ADAS Kit Gazebo/ROS Simulator

Documentation for the Gazebo/ROS
simulation of the Dataspeed ADAS Kit.

Point of Contact:
(support@dataspeedinc.com)

Version: 1.2.0

Contents

SIMULATOR VERSION HISTORY

Version 1.2.0 (January 2018)

- Updated multi_robot_sim to demonstrate TF publishing and Velodyne point clouds.
- Simulator reports the production GPS output messages over the simulated CAN bus.
- Minor bug fixes.

Version 1.1.0 (October 2017)

- Added a simulated Velodyne VLP-16 LIDAR to the default URDF file.
- Implemented a much more true-to-life steering control dynamics model.

Version 1.0.5 (June 2017)

- Initial release.

1 Installation

The simulator supports Ubuntu 14.04 with ROS Indigo and Gazebo 2.4, and Ubuntu 16.04 with ROS Kinetic and Gazebo 7.0. To install the ADAS Kit simulator, first set up your computer to accept software from the Dataspeed server:

```
bash <(wget -q -O - https://bitbucket.org/DataspeedInc/ros_binaries/raw/default/scripts/setup.bash)
```

Then install the actual simulator package:

```
sudo apt-get install ros-$ROS_DISTRO-dbw-mkz-simulator
```

2 URDF Models

Four URDF models representing the different vehicles supported by the Dataspeed ADAS Kit are included in the simulation. These are shown in Figure 1. The TF trees of the simulation models are all the same, and this common TF tree is shown in Figure 2.

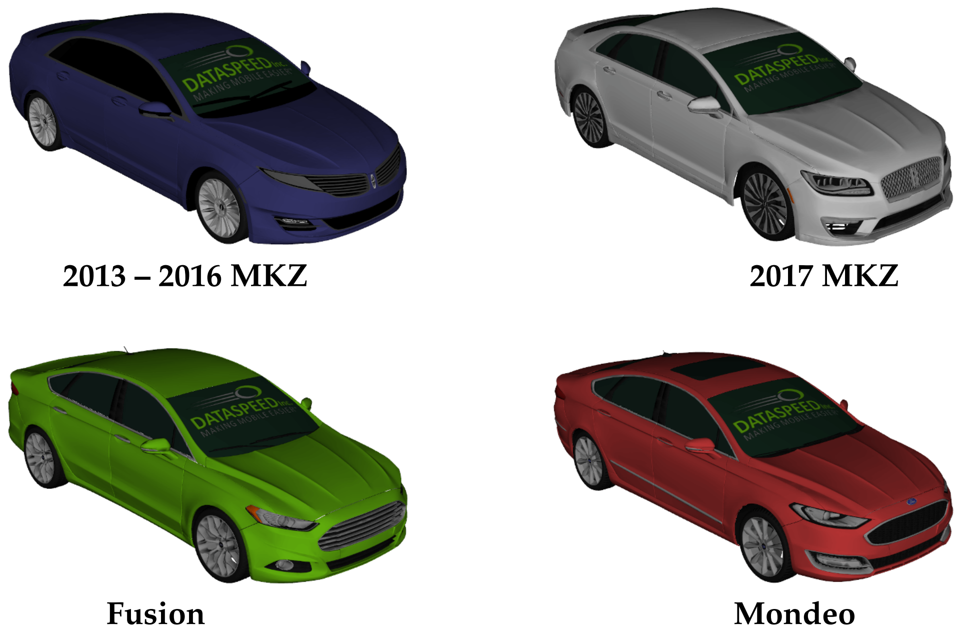

Figure 1: Models in ADAS Kit simulator.

2.1 URDF File Hierarchy

The detailed URDF files for these models can be found in the **dbw_mkz_description** package. The **vehicle_structure.urdf.xacro** and **vehicle_gazebo.urdf.xacro** files are included in the master file **mkz.urdf.xacro** in the **dbw_mkz_gazebo** package, which is the default file that is parsed to load the model.

2.2 Customizing the URDF Model

The default master URDF file contains three example sensors mounted on the simulated vehicle: a perfectly accurate GPS, a front-facing camera, and a Velodyne VLP-16 LIDAR. The Velodyne sensor is imported from the public **velodyne_simulator** ROS package (https://bitbucket.org/dataspeedinc/velodyne_simulator).

The example GPS and camera sensors are defined by xacro macros that can be found in **vehicle_sensors.urdf.xacro** in the **dbw_mkz_description** package. See Section 6 for more details about these example sensors.

To change the simulated sensors mounted on the vehicle, it is recommended to create a new master URDF file somewhere else on the filesystem, modify it as desired, and then pass the path to this new master URDF file to **dbw_mkz_gazebo.launch** as an argument. See Section 4 for all the arguments of this launch file.

When creating a custom master URDF file, be sure to include lines 6 and 7 from the default file:

```
<xacro:include filename="$(find dbw_mkz_description)/urdf/vehicle_structure.urdf.xacro" />
<xacro:include filename="$(find dbw_mkz_description)/urdf/vehicle_gazebo.urdf.xacro" />
```

These two xacro includes define the visual, collision, and inertial properties of the model, and contain the Gazebo properties and plugins necessary for proper simulation behavior. Any other xacro or URDF content can then be added to implement custom sensor configurations for simulation.

3 Simulated CAN Message Interface

The simulator emulates the CAN message interface to the real ADAS Kit. Therefore, there are only two ROS topics used to interact with the simulated vehicle: **can_bus_dbw/can_tx** to send CAN messages to the vehicle, and **can_bus_dbw/can_rx** to receive feedback data from the vehicle. These topics and their corresponding message types are listed in Table 1.

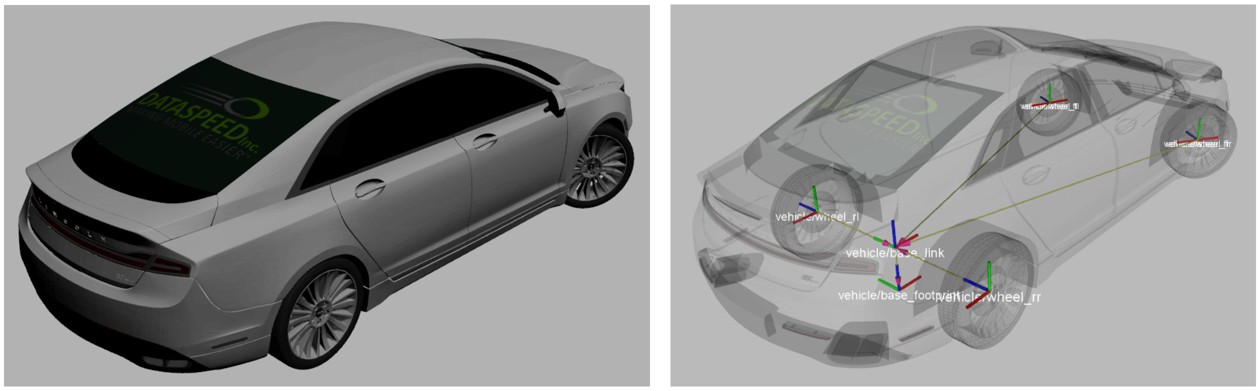

Figure 2: Simulation model and corresponding TF tree.

The simulator only implements a subset of the complete Dataspeed CAN message specification. The supported command messages are listed in Table 2, and the supported report messages are listed in Table 3. See the ADAS Kit datasheets for complete CAN message information.

Topic Name	Msg Type
~<name>/can_bus_dbw/can_rx	can_msgs/Frame
~<name>/can_bus_dbw/can_tx	can_msgs/Frame

Table 1: CAN message topics to interact with simulated ADAS Kit.

Command Msg	CAN ID
Brake	0x060
Throttle	0x062
Steering	0x064
Gear	0x066

Table 2: Command CAN messages supported by the ADAS Kit simulator.

Report Msg	CAN ID	Data Rate
Brake	0x061	50 Hz
Throttle	0x063	50 Hz
Steering	0x065	50 Hz
Gear	0x067	20 Hz
Misc	0x069	50 Hz
Wheel Speed	0x06A	100 Hz
Accel	0x06B	100 Hz
Gyro	0x06C	100 Hz
GPS1	0x6D	1 Hz
GPS2	0x6E	1 Hz
GPS3	0x6F	1 Hz
Brake Info	0x074	50 Hz

Table 3: Report CAN messages supported by the ADAS Kit simulator.

4 Launch Files

The following launch files are provided in the launch folder of the **dbw_mkz_gazebo** package.

4.1 dbw_mkz_gazebo.launch

dbw_mkz_gazebo.launch is the main launch file used to start the simulator. There are four arguments to this launch file:

- **use_camera_control** – This is a boolean argument that enables or disables a Gazebo plugin to automatically move the camera to follow a given model in the world. If unspecified, this argument defaults to true. See Section 7 for how the camera control plugin works.
- **world_name** – This is a path to a world file to load at startup. If unspecified, an empty world will be used. When saving a world file for future simulations, be sure to delete any vehicle models present in the world.
- **sim_param_file** – This is a path to a YAML file containing simulation configuration parameters. If unspecified, the **default_sim_params.yaml** file in the **dbw_mkz_gazebo** package is used. See Section 5 for formatting details and the available options.
- **urdf_file** – This is a path to the master URDF file for the simulated vehicle. If unspecified, the default master URDF file **mkz.urdf.xacro** in the **dbw_mkz_gazebo** package is used. See Section 2.2 for how this URDF file should be structured.

4.2 gazebo_world.launch

The main launch file **dbw_mkz_gazebo.launch** includes **gazebo_world.launch** to start up Gazebo, load a specified world file, and run the camera control plugin.

4.3 joystick_demo_sim.launch

The **joystick_demo_sim.launch** file simulates the same joystick demo program that can be run with the real ADAS Kit. It is a standalone launch file that brings up all necessary nodes at once. A screenshot of the default joystick demo world is shown in Figure 3.

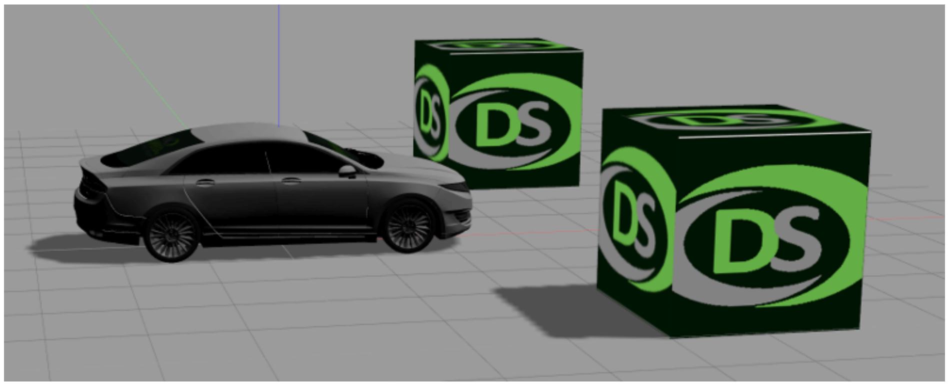

Figure 3: Joystick demo simulation world.

4.4 multi_car_sim.launch

The **multi_car_sim.launch** file demonstrates how multiple vehicles can be simulated at the same time. It is a standalone launch file that spawns both a Fusion and an MKZ model and publishes separate speed and yaw rate commands to each one. A screenshot of the example multi-car simulation is shown in Figure 4.

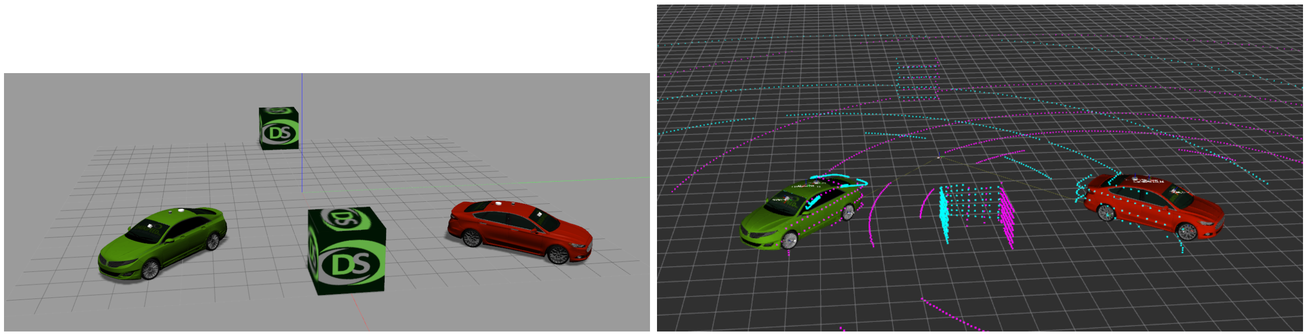
Figure 4: Example multi-car simulation.

4.5 lane_keep_demo.launch

The **lane_keep_demo.launch** file demonstrates the example camera sensor, as well as the simple road section models described in Section 8. It is a standalone launch file that brings up the simulation, an image processing pipeline to extract lane lines, and a path following control algorithm. The image processing pipeline finds the center of the lane, and the path following controller generates brake, throttle, and steering commands to follow the lane. A screenshot of the lane keeping simulation is shown in Figure 5.

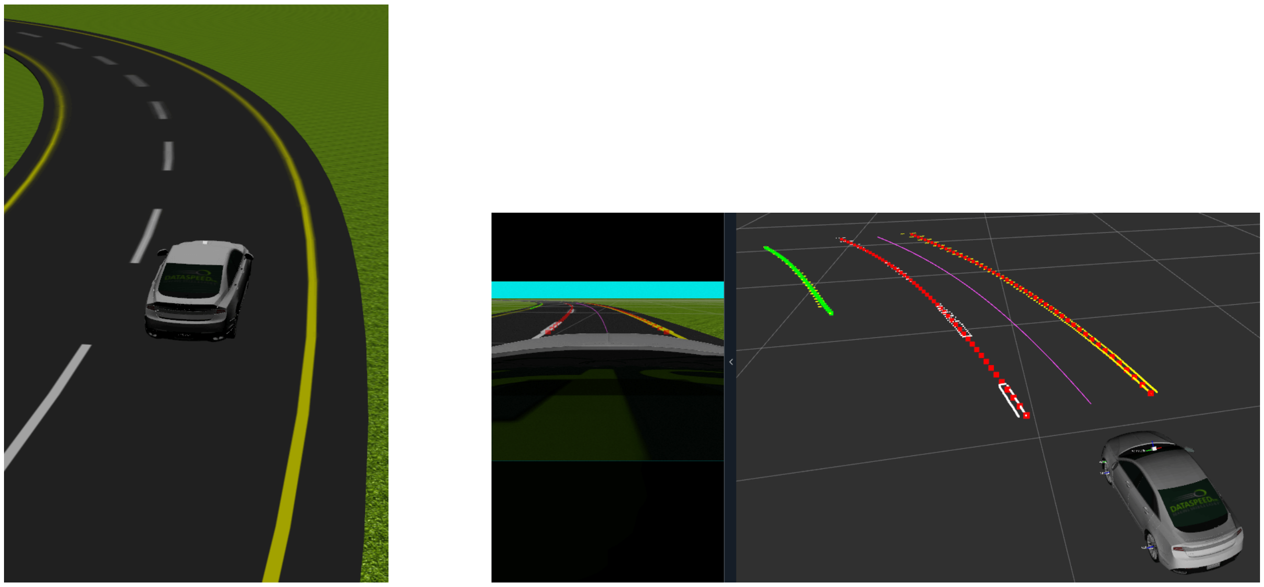
Figure 5: Gazebo and Rviz displays while running the example lane keeping simulation.

5 Configuration YAML Files

The parameters of the ADAS Kit Gazebo simulation are set using a single YAML file. This section describes the options and formatting of the YAML file.

5.1 Available Parameters

The parameters shown in Table 4 can be specified in a YAML file to control the behavior of a simulated vehicle.

Param Name	Default	Description
x	0.0	x coordinate of the spawn location of the model.
y	0.0	y coordinate of the spawn location of the model.
z	0.0	z coordinate of the spawn location of the model.
yaw	0.0	Initial heading angle of the vehicle, relative to Gazebo world frame.
start_gear	′P′	Single character indicating the starting transmission gear. Must be one of ′P′, ′R′, ′N′, ′D′, or ′L′. Others will result in the default of ′P′.
color	′silver′	Color of the vehicle body. See Section 5.4 for details on the behavior of this parameter.
pub_odom	false	Publish a nav_msgs/Odometry message with the true pose and velocity of the vehicle in Gazebo world frame.
pub_tf	false	Publish a tf transform from the footprint frame of the vehicle to ′world′ frame using the true pose of the vehicle in Gazebo world frame.
model	′mkz′	Model of vehicle to spawn. Must be one of ′mkz′, ′fusion′, or ′mondeo′. Others will result in the default of ′mkz′.
year	2013	Year of vehicle to spawn. Supported years are 2013 through 2017. Outside of this range will saturate between 2013 and 2017.

Table 4: ADAS Kit simulation configuration parameters.

5.2 Dictionary Format

The parameters listed in Table 4 must be set in an array of YAML dictionaries, where each dictionary corresponds to a single simulated vehicle. The name of the dictionary becomes the name of the spawned model. Consider the following YAML file:

```yaml
- vehicle:
  x: 5.0
  y: 20.0
  z: 0.0
  yaw: 0.0
  start_gear: D
```

This would simulate a single 2013 MKZ whose model name is **vehicle**. It would spawn at coordinates $(5.0,\ 20.0,\ 0.0)$ with a heading of 0.0, and the transmission would start in drive. The unspecified parameters would be the default values shown in Table 4.

5.3 Simulating Multiple Vehicles

To simulate multiple vehicles simultaneously, simply add more dictionaries to the array in the YAML file. Below is an example:

```yaml
- vehicle1:
  x: 0.0
  y: -2.0
  color: red
  model: mkz
  year: 2017
- vehicle2:
  x: 0.0
  y: 2.0
  color: green
  model: fusion
```

This would spawn two vehicles: one red 2017 MKZ with model name **vehicle1** spawned at $(0.0,\ -2.0,\ 0.0)$ and one green 2013 Fusion with model name **vehicle2** spawned at $(0.0,\ 2.0,\ 0.0)$. Both vehicles would have the default values of the parameters not set in the individual dictionaries.

5.4 Body Color

The **color** parameter is used to change the paint color of the simulated vehicle. It can be set as either a string or a dictionary with entries **r**, **g**, and **b**. Using a string selects between a number of preset colors, which are:

silver   red   green   blue   pink   white   black

Any other color strings will result in the default silver color. However, if the color parameter is set using a dictionary, any arbitrary color can be used. Below is an example YAML file that spawns one vehicle with the preset green color, and another with r=0.1, g=0.2, b=0.3:

```
– mkz1:
color: green
– mkz2:
color: {r: 0.1, g: 0.2, b: 0.3}
```

If the color dictionary is malformed, the default silver color is used instead.

5.5 Differences Between Different Models and Years

There are some differences between the various car models and years that the real ADAS Kit supports. In particular, the wheel speed sensors are different between different models, and some models have restrictions in the transmission shifting control.

Therefore, the simulator is designed to emulate some of these differences. However, at this time only the subtle wheel speed sensor differences are implemented, where all MKZ models report unsigned wheel speed measurements with a 0.8 rad/s deadband, and all Fusion and Mondeo models report signed wheel speed measurements with a 0.2 rad/s deadband. The shifting limitations are an anticipated feature that will be implemented in a future release, and is discussed more in Section 9.

6 Example Sensors

A GPS and front-facing camera are included with the simulator as examples. They are defined by xacro macros that are found in **vehicle_sensors.urdf.xacro** in the **dbw_mkz_description** package. When included, the example GPS sensor macro adds a white disk where it is mounted, and the example camera macro adds a white box. This is shown in Figure 6.

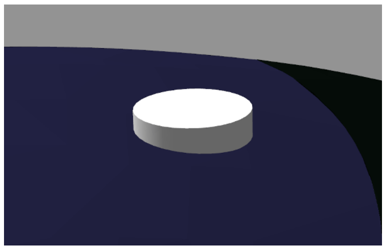
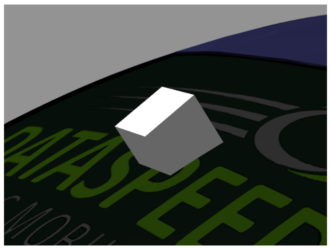

Figure 6: Visual models of the example GPS sensor (left), and the example camera (right).

6.1 Example GPS Receiver

The example GPS sensor built into the ADAS Kit simulator provides perfectly accurate GPS position and heading measurements based on the vehicle model's position in the Gazebo world frame. To use the example sensor, it is recommended to use the xacro macro as shown in the main URDF file **mkz.urdf.xacro**. The macro's parameters are described in Table 5. The GPS simulation plugin provides data on a collection of ROS topics that are advertised in the plugin's namespace. These topics are outlined in Table 6.

Param Name	Type	Units	Description
name	string	—	Name of the sensor. This defines the link name of the sensor, its corresponding TF frame name, and the namespace of the simulated data topics.
parent_link	string	—	Name of the link to which the sensor is attached.
x	float	meters	x offset of the sensor link, relative to **parent_link**.
y	float	meters	y offset of the sensor link, relative to **parent_link**.
z	float	meters	z offset of the sensor link, relative to **parent_link**.
rate	float	Hz	Rate at which to publish simulated GPS position and heading measurement messages.
ref_lat	float	degrees	Latitude corresponding to point $(0, 0, 0)$ in Gazebo world frame.
ref_lon	float	degrees	Longitude corresponding to point $(0, 0, 0)$ in Gazebo world frame.

Table 5: Gazebo GPS plugin parameters.

Topic Name	Msg Type	Description
~<name>/fix	sensor_msgs/NavSatFix	Simulated position in latitude, longitude, altitude.
~<name>/utm	geometry_msgs/Vector3Stamped	Simulated position in UTM coordinates. Zone and hemisphere boundary transitions are simulated.
~<name>/vel	geometry_msgs/Vector3Stamped	Simulated velocity vector. x velocity component is relative to Due East, y velocity component is relative to due North.
~<name>/heading	std_msgs/Float64	Simulated heading angle in NED frame. Angle is represented in degrees and wrapped between 0 and 360.

Table 6: Simulated GPS measurement ROS topics.

6.2 Example Camera

The example camera sensor built into the ADAS Kit simulator is a xacro macro that runs a standard Gazebo camera simulation. Besides creating the simulated camera, the xacro macro sets up URDF links for the camera sensor and corresponding optical frame, and provides parameters that allow the camera to be easily attached to the main vehicle's URDF model. The macro parameters are listed in Table 7.

Param Name	Type	Units	Description
name	string	—	Name of the sensor. This defines the link name of the sensor, its corresponding TF frame name, and the namespace of the simulated data topics.
parent_link	string	—	Name of the link to which the sensor is attached.
x	float	meters	x offset of the sensor link, relative to **parent_link**.
y	float	meters	y offset of the sensor link, relative to **parent_link**.
z	float	meters	z offset of the sensor link, relative to **parent_link**.
roll	float	radians	Roll angle relative to parent.
pitch	float	radians	Pitch angle relative to parent.
yaw	float	radians	Yaw angle relative to parent.

Table 7: Gazebo camera plugin xacro macro parameters.

7 Camera Control Plugin

The camera control plugin is used to automatically follow a model in the Gazebo world as it moves. The behavior of the plugin is controlled using **dynamic_reconfigure**. The reconfigurable parameters for the plugin are shown in Table 8.

Param Name	Type	Description
enable	bool	Enable automatic camera control. If this is unchecked, the Gazebo camera behaves normally.
model_name	string	This is the name of the Gazebo model to follow with the camera. If the model name doesn't exist or isn't spawned yet, the Gazebo camera behaves normally.
view_dist	float	This is the following distance of the Gazebo camera from the target model. The camera orientation can be changed manually, but the distance from the target model will be fixed to this value.

Table 8: Dynamic reconfigure parameters for the camera control Gazebo plugin.

8 Gazebo Models

To set up simple road scenarios, some modular road segment models are included with the simulator and can be inserted into the world. The different road segments are shown in Figure 7, where there are 50, 100, and 200 meter straight road segments; 50 and 100 meter radius curve segments; and an intersection. An example road setup is shown in Figure 8, which is part of the simulated track used in the lane keep demo addressed in Section 4.5.

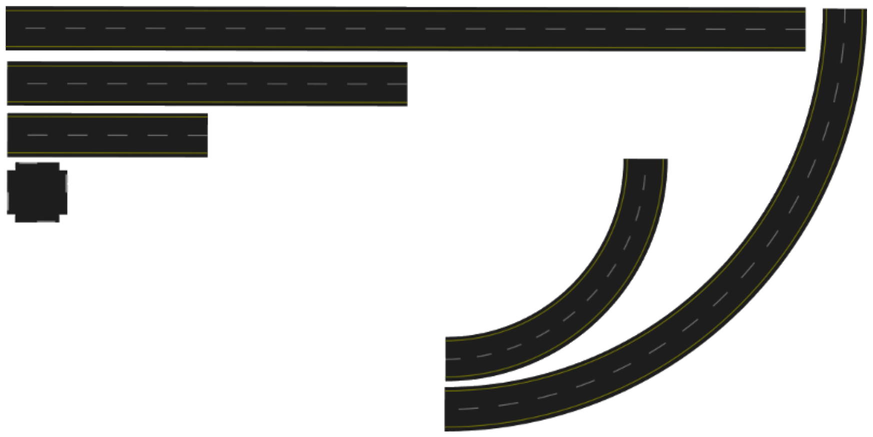

Figure 7: Modular road segment models to build simple simulation scenarios.

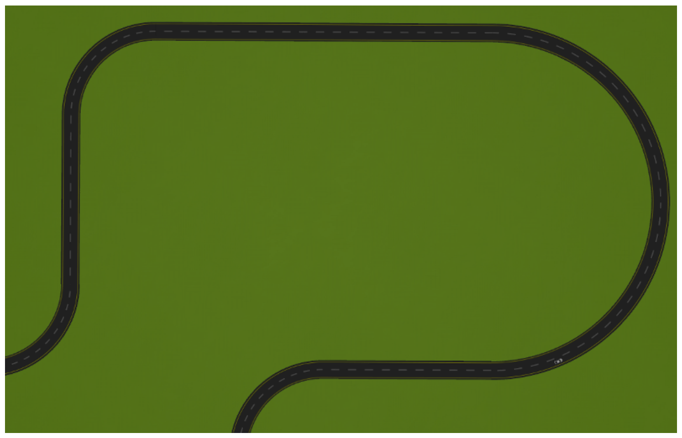

Figure 8: Example configuration of modular road section models.

9 Anticipated Future Features

9.1 GUI to Simulate Physical Vehicle Inputs from Driver

At this time, the vehicle only responds to commands received over the simulated CAN interface. Therefore, the physical pedals, steering wheel, and shifter are not simulated at all. This means that the driver override behavior of the real ADAS Kit is also not simulated.

The next release will include a GUI that can be used to provide simulated inputs from the physical system. With this GUI, the driver override behavior can be observed and tested in the simulator.

9.2 Different Shifting Restrictions Based on Model and Year

Some of the models and years of vehicles that are supported by the real ADAS Kit have certain limitations on the control of the shifter. For example, MKZ models with a pushbutton shifter allow full control using the Dataspeed drive-by-wire command messages. However, newer Fusion models have a rotary shifter that prevents autonomous shifting from park, while older Fusion models with a physical shifting lever cannot be controlled at all.

In the next release, these different shifting limitations will be implemented depending on the model and year of the car being simulated. However, this requires the physical input simulation GUI feature from Section 9.1 so the user can shift gears like a human driver would in the real vehicle.