Project Proposal: Development of a Software-Based Test Environment for Autonomous Vehicles Using Google Street View

1. Project Title

Development of a Virtual Test Environment for Autonomous Vehicle Algorithms Utilizing Google Street View Imagery

2. Project Overview

2.1 Background

Autonomous vehicles (AVs) represent a transformative technology in transportation, promising improved safety, efficiency, and accessibility. However, testing AV systems in real-world environments poses significant challenges, including high costs, safety risks to human participants, regulatory hurdles, and the need for diverse testing scenarios. Traditional simulation tools like CARLA or Gazebo provide synthetic environments, but they often lack the realism of actual urban landscapes. Google Street View offers a vast repository of real-world panoramic imagery captured from streets worldwide, providing an opportunity to create a cost-effective, software-based testbed that bridges the gap between synthetic simulations and physical testing.

This project aims to develop a software platform that leverages Google Street View API to construct virtual driving scenarios for testing AV perception, navigation, and decision-making algorithms. By simulating vehicle movement through static Street View images, the system will enable students and researchers to evaluate AV software in realistic settings without the need for physical hardware or on-road trials.

2.2 Problem Statement

Current AV testing methods are either:

- **Real-world testing**: Expensive, time-consuming, and potentially hazardous.
- **Simulation-based testing**: Often limited by artificial environments that do not fully capture real-world complexities like varied lighting, occlusions, or urban clutter.

A software-based approach using publicly available Street View data can provide an accessible alternative, allowing rapid prototyping and iteration of AV algorithms in a controlled, virtual space.

2.3 Significance

This project will democratize AV testing for educational purposes, enabling students to experiment with cutting-edge technologies using minimal resources. It could also contribute to open-source tools for the AV research community, potentially reducing the entry barriers for small teams or academic institutions.

3. Objectives

3.1 Primary Objective

To design and implement a software framework that integrates Google Street View data to simulate driving environments for testing autonomous vehicle algorithms.

3.2 Secondary Objectives

- Integrate computer vision techniques to process Street View panoramas for object detection, lane recognition, and obstacle avoidance.
- Develop a user interface for configuring test scenarios, such as route selection, weather simulation (via image augmentation), and dynamic element injection (e.g., virtual pedestrians or vehicles).
- Evaluate the system's effectiveness by testing sample AV algorithms (e.g., path planning or collision detection) against benchmark scenarios.
- Document limitations and propose enhancements for handling dynamic real-world elements not captured in static Street View images.

4. Methodology

4.1 System Architecture

The proposed system will consist of the following modules:

- **Data Acquisition Module**: Use the Google Street View Static API to fetch panoramic images based on GPS coordinates, headings, and pitch. Routes will be defined using Google Maps API for path planning.
- **Simulation Engine**: A custom simulator built in Python using libraries like OpenCV for image processing, NumPy for numerical computations, and Pygame or Unity for rendering the virtual environment. The simulator will "stitch" sequential Street View images to mimic vehicle motion.
- **AV Algorithm Integration**: Incorporate open-source AV stacks (e.g., from Apollo or Autoware) or simple custom implementations for perception (using YOLO for object detection) and control (PID controllers for steering).
- **Testing Interface**: A graphical user interface (GUI) developed with Tkinter or Flask, allowing users to select starting points, define test parameters, and visualize results.

4.2 Technologies and Tools

- **Programming Languages**: Python (primary), with potential C++ extensions for performance-critical parts.
- **Libraries**: Google Maps/Street View APIs, OpenCV, TensorFlow/PyTorch for ML-based perception, Matplotlib for visualizations.
- **Development Environment**: Jupyter Notebooks for prototyping, Git for version control.
- **Hardware Requirements**: Standard laptop with GPU support for faster image processing (optional).

4.3 Assumptions and Limitations

- Street View data is static; dynamic elements (e.g., moving traffic) will be simulated via overlays or procedural generation.
- API usage will comply with Google's terms, including rate limits and fair use policies.
- The system will focus on perception and planning, not full vehicle dynamics (e.g., no physics engine for collisions).

5. Expected Outcomes

- A functional prototype software tool deployable as a desktop application or web app.
- Demonstration of AV testing in at least 3 diverse scenarios (e.g., city streets, highways, residential areas).
- Performance metrics showing how the virtual environment compares to real-world benchmarks (e.g., via confusion matrices for object detection).
- Open-source code repository on GitHub, including setup instructions and sample datasets.
- A final report and presentation detailing the project's findings, suitable for academic submission.

6. Budget

This project is designed to be low-cost, leveraging free/open-source tools. Estimated expenses:

Item	**Description**
Google API Key	Potential costs if exceeding free tier (e.g., high-volume queries)
Compute Resources	Cloud GPU for ML training (if needed, e.g., Google Colab Pro)
Software Licenses	None (all open-source)
Miscellaneous	Domain for web demo (optional)
Total	

Funding could be sourced from university grants or personal resources.

7. Team and Resources

- **Team Members**: 3-4 members with roles in software development, computer vision, and testing.
- **Supervision**: Supervised by Prof. Huang.
- **Risks and Mitigation**: API rate limits – Use caching; Technical challenges – Start with simple prototypes; Time constraints – Prioritize core features.