Self-Driving Software System

Daniel Flores, Alyssa Mollner, Julian Jamil

Professor Miranda Parker

CS 250 Section: 02

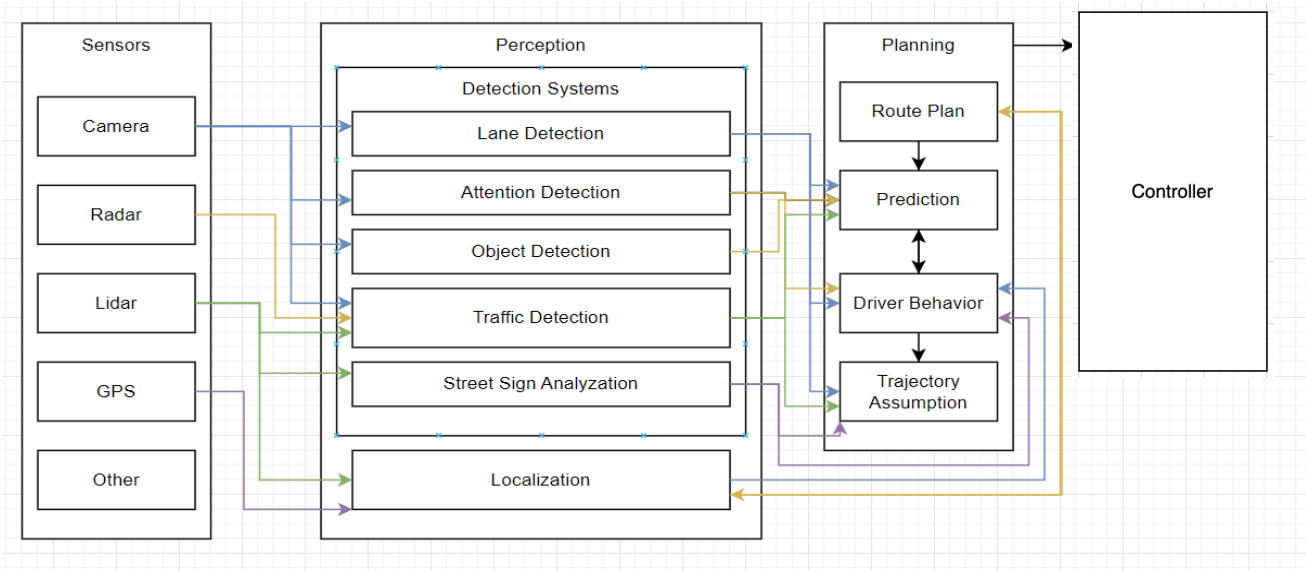
11 October 2024

System Overview:

This self-driving software system helps vehicles navigate without needing any steering or acceleration from a user or driver. This system is designed to be completely hands-free, allowing passengers to get from one place to another safely and efficiently without needing a driver or chauffeur. Its main purpose is to make roads safer by reducing the factors that cause humans to cause accidents and also to improve how cars drive on their own and do it in a safe way. This is especially helpful for people who have health problems or physical limitations that may prevent them from driving or for others who are simply scared or do not know how to drive. With this system, they can use their vehicles without worrying about causing harm to themselves or others. Beyond helping those with health challenges, this system can also reduce car accidents by removing common mistakes made by drivers. When the system is turned on, it takes full control of the vehicle, avoiding errors like distractions, wrong turns, or poor decisions. Other human errors it helps avoid are fatigue, reaction times, or navigating weather conditions poorly. The system has important features such as real-time navigation, which helps the car find and follow the best route to its destination. It also has obstacle detection, meaning it can spot any objects or hazards on the road and respond appropriately, either by slowing down, stopping, or steering around them. This software also monitors the passenger to make sure they remain aware of their surroundings, even though they aren't driving. Other features include checking the health of the vehicle, showing key data to the passenger, and adjusting the vehicle's behavior in different weather conditions to ensure a smooth and safe ride.

Software Architecture Overview

Architectural diagram of all major components



1. Sensors

Camera: The camera records visual data that is used for lane detection, traffic signs, and other object recognition that can help avoid any collisions,

Radar: The radar detects objects and obstacles, especially when there are weather conditions such as fog, that make it difficult for the camera to detect.

Lidar: The lidar uses laser scanning to create detailed 3D maps of the environment surrounding the car which also helps in object detection and lets the car know where it is.

GPS: The GPS provides information about the position and location of the car and helps it understand where exactly it is on the map.

Other: Could include other additional features or sensors, depending on the model of the car.

2. Perceptions

- Lane Detection: Lane Detection identifies the lane boundaries using the camera data, keeping the car in between the lines at all times unless it is changing lanes.
- Attention Detection: If the passenger chooses to drive the car or even if they are not driving, the attention detection system detects if they are distracted and alerts them to pay attention in case of a dangerous situation.
- Object Detection: The object detection uses camera data to detect certain objects to avoid on the road such as pedestrians, obstacles, or other random objects.
- Traffic Detection: Traffic detection also uses the data from the sensors to recognize traffic signs or signals to make sure the car obeys the rules on the road.
- Street Sign Analyzation: Similar to traffic detection, street sign analysis extracts necessary information such as stop signs or the numbers on speed limits

- Localization: Pulls data from GPS and other sensors to pinpoint the car's exact location on the map.

3. Planning

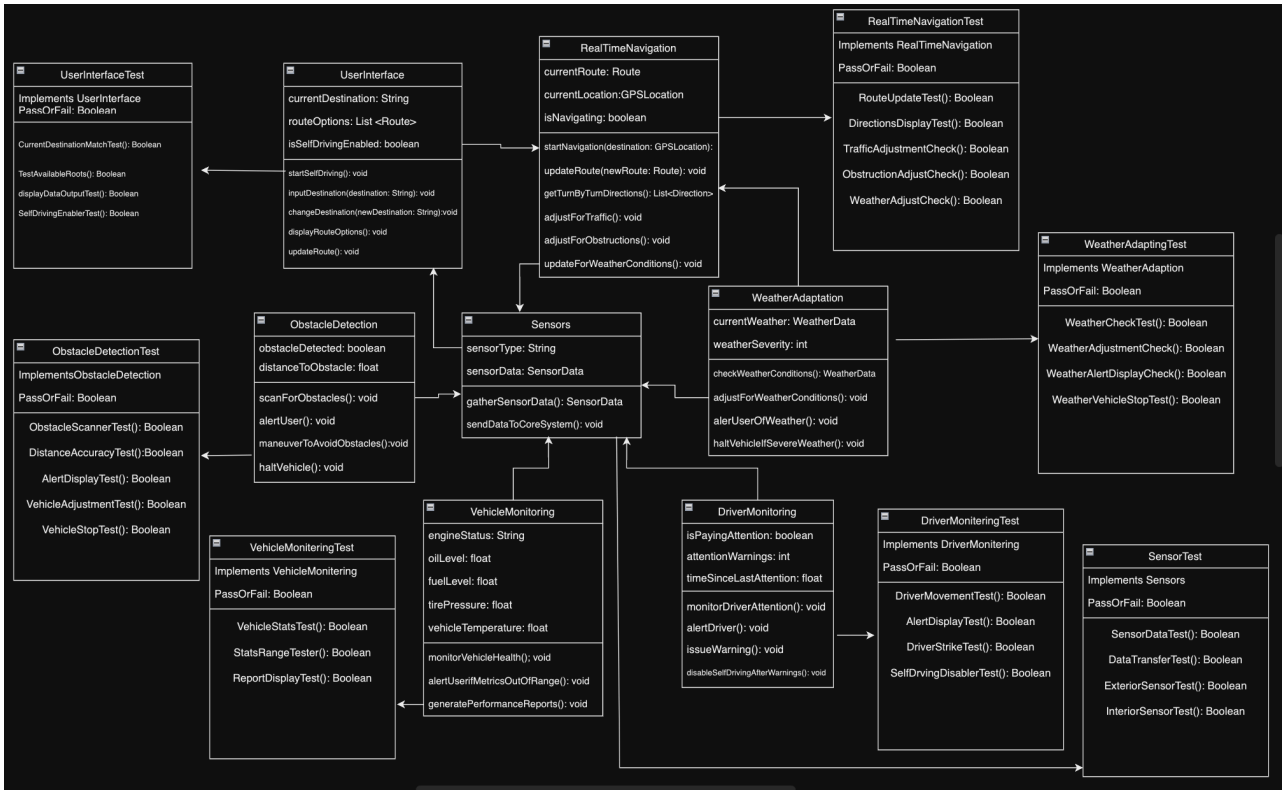
- Route Plan: Uses navigation data to create an efficient route from point A to B.
- Prediction: Prediction predicts the movement of surrounding objects on the road such as other vehicles, pedestrians, or other objects in order to prepare in case it needs to stop to avoid those items.
- Driver Behavior: If the user inputs anything, the car will do as the driver says.
- Trajectory Assumption: This calculates the safest and most efficient route and it decides this based on the ETA's given by GPS.

4. Control

- Controller: The controller adjusts the car's speed, the steering angle, and braking based on the real time input it is receiving.

UML Class Diagram

UML Class Diagram



1. User Interface

- currentDestination - String: this is the input given by the user and it is represented by a String such as an (ex: address, location name)
- routeOptions - List<Route>: Gives a list containing all the possible routes (Route objects) to the destination. Each route offers different paths and conditions (ex: shortest, fastest)
- isSelfDriving - Boolean: Boolean which indicates whether the self-driving mode is active (ex: true or false)

2. RealTimeNavigation

- currentRoute - Route from list <Route> : shows the currently selected route and is represented as a Route object that contains information about the path selected.
- currentLocation - GPS location: represents the car's current location on the GPS and can contain information such as longitude and latitude or direction of vehicle and others.
- isNavigating - Boolean: Tracks whether the car is actively navigating to its chosen destination.

3. ObstacleDetection

- obstacleDetected - Boolean: this indicates whether an obstacle has been detected (ex: true if yes, false if no)
- distanceToObstacle - Float: measures the distance to the detected obstacle (ex: in meters or other units) (driver can choose units)

4. Sensors

- sensorType - String: this is a string specifying the type of sensor (ex: proximity, weather, temperature).
- sensorData - SensorData: represents the data collected from the sensor as a SensorData object which can include data such as the sensor accuracy or time stamps.

5. WeatherAdaptation

- currentWeather - WeatherData: gives information on current weather conditions affecting the car (ex: temperature, precipitation, wind speed)

- weatherSeverity - int: measures how severe the weather is (ex: 0 for clear skies, 1 for light rain, until it reaches 5 for severe storms) Higher numbers mean more dangerous conditions to drive.

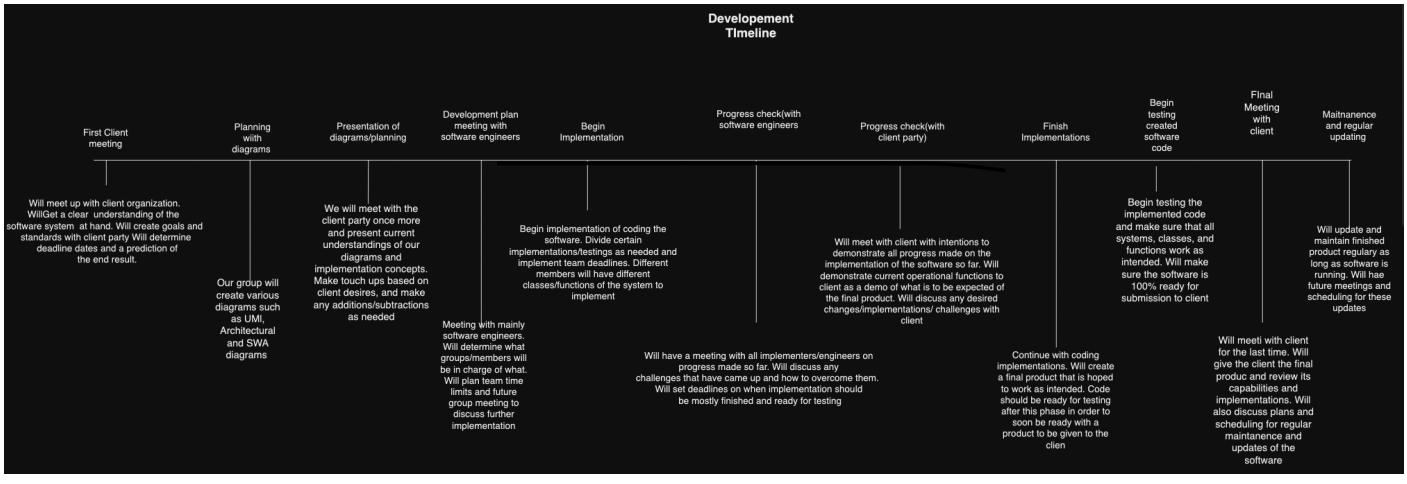
6. VehicleMonitoring

- engineStatus - String: gives current state of engine (ex: engine running, engine stopped)
- oilLevel - Float: gives the amount of oil in the car measured in liters or gallons
- fuelLevel - Float: gives remaining fuel in the car measured in liters or gallons
- tirePressure - Float: air pressure in the tires measured in units like PSI
- vehicleTemperature - Float: overall temperature of the vehicle's components measured in degrees Celsius or Fahrenheit.

7. DriverMonitoring

- isPayingAttention - Boolean: indicates if the driver is attentive, can be monitored by sensors such as cameras in front of the driver (true if yes, false if no).
- attentionWarnings - int: gives number of warnings issued to the driver for not paying attention.
- timeSinceLastAttention - float: gives the time (in seconds or minutes) since the last attention warning was given so it can decide when to give another warning.

Development plan and timeline



UNIT TESTS:

Test 1: Obstacle Detection

The feature that we are testing is the obstacle detection and its ability to detect obstacles and measure the distance from the obstacle correctly. This unit test focuses on checking that the obstacles are accurately identified within a 10 meter range. The test takes an obstacle placed at 5 meters. If the distance is within the 10 meter range, the system sets obstacleDetected to true and distanceToObstacle to the distance between the car and obstacle. If both conditions are met, the test passes and confirms that the obstacle detection function works.

Pseudocode:

```
ObstacleDetection.detectObstacle(distance);
distance = 5.0;
maxRange = 10.0;
```

```
bool obstacleDetected;
if distance <= maxRange
obstacleDetected = true;
distanceToObstacle = distance;
return PASS;
else
return FAIL;
```

Test 2: Speed Control

This unit checks the functionality of the speedControl component to make sure it correctly sets the vehicle's speed to a value that is equal to or less than the speed limit. In this test, targetSpeed is set to 30 km/hr which is below the maxSpeedLimit of 50 km/hr. The test verifies that the system correctly updates currentSpeed to targetSpeed. If this condition is met, the test passes and confirms that the speed control function prevents the car from going over the speed limit.

Pseudocode:

```
SpeedControl.setSpeed(targetSpeed);
targetSpeed = 30;
maxSpeedLimit = 50;
if (targetSpeed <= maxSpeedLimit):
currentSpeed = targetSpeed;
return PASS;
else
return FAIL;
```

INTEGRATION TESTS:

Test 1: Obstacle Detection and Emergency Braking

This integration test checks to make sure that the ObstacleDetection and EmergencyBraking components work well together. In this test, an obstacle is set to 5 meters from the car, just like in the first unit test, below the 10 meter threshold. First, the ObstacleDetection component

checks if the obstacle is within the allowed range, setting obstacleDetected to true and recording distanceToObstacle. If obstacleDetected is true, the EmergencyBraking component is then tested to confirm it correctly activates the brakes in response to the detection. If both components work together as expected and the brakes are activated, the tests pass confirming that the system can detect obstacles and brake in order to avoid crashing into them.

Pseudocode:

```
testObstacleDetectionAndEmergencyBraking();
ObstacleDetection obstacleDetector;
EmergencyBraking brakeSystem;
distance = 5;
maxRange = 10;
obstacleDetector.detectObstacle(distance);
if (obstacleDetector.obstacleDetected == true and obstacleDetector.distanceToObstacle ==
distance)
if (brakeSystem.activateBrakes(obstacleDetector.obstacleDetected ==true)
return PASS;
else return FAIL;
else return FAIL;
```

Test 2: User Interface and Real Time Navigation

This integration test makes sure that the User Interface and Real Time navigation components work well together. The test begins by setting a destination through the user interface, simulating the user's action of inputting their desired location. After this, the system's navigation function comes in to process the inputted location. The test then verifies whether the navigation system reflects the destination that the user inputted in the user interface. If the destination set in the user interface aligns with the navigation system's current route, thet test passes. If not, the test fails.

Pseudocode:

```
Navigation Test();
UserInterface  userInterface = new userInterface();
RealTimeNavigation navigation = new RealTimeNavigation();
```

```
userInterface.currentDestination = “1234 Main Street”;
navigation.setDestination(userInterface.currentDestination);
If navigation.currentRoute.destination == userInterface.currentDestination;
return PASS;
Else
return FAIL;
```

SYSTEM TESTS:

Test 1: Self Driving Activation and Navigation

This system test simulates a full self driving journey, from activation to destination arrival, including obstacles that are encountered along the route. The test begins with a user enabling self driving mode and inputting a destination. The vehicle should autonomously start navigation, following the route while maintaining real time updates of its GPS location. After a while, an obstacle is introduced to simulate a potential hazard. The system’s obstacle detection feature should recognize the obstacle, triggering the control system to either activate brakes or reroute as needed. Also, if the vehicle requires driver intervention at any point, an alert should notify the user, This test verifies the end to end functionality of the self driving system, making sure that all the critical features work together to make it a safe and efficient journey.

Test 2: Vehicle Monitoring and Alerts

This system test evaluates the car’s ability to monitor driver attention and critical vehicle conditions, such as fuel levels and tire pressure, and makes sure it provides alerts on time to the user. The test simulates scenarios where the driver becomes inattentive and when there is need for vehicle maintenance such as low fuel or low tire pressure. The system should detect inattentiveness, such as if the driver’s gaze shifts away from the road, and send an alert to the user so that they can pay attention to the road. The vehicle monitoring system should provide maintenance alerts if the needed components to run the car are below safe levels, in order for the driver to be safe on the road.

Data Management Strategy

For our self-driving software system, we have chosen SQL. SQL databases are consistent and reliable and also a good structure for all the data types we are working with for this self driving software. For example the sensor logs, user data, vehicle maintenance records, route information, etc.

Diagrams and Database Organization

The SQL database is organized into several tables to maintain data and make it efficient

Example:

User: stores user profiles, preferences, and trip histories

Vehicle Status: Logs details like fuel levels, engine health, and maintenance history.

Routes: Contains route information, including saved routes and frequently visited destinations.

Trips: Logs trip details and links to both Users and Routes

Sensor Data: Logs real time sensor data such as the radar data which is collected when the vehicle is driving

Maintenance Logs: Records service and maintenance history to track vehicle health

These data are all related. For example, the Users table links to Routes table to store historical data, while Vehicle Status connects with Maintenance Logs to track vehicle Health

SQL Diagrams

USERS

user_id	name	preferences	trip_history

ROUTES

user_id	route_id	route_details

TRIPS

user_id	trip_id	route_id	trip_details

VEHICLE STATUS

status_id	vehicle_id	fuel_level	engine_status

MAINTENANCE LOGS

maintenancelog_id	vehicle_id	service_date	log_details

SENSOR DATA

sensordata_id	trip_id	sensor_type	data_value

Advantages of SQL for Data Management

SQL databases has a structure that is ideal for our system's specific data

For example:

- It has consistent, structured data. User profiles and Trip logs are good to keep in this data because it ensures reliable access to the data.
- It also is reliable with vehicle monitoring. It logs for vehicle status and maintenance which is important because in order for the car to stay healthy and safe we need to keep track of the maintenance log and records.

Trade Offs

Why we chose SQL:

- It ensures data reliability and it is also accurate, which is important for data that contains key features like trip records, car maintenance logs and other things like user preferences.
- It is also more organized especially when it comes to the data relationships. It allows us to link related data easily, such as connecting a user's trips or health records.
- SQL's language allows us to run complex searches such as filtering trips by duration or identifying unusual sensor readings that may happen sometimes. It can help us create a better software with that with minimal mistakes

Tradeoff

- SQL can be less scalable for high frequency data, like sensor readings, which need quick processing and flexible data formats.
- Our solution is: We can use indexing and data partitioning to speed this up without losing data or data consistency

Tradeoff

- SQL requires a fixed data structure which could be a limitation if we need to adjust how we store some of this data such as sensor or vehicle data.
- Our solution is: Since our data does need to be stable and shouldn't have much room for flexibility, it is good to have fixed structures and have them work well.

Why we did not choose NoSQL:

- NoSQL databases are very flexible and scalable but they lack the strong data consistency that our system requires. This makes SQL a better choice for managing this important data and is also very reliable.
- SQL meets our goals for reliable data, good structure, and it has data relationships which are important AND crucial for the safety and efficiency of not only our self driving system but also of our driver.