Creation and Variation of Traffic Scenarios for Virtual Validation of Automated Driving Systems

Marco Roggero[1], Shashank Sharma[2], Will Tripp[3], Mark Corless[4], Sravan Kumar Chaganti[5]
1: The MathWorks GmbH, Aachen, Germany
2: The MathWorks GmbH, Ismaning, Germany
3: The MathWorks Inc, Natick (MA), USA
4: The MathWorks Inc, Novi (MI), USA
5: The MathWorks India Pvt Ltd, Hyderabad, INDIA

Abstract—Automated driving is getting massive attention from industry, road authorities, and consumers. Tests are crucial for convincing the market that autonomous vehicles are reliable and secure. Virtual validation provides an efficient and systematic way to test these systems. Such validation requires simulation environments that closely resemble real-world environments and include control algorithms, options for creating traffic scenarios, sensor models, and representations of vehicle dynamics. In this paper, we first show different ways of generating virtual driving scenarios. These ways include designing virtual driving scenarios using both scripts and graphical user interfaces, importing scenarios from preexisting libraries, and generating scenarios using data recorded from on-vehicle sensors. In a second section, we explain how to programmatically create variations of an existing scenario. The generated scenarios can be used in closed-loop simulations to model and test controllers.

Keywords—Automated driving; Driving scenario; Virtual validation; Scenario generation

I. INTRODUCTION

Validation of autonomous driving (AD) systems requires, besides road tests, computer-based simulations in which perception and control algorithms are tested under many different conditions. Such conditions include events such as image reflections, changes in road marking colors, or the presence of unexpected objects in the driving environment. If compared with traditional automotive simulation methods, simulations for AD need model not only the ego vehicle but also the behavior of vehicles, pedestrians, and other actors. Simulations must also model sensors such as lidars, cameras, and radars to enable the generation of synthetic sensor data. AD systems cannot be developed without simulations.

In this paper we first describe different options for scenario creation and simulation in MATLAB®. We then focus on two examples for the cuboid simulation environment that show how to generate a virtual scenario from recorded vehicle data and how to programmatically change scenario parameters to generate many different traffic conditions on the same road. The framework based on MATLAB provides two simulation environments in which to test algorithms: cuboids and 3D. Both environments have their advantages and are useful for different steps in the development process.

A. Cuboid Simulation Environment

In this environment, vehicles and other actors in the scenario are represented as simple box shapes. This simulation environment is useful for rapidly authoring scenarios, generating detections using low-fidelity radar and camera sensors, and testing controllers and tracking and sensor fusion algorithms.[1]

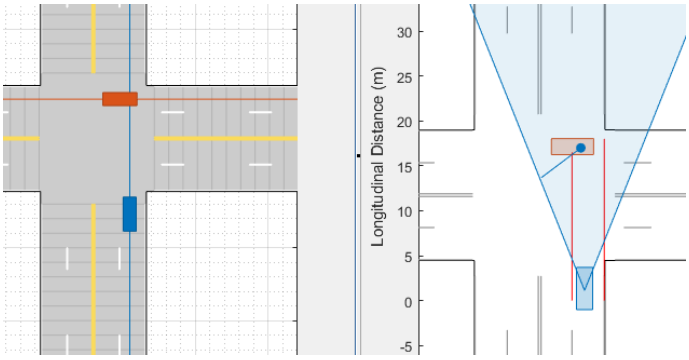

Fig 1: Left: Graphical representation of road topology, and vehicles and their trajectories, using the Driving Scenario Designer app. Right: Visualization during simulation showing moving actors, sensor range, and synthetic sensor data such as detected lanes, detected vehicles, and estimated velocity of detected vehicles.

Cuboid scenarios can be created by manually writing MATLAB code or with the Driving Scenario Designer app, which allows the user to:

- Create road and actor models using a drag-and-drop interface.
- Configure vision and radar sensors mounted on the ego vehicle, and use these sensors to simulate detections of actors and lane boundaries in the scenario.
- Load driving scenarios representing European New Car Assessment Programme (Euro NCAP®) test protocols[2][3][4] and other prebuilt scenarios.
- Import OpenDRIVE® roads and lanes into a driving scenario. The app supports OpenDRIVE format specification version 1.4H.[5]
- Export synthetic sensor detections to MATLAB.
- Generate MATLAB code of the scenario and sensors, and then programmatically modify the scenario and import it back into the app for further simulation.
- Generate a Simulink® model from the scenario and sensors, and use the generated models to test your sensor fusion or vehicle control algorithms.

B. 3D Simulation Environment

With this option, scenarios are rendered using the Unreal Engine® from Epic Games® and visualized using realistic graphics. Synthetic high-fidelity sensor data can be generated for radars, cameras, and lidars and used for testing path planning, vehicle control, perception algorithms, and perception-in-the-loop systems.[6]

Fig 2: 3D simulation environment with realistic graphics.

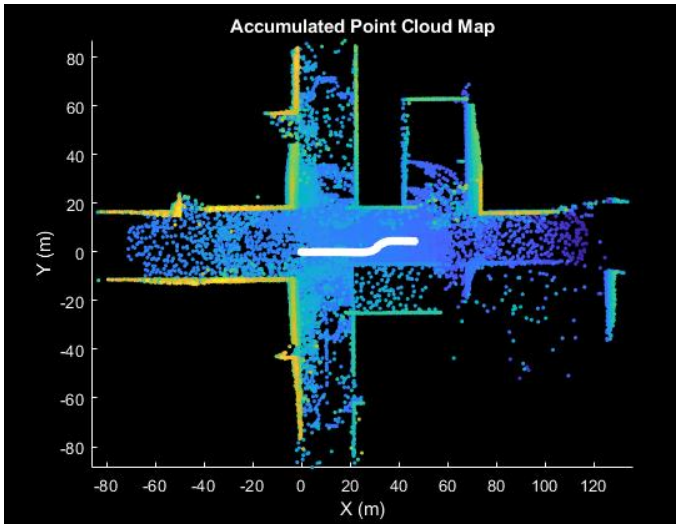

Fig 3: Example of lidar perception algorithm using synthetic sensor data from the 3D simulation environment. This algorithm creates and visualizes an accumulated map computed using synthetic data.

C. Closed-Loop Simulations

After a perception system has been tested and designed, it can be used as input to a control system that actually steers a vehicle. In this case, rather than manually set up a trajectory, the vehicle uses the perception system to drive itself. By combining perception and control into a closed-loop system, complex AD systems can be designed and tested.

Creating scenarios from logged data allows simulation of realistic traffic conditions. In the next section we show how to generate a virtual driving scenario from recorded vehicle data. The scenario is generated from position information recorded from a GPS sensor and recorded object lists processed from a lidar sensor.

II. CREATION OF SCENARIOS FROM LOGGED DATA

In the proposed example, we create a virtual driving scenario by generating a `drivingScenario` object containing data that was recorded from a test vehicle and an OpenDRIVE file. A complete description of the steps we have followed, including code and logged data, is available.[7]

The OpenDRIVE file describes the road network of the area where the data was recorded. The recorded vehicle data includes:

- GPS data**:** Text file containing the latitude and longitude coordinates of the ego vehicle at each timestamp.
- Lidar object list data**:** Text file containing the number of non-ego actors and the positions of their centers, relative to the ego vehicle, at each timestamp.
- Video data**:** MP4 file recorded from a forward-facing monocular camera mounted on the ego vehicle.

To generate and simulate the driving scenario, the following steps are proposed:

a. Explore recorded vehicle data.
b. Import OpenDRIVE road network into driving scenario.
c. Add ego vehicle data from GPS to driving scenario.
d. Add non-ego actors from lidar object list to driving scenario.
e. Simulate and visualize generated scenario.

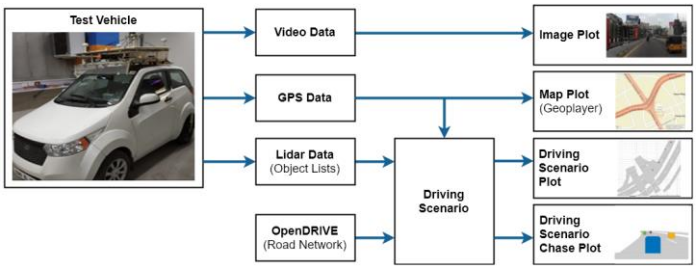

Fig 4: Diagram showing how recorded data has been used. The driving scenario is created from the GPS, lidar object lists, and OpenDRIVE files. The camera data is used to visualize the original scenario and compare this data with the generated scenario. The scenario route on the map can also be visualized using the `geoplayer` tool available in the MATLAB environment.

A. Exploring Recorded Vehicle Data

The positions of the ego vehicle were captured using a u-blox GPS NEO-M8N® sensor. The GPS sensor was placed on the center of the roof of the ego vehicle.

The positions of the non-ego actors were captured using a Velodyne® VLP-16 lidar sensor with a range of 30 meters. The VLP-16 sensor was placed on the roof of the ego vehicle at a position and height that avoids having reflections generated by the ego vehicle itself. The point cloud from the lidar sensor was processed on the vehicle to detect objects and their positions relative to the ego vehicle.

To help understand the original scenario, video from a monocular camera was recorded as a reference. This video can also be used to visually compare the original and generated scenarios.

Fig 5: Using video recorded from the camera mounted on the ego vehicle to visualize the preview of the urban traffic scenario.

Though the sensor coverage area can be defined around the entire ego vehicle, this example shows only the forward-looking scenario.

B. Importing OpenDRIVE Road Network into Driving Scenario

The road network data for generating the virtual scenario is obtained from OpenStreetMap®. The OpenStreetMap data files are converted to OpenDRIVE files and saved with extension `.xodr`. The `roadNetwork` function is used to import this road network data into a driving scenario:

```
scenario = drivingScenario;
openDRIVEFile = 'OpenDRIVEUrban.xodr';
roadNetwork(scenario,'OpenDRIVE', ...
    openDRIVEFile);
```

C. Adding Ego Vehicle Data from GPS to Generated Scenario

The ego vehicle data is collected from the GPS sensor and stored as a text file. The text file consists of three columns that store the latitude, longitude, and timestamp values for the ego vehicle. The structure we have used contains three fields specifying the latitude, longitude, and timestamps.

Trajectory waypoints of the ego vehicle are extracted from the recorded GPS coordinates. The `geodetic2enu` function[8] is used to convert the raw GPS coordinates to the local east-north-up Cartesian system. The transformed coordinates define the trajectory waypoints of the ego vehicle.

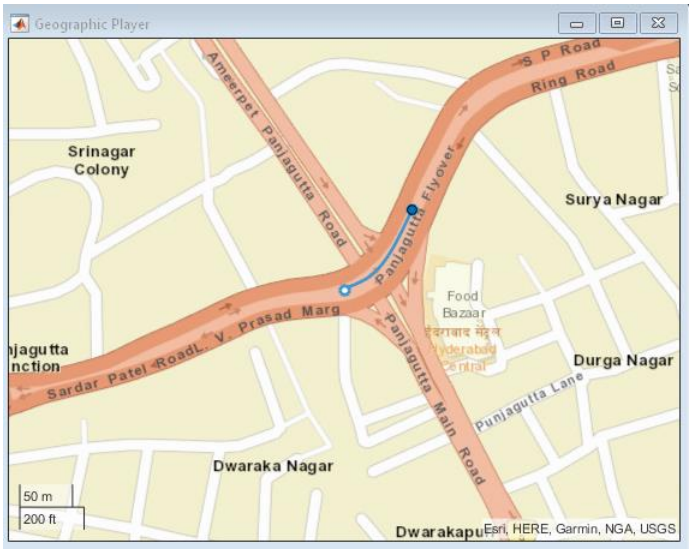

Fig 6: Scenario route map visualized with geographic player function.[9]

Speed and heading angle values of the ego vehicle are computed at each sensor data timestamp and used to create a trajectory for the ego vehicle. The ego vehicle follows the trajectory at the specified speed.

D. *Adding Non-Ego Actors from Lidar Object Lists to Generated Scenario*

The non-ego actor data is collected from the lidar sensor and stored as a text file. The text file consists of five columns that store the actor IDs, `x`-positions, `y`-positions, `z`-positions, and timestamp values, respectively. Non-ego actor data is imported into the MATLAB workspace and structured as follows:

1. `ActorID`: Scenario-defined actor identifier, specified as a positive integer.
2. `Position`: Position of actors relative to ego vehicle, specified as an [*x y z*] real vector. Units are in meters.
3. `Time`: Timestamp of the sensor recording.

These values are used to compute the trajectory waypoints and the speed of each non-ego actor at each timestamp. The trajectory waypoints are computed relative to the ego vehicle.

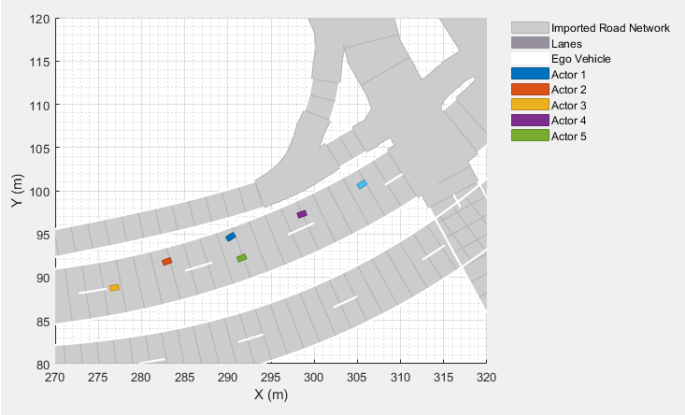

Fig 7: Visualization of the ego vehicle and non-ego actors imported into the generated scenario.

E. *Simulating and Visualizing Generated Scenario*

As a final step, the scenario can be simulated to visualize the road topology and moving actors following their respective trajectories. Animated scenario simulations can be compared with the reference video recorded with the monocular camera.

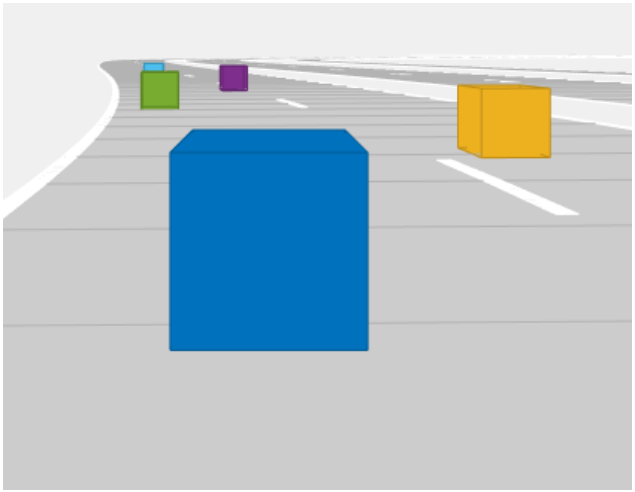

Fig 8: Chase camera view when synthetic scenario is running.

III. PROGRAMMATIC CREATION OF SCENARIO VARIATIONS

Once a virtual scenario has been created, we can easily obtain variations by programmatically changing actor parameters such as speed, dimensions, and radar cross section.

In this section, we describe the following steps to create programmatic variations of a driving scenario:

1. Export a MATLAB function that generates the MATLAB code that is equivalent to the original scenario.
2. Modify the exported function to create variations of the original scenario.
3. Call the function to generate a `drivingScenario` object that represents the scenario.
4. Import the scenario object into the Driving Scenario Designer app to simulate the modified scenario or generate additional scenarios. Alternatively, to simulate the modified scenario in Simulink, import the object into a Simulink model by using a Scenario Reader block.

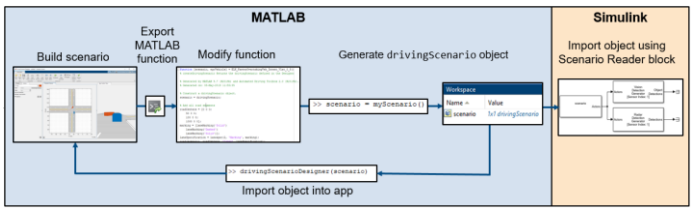

Fig 9: Workflow for programmatical creation of scenario variations.[10]

A. *Export MATLAB Function of Scenario*

After the scenario has been viewed and simulated, it can be exported to the MATLAB command line by using the export MATLAB function option available in the Driving Scenario Designer app. The created function contains the following information:

`scenario`: Roads and actors of the scenario

`egoVehicle`: Ego vehicle defined in the scenario

B. *Modify Function to Create Scenario Variations*

By modifying the code in the exported MATLAB function, multiple variations of a single scenario can be generated. One common option is to test the ego vehicle at different speeds. For example, in the exported MATLAB function, suppose that the speed of the ego vehicle is set to a constant value of 10 meters per second (speed = 10). To generate varying ego vehicle speeds, the speed variable can be converted into an input argument of the function by:

- Including ego speed as an input argument
- Deleting the constant variable `speed`

- Using `egoSpeed` instead of `speed` to compute the ego vehicle trajectory

To produce additional variations, consider:

- Modifying the road and lane parameters to view the effect on lane detections
- Modifying the trajectory or starting positions of the vehicles
- Modifying the dimensions of the vehicles

C. Call Function to Generate Programmatic Scenarios

Using the modified function, a variation of the scenario can be generated in which the environment and vehicles are described with different parameter values.

D. Import Modified Scenario into Driving Scenario App

The `drivingScenarioDesigner` function can be used to import the modified scenario into the app by specifying the `drivingScenario` object as an input argument.

Import Modified Scenario into Simulink

The Scenario Reader block can import the modified scenario into a Simulink model. This block reads the roads and actors from either a scenario file saved from the app or a `drivingScenario` variable saved to the MATLAB workspace or the model workspace. Once the Scenario Reader block has been added to a Simulink model and set to read the desired scenario variable, simulation results can be visualized with the Bird's-Eye Scope[11] and synthetic sensor data can be generated using radar detection generator or vision detection generator blocks.

IV. CONCLUSIONS

We have described how to create scenarios from vehicle-logged data and how to programmatically change scenario parameters for agile creation of new scenarios for testing automated driving functions under different traffic conditions.

REFERENCES

[1] https://mathworks.com/help/driving/ref/drivingscenariodesigner-app.html

[2] European New Car Assessment Programme. *Euro NCAP Assessment Protocol - SA*. Version 8.0.2. January 2018

[3] European New Car Assessment Programme. *Euro NCAP AEB C2C Test Protocol*. Version 2.0.1. January 2018

[4] European New Car Assessment Programme. *Euro NCAP LSS Test Protocol*. Version 2.0.1. January 2018

[5] Dupuis, Marius, et al. *OpenDRIVE Format Specification*. Revision 1.4, Issue H, Document No. VI2014.106. Bad Aibling, Germany: VIRES Simulationstechnologie GmbH, November 4, 2015

[6] https://mathworks.com/help/driving/ug/3d-simulation-for-automated-driving.html

[7] https://mathworks.com/help/driving/examples/scenario-generation-from-recorded-vehicle-data.html

[8] https://mathworks.com/help/map/ref/geodetic2enu.html

[9] https://mathworks.com/help/driving/ref/geoplayer.html

[10] https://mathworks.com/help/driving/ug/create-driving-scenario-variations-programmatically.html

[11] https://mathworks.com/help/driving/ref/birdseyescope.html