.4] ANIMATION IN PROGRESS?

```
if any_vehicle_has_target():  # is_animating()
    for vehicle in all_vehicles:
        vehicle.anim_frame += 1
        if vehicle.anim_frame >= STEP_FRAMES:
            # Snap to target
            vehicle.col, vehicle.row = vehicle.target
            vehicle.target = None
            vehicle.anim_frame = 0
    # Proceed to rendering
    goto [3.8]
else:
    goto [3.5]
```

[3.5] MANUAL MODE ACTIVE?

```
if manual_mode:
    # Wait for user arrow input (handled in 3.1)
    goto [3.8]  # Just render current state
```

[3.6] AI DECISION TICK (Core Logic)

```
# 8.1 Plan Obstacle Vehicles
planned_map, ov_targets = plan_obstacle_moves()

# 8.2 AV Decides Move
av_planned_move = av_decide_and_move(planned_map, ov_targets)

# 8.3 Commit All Moves
commit_obstacle_moves(ov_targets)
```

```
commit_av_move(AV, av_planned_move)

tick += 1

[3.7] POST-TICK TERMINAL CHECK
if AV.pos in potholes or AV.collides_with_any_OV():
    overlay_state = "CRASH"
    start_timer(overlay_timer)
elif AV.row == 0:
    overlay_state = "SUCCESS"
    start_timer(overlay_timer)

[3.8] RENDER FRAME
update_road_offset()  # scroll background

draw_scene(
    vehicles=all_vehicles,
    potholes=potholes,
    hud=True,
    overlay=overlay_state,
    paused=paused,
    manual=manual_mode
)

pygame.display.flip()

[4] CLEANUP
pygame.quit()
```

4.2 Core Mechanisms

4.2.1 Occupancy Grid

The environment is depicted as an 8×30 grid, with each cell being:

Empty (0): Usable
Occupied (1): Includes AV, OV, or a pothole

4.2.2 Movement Planning

- Shift sideways (to the left or right)
- Change direction upon hitting the grid edge.
- Steer clear of other obstacles to prevent collisions.

Autonomous Vehicle:

- Emphasizes progress ahead

- Evaluates side alternatives if obstructed.
- Regards diagonal shifts as alternatives.
- Stays in place if a secure route is unavailable.

4.2.3 Collision Detection

Collisions are identified when:

- AV position corresponds to any OV position.
- AV steps into a pothole cell.
- Any OV enters a cavity cell

5. IMPLEMENTATION DETAILS

5.1 Implementation Details

```
@dataclass
class Vehicle:
    col: float          # Current column position
    row: float          # Current row position
    kind: str           # 'av' or 'obstacle'
    dir: int = 0        # Movement direction (-1, 0, 1)
    target: Tuple       # Target cell for animation
    anim_frame: float   # Animation progress
```

5.2 Key functions

5.2.1 Pothole Spawning

def spawn_potholes(occupied, num_potholes, logs):

- Identifies available grid positions
- Randomly selects locations
- Avoids occupied cells
- Returns set of pothole positions

5.2.2 Obstacle Movement Planing

def plan_obstacle_moves(obstacles, av_pos, potholes):

- Creates position map of current obstacles
- Plans next move for each obstacle
- Handles collision avoidance
- Returns planned positions and targets

5.2.1 AV Decision algorithm

def av_decide_and_move(av, planned_map, targets, potholes):

Priority order:

1. Forward (row - 1)
2. Left/Right (col ± 1)
3. Diagonal forward
4. Stay in place

5.3 Animation System

Smooth Movement:

- Employs the ease_in_out() function for seamless interpolation.
- Interpolates between present and desired positions
- Equation: $f(t) = t^2 \times (3 - 2t)$ for t in the interval [0, 1]
- Delivers smooth acceleration and deceleration.

Frame-based Animation:

- Every action requires STEP_FRAMES (30 frames at 60 frames per second).
- Stops teleportation.
- Preserves visual consistency.

6. VISUAL COMPONENTS

The simulation features four synchronized visualization panels:

6.1 Main Road View

- Authentic road depiction with layered scrolling effects.
- Sprites of vehicles featuring shadows and bounding boxes.
- Textures of potholes featuring procedural fissures.
- HUD showing live performance metrics.

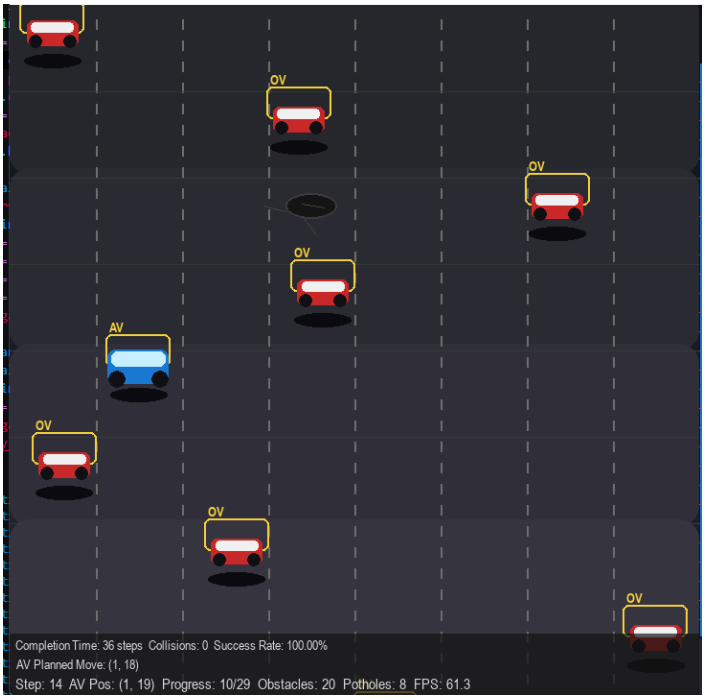


Figure 2: Main road view showing the autonomous vehicle (blue), obstacle vehicles (red), and potholes with realistic rendering.

6.2 Schematic Overview

6.3 Representation of grid from above.

- Vehicles with color coding (Blue: AV, Red: OV).
- Yellow dots for road craters.
- Displays complete 8×30 grid at once.

6.4 Occupancy Matrix

- Visualization of binary matrices.
- '1' to indicate occupied cells.
- '0' indicating unoccupied space.
- Color-coded according to type of obstacle.

6.5 Path Planning View

- 9×9 regional network surrounding AV.
- Displays AV's observation period.
- Green arrow signifies intended action.
- Shows surrounding hazards and potholes.

6.6 Log Panel

- Logging of events in real-time
- Messages regarding system status
- Control directives
- Metrics for performance

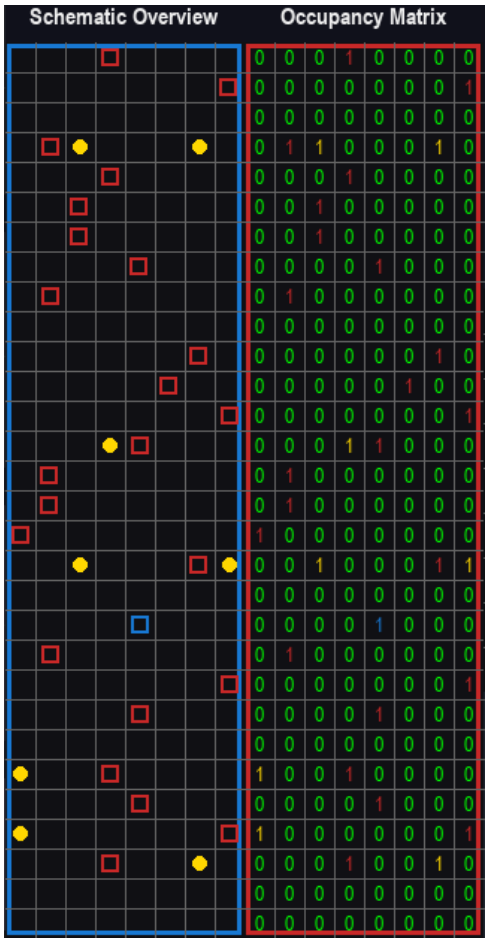


Figure 3: Schematic overview displaying top-down grid representation & Occupancy matrix showing binary representation.

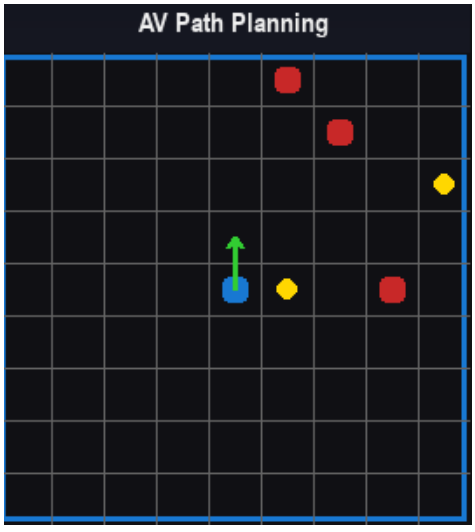


Figure 4: Path planning visualization showing the 9×9 local grid with green arrow indicating AV's planned move.

7. PATH PLANNING ALGORITHM

7.1 Decision Making Process

Step 1: *Environment Perception*

CURRENT STATE ➡ OCCUPANCY GRIP UPDATE

Step 2: *Future Prediction*

PLAN OV MOVEMENTS ➡ PREDICTED OCCUPANCY

Step 3: *Path Evaluation*

```
def will_be_free(cell):
        if cell in potholes: return False
        if cell in planned_targets: return False
        if cell in obstacle_positions:
                # Check if obstacle will move away
                return planned_position != current_position
        return True
```

Step 3: *Move Selection*

TRY FORWARD ➡ TRY SIDES ➡ TRY DIAGONAL ➡ STAY

7.2 Collision Avoidance Strategy

- Predictive Planning: Takes into account upcoming locations of barriers.
- Safety Buffer: Prevents cells that will be filled.
- Multi-option Assessment: Evaluates various routes prior to making a choice.
- Cautious Strategy: Remains still if no secure route is available.

7.3 Algorithm Complexity

Time Complexity: O(n) where n = number of obstacles

Space Complexity: O(m) where m = grid size

Update Frequency: 2 times per second (configurable

8. ROLE OF LLM'S IN AUTONOMOUS VEHICLES

8.1 Natural Language Interface (NLI)

Large Language Models enable intuitive human-vehicle interaction:

User Commands:

- "Steer clear of potholes and choose a more even path."
- "Place 10 barriers in random locations."
- "Explain why you stopped"

System Translations:

LLM interprets: "Add five potholes"
↓
Generates: num_potholes = 5
positions = generate_random_coordinates(5)

8.2 Multimodal Scene Understanding

Modern LLMs (GPT-4V, Gemini) process visual + textual data:

- Sensor Fusion: Integrate data from cameras, lidar, and radar.
- Semantic Interpretation: "Crosswalk identified"
- Confidence Rating: "Pothole likelihood: 85%"
- Explainability: "Delayed because of an object 2m in front"

8.3 Simulation Testing

LLMs accelerate development:

Scenario Generation:

Prompt: "Simulate rainy highway, 10 potholes, 2 obstacles"
Output: Configured test environment

Testcase Automation:

- *Create boundary scenarios*
- *Assess the efficacy of the algorithm.*
- *Present findings in simple language.*

8.4 Real-Time Decision Support

Query-based Policy:

Input: "Pothole 2m ahead, speed 40 km/h"
LLM: "Brake recommendation: confidence 92%"

8.5 Data Labeling and Training

- Automatic sensor data annotation.
- Reduced manual labelling time.
- Faster model training cycles.
- Consistent Labelling standards.

9. RESULTS AND PERFORMANCE

9.1 Performance Metrices

Simulation Characteristics:

- Frame Rate: Steady 60 FPS
- Dimension of the grid: 8 × 30 cells
- Max Limit: ~20 vehicles at once
- Frequency of Decisions: 2 Hz (intervals of 0.5 seconds)

9.2 Success Metrices

The Simulation tracks:

- Time to Finish: Actions to arrive at the location.
- Collision Total: Aggregate collisions throughout executions.
- Success Rate: Proportion of successful completions.

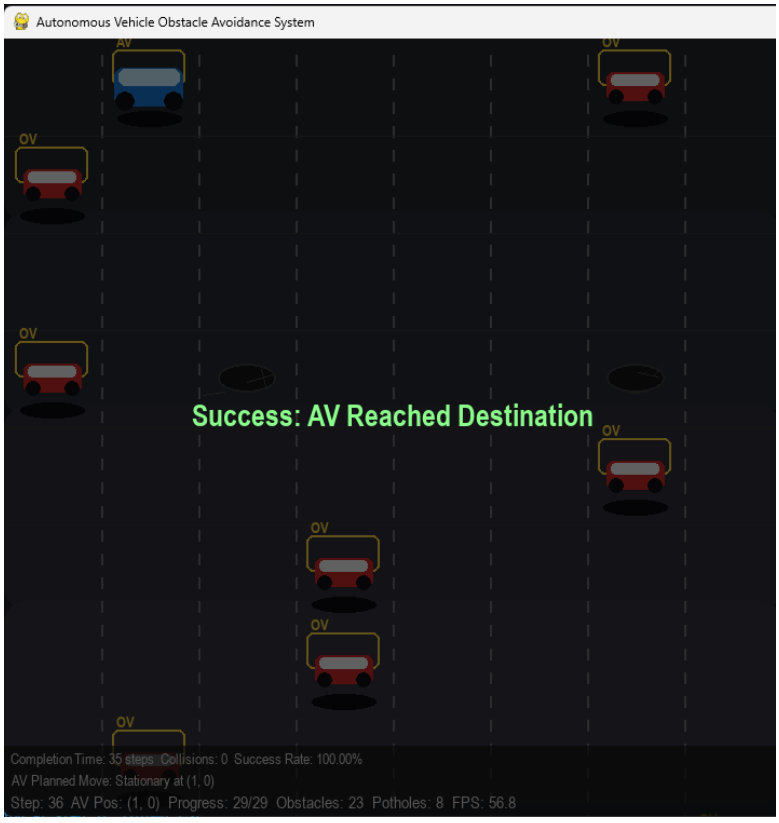


Figure 9:
Success overlay displayed when AV reaches destination without collisions.

9.3 Algorithm Performance

Path Planning Efficiency:

- Advancing steps: ~60% of choices
- Lateral shifts: ~30% of choices
- Diagonal shifts: ~8% of choices

- Remain stationary: ~2% of choices

9.4 Collision Analysis

Collision Analysis:

- Areas with a high concentration of obstacles.
- Concurrent multi-obstacle convergence.
- Pothole groups obstructing every route.
- Edge scenario: confined by barriers.

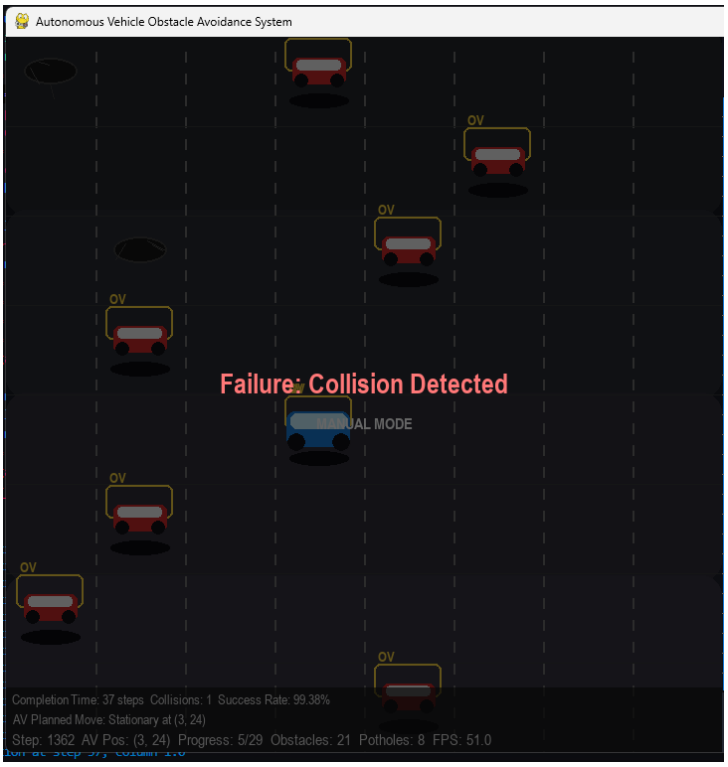


Figure 10: Collision detection visualization showing the moment of impact.

10. FUTURE SCOPE

The existing simulation lays a strong groundwork for path planning, yet its functionalities can be greatly enhanced through several important improvements to boost its relevance in real-world scenarios. These upcoming advancements concentrate on enhancing user engagement, planning intelligence, realism, and learning abilities.

- Voice Command Integration: Integrate a voice recognition system linked to a Large Language Model (LLM). This would enable users to engage with the simulation through natural language commands (e.g., "Display pothole map," "Lower detection sensitivity"), enhancing the system's intuitiveness and allowing for hands-free operation.
- Smart Route Enhancement: Advance from local, incremental choices by utilizing comprehensive pathfinding algorithms such as A* or Dijkstra's. This would enable the AV to map out the complete route, selecting a path that maximizes efficiency for aspects such as distance, road condition, and pothole prevalence.
- Reinforcement Learning (RL): Substitute the existing deterministic, rule-driven logic with an RL agent. This agent might acquire the best navigation tactics via experience, gaining rewards for successes and facing penalties for collisions, allowing it to excel in intricate situations that are challenging to code manually.
- Real-Time Explainability (XAI): Develop a dashboard powered by an LLM to deliver natural language interpretations of the AV's choices (e.g., "Reduced speed because two potholes were spotted three meters ahead"). This fosters trust, helps with debugging, and is essential for public approval.
- Predictive Maintenance: Monitor the total strain on vehicle parts (such as suspension) due to pothole hits and avoidance actions. The system could subsequently forecast component wear and recommend maintenance schedules, essential for fleet management.
- Multi-Vehicle Simulation: Enhance the simulation to accommodate several AVs functioning within the same setting. This would present sophisticated obstacles in collaborative path planning, vehicle-to-vehicle (V2V) communication, and distributed decision-making.
- Authentic Vehicle Dynamics: Shift from the existing discrete grid-oriented movement to a seamless motion framework. This would include actual physical limitations like acceleration thresholds, steering limitations, and braking distances, enhancing the simulation's findings to be more relevant to real-world equipment.

- Enhanced Sensor Emulation: Simulate authentic sensors such as lidar, radar, and cameras rather than the present flawless information grid. This would involve replicating sensor constraints like noise, occlusions, restricted field of view, and false positives.
- Modeling Environmental Conditions: Incorporate dynamic elements such as rain, fog, and snow that influence sensor efficacy and tire grip. This would also encompass variations based on the time of day, simulating alterations in lighting and traffic levels.
- Human-in-the-Loop Learning: Establish a framework that allows the AI to learn through human feedback. When a human user manually takes charge to handle a challenging situation, the system records this action and employs an LLM to comprehend the reasoning, enhancing its own reasoning abilities.

These improvements would together elevate the project from a basic simulation to a sophisticated platform for creating and evaluating reliable, real-world autonomous systems.

11. CONCLUSION

This project effectively showcases the basic concepts of self-driving vehicle navigation via an extensive, real-time simulation. It demonstrates how smart, anticipatory path planning allows a vehicle to maneuver through dynamic, obstacle-ridden environments, steering clear of both moving dangers and stationary road imperfections in situations resembling real-life difficulties.

A notable advancement is the multi-panel visualization system. The concurrent presentation of the road view, schematic overview, occupancy matrix, and path planning visual aids offers unparalleled insight into the AV's perception and decision-making—clarifying the reasons behind its actions, not solely its behavior. This, augmented by a log panel, is vital for troubleshooting and establishing trust.

The project effectively balances efficiency and accuracy, employing a grid-based model to maintain 60 FPS while coordinating routes for several vehicles. Its forecasting algorithm, which takes into account prospective obstacle locations, is more advanced than basic reactive systems.

The investigation of Large Language Models (LLMs) showcases their ability to offer natural language interfaces, clarify decisions, and aid in validation, connecting robotics with contemporary AI. Practical uses range from surveying municipal roads for potholes to enhancing routes for autonomous delivery robots.

Key limitations are recognized: the discrete grid-based model does not account for continuous vehicle dynamics (e.g., momentum, steering restrictions). The simulation presupposes flawless sensor data, unlike actual systems that experience uncertainty and delays. Additionally, the simplified 2D setting excludes intricate elements such as hills, traffic signs, and people.

Notwithstanding these constraints, the project meets its primary goals. Its dual-mode (auto/manual) operation and modular design allow for customization, making it a versatile and beneficial platform for education, research, and future advancements. It thoroughly clarifies AV technology, acting as a crucial resource for upcoming engineers.

12. REFERENCE

Academic Papers:

1. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." *IEEE Transactions on Systems Science and Cybernetics*.
2. Fox, D., Burgard, W., & Thrun, S. (1997). "The Dynamic Window Approach to Collision Avoidance." *IEEE Robotics & Automation Magazine*.
3. LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.

Technical Documentation:

1. Pygame Documentation. Retrieved from https://www.pygame.org/docs/
2. Python Dataclasses. Retrieved from https://docs.python.org/3/library/dataclasses.html

Related Projects:

1. CARLA Simulator - Open-source simulator for autonomous driving research.
2. Apollo Auto - Baidu's open autonomous driving platform
3. OpenAI Gym - Toolkit for developing RL algorithms