Basic framework in SQL and Python for implementing the autonomous vehicle project using LiDAR data. The code focuses on database design, data storage, and retrieval for real-time processing.

1. Database Schema for SQL

sql

CopyEdit

```sql
-- Create a database for the project
CREATE DATABASE AutonomousVehicle;
USE AutonomousVehicle;
-- Table for storing LiDAR point cloud data
CREATE TABLE LiDAR_Data (
   id INT AUTO_INCREMENT PRIMARY KEY,
   timestamp DATETIME NOT NULL,
   x_coordinate FLOAT NOT NULL,
   y_coordinate FLOAT NOT NULL,
   z_coordinate FLOAT NOT NULL,
   object_type VARCHAR(50) NOT NULL, -- e.g., vehicle, pedestrian, pothole
   confidence_level FLOAT NOT NULL
);
-- Table for road condition data (e.g., potholes, speed breakers)
CREATE TABLE Road_Conditions (
   id INT AUTO_INCREMENT PRIMARY KEY,
   location VARCHAR(255) NOT NULL,
   latitude FLOAT NOT NULL,
   longitude FLOAT NOT NULL,
   condition_type VARCHAR(50) NOT NULL, -- e.g., pothole, obstruction
   severity INT NOT NULL, -- 1 (low) to 5 (high)
   timestamp DATETIME NOT NULL
```

```sql
);
-- Table for traffic patterns and congestion data
CREATE TABLE Traffic_Data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    road_segment VARCHAR(255) NOT NULL,
    average_speed FLOAT NOT NULL,
    traffic_density INT NOT NULL, -- Number of vehicles
    timestamp DATETIME NOT NULL
);
```

---

2. Python Code for Ingesting and Processing Data

This code integrates LiDAR data and performs real-time processing using SQL queries.

Install Required Libraries

```
pip install mysql-connector-python
```

Python Code

```python
import mysql.connector
from datetime import datetime
# Database connection
def connect_to_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="your_password",
        database="AutonomousVehicle"
    )
# Insert LiDAR data
def insert_lidar_data(db_conn, x, y, z, object_type, confidence):
    cursor = db_conn.cursor()
    query = """
    INSERT INTO LiDAR_Data (timestamp, x_coordinate, y_coordinate,
```

```
z_coordinate, object_type, confidence_level)
    VALUES (%s, %s, %s, %s, %s, %s)
    """
    timestamp = datetime.now()
    cursor.execute(query, (timestamp, x, y, z, object_type, confidence))
    db_conn.commit()

# Query road conditions
def get_road_conditions(db_conn, latitude, longitude, radius=0.01):
    cursor = db_conn.cursor(dictionary=True)
    query = """
    SELECT * FROM Road_Conditions
    WHERE ABS(latitude - %s) <= %s AND ABS(longitude - %s) <= %s
    """
    cursor.execute(query, (latitude, radius, longitude, radius))
    return cursor.fetchall()
# Example: Process real-time data
if __name__ == "__main__":
    db_conn = connect_to_db()
    # Insert a LiDAR point (example data)
    insert_lidar_data(db_conn, x=10.5, y=20.3, z=5.2, object_type="vehicle",
confidence=0.98)
    # Fetch road conditions near a specific location
    conditions = get_road_conditions(db_conn, latitude=19.0760, longitude=72.8777)
    for condition in conditions:
        print(f"Condition: {condition['condition_type']}, Severity:
{condition['severity']}")
    db_conn.close()
```

---

3. SQL Queries for Real-Time Processing

- **Get Nearby Obstacles:**

```
SELECT * FROM LiDAR_Data
WHERE timestamp > NOW() - INTERVAL 10 SECOND
```

AND object_type IN ('vehicle', 'pedestrian');

- **Identify High-Severity Road Conditions:**

SELECT * FROM Road_Conditions

WHERE severity >= 4

ORDER BY severity DESC;

- **Predict Traffic Congestion:**

SELECT road_segment, AVG(traffic_density) AS avg_density

FROM Traffic_Data

WHERE timestamp > NOW() - INTERVAL 1 HOUR

GROUP BY road_segment

ORDER BY avg_density DESC;

Key Functionalities

1. **Data Ingestion: LiDAR data and road conditions are ingested and stored in SQL tables.**
2. **Real-Time Querying: Retrieve critical data (e.g., nearby obstacles, road anomalies).**
3. **Predictive Analytics: Analyze historical traffic data for route optimization.**
4. **Integration: Use Python for real-time interaction between the vehicle and the database.**