755 Phoenix Drive, Ann Arbor MI, 48108, USA
Phone: 734 668-2930 • Fax: 734 668-2877 • Email: info@carsim.com
carsim.com

Camera Sensor Example

This memo describes an example simulation setup that shows how to connect a vehicle model from CarSim, TruckSim or BikeSim with VS Visualizer and Simulink, such that live camera information from VS Visualizer is available in Simulink. In this example, Simulink image-processing functions are used to identify other vehicles as needed for some Advanced Driver Assistance Systems (ADAS) and autonomous driving scenarios.

Be aware that the example requires software from The MathWorks:

1. MATLAB/Simulink R2014b or newer is needed. (R2017a required for Pedestrian Detection Example.)
2. The Simulink model also requires image-processing libraries: *MATLAB Computer Vision System Toolbox* and *Image Processing Toolbox.*

Also, CarSim, TruckSim or BikeSim need the optional Sensors license.

Two Sensor Options in VehicleSim Products

CarSim, TruckSim, and BikeSim support two different means for detecting objects in support of simulation of ADAS and autonomous driving scenarios.

The first method for detecting objects is handled completely by the VS Solvers that perform the math model calculations for vehicle dynamics and controllers. The VS Solvers support targets that are rectangular or cylindrical. Vectors are calculated automatically to connect the on-board sensor with a few points on each target object (left edge, closest point, right edge). This method is relatively easy to use but is limited to simple object shapes.

The other method uses VS Visualizer to generate realistic images which are then connected to other software. This memo describes a vehicle detection example, a highway lane detection example, and a pedestrian detection example. These examples make use of image-processing capabilities in Simulink, running with the optional toolboxes mentioned above.

Overview of Image Processing

Image processing for autonomous vehicles and pre-crash safety systems has been used in prototypes and some commercial vehicles in the last several decades. The traditional way to study an image processing method related to vehicles was put a video camera on a test vehicle and obtain video data together with vehicle measurements such as speeds, accelerations, angular rate, etc. [1] The preparations for experiments and post processing of experimental data usually took massive time and effort.

These types of experiments are easily handled in CarSim, TruckSim, and BikeSim. VS Visualizer provides direct access to camera sensor data that can be used with image processing methods. The video image is generated by a *Live Video* of VS Visualizer whose camera position is specified at any location on the VS vehicle model or VS road. This image information can be combined with conventional vehicle dynamics output variables in external software such as Simulink.

In order to show animated video, VS Visualizer works with a simulated 3D world assembled with 3D shape files along with motion information from the VS Math Model. For normal video viewing, VS Visualizer renders one or more camera views as rectangular regions with colors set for each pixel. These capabilities have been extended in Camera Sensors to provide camera-based sensor information in shared memory buffers that can be used by “client” programs. Any number of Camera Sensors can be created and attached to any Reference Frame (or Transform) in the animation.

There are several ways to access the camera memory buffers. The example described in this memo uses Simulink with *Computer Vision System* and *Image Processing Toolbox.* A second interface is available from a MATLAB MEX function. A third interface (used to create the others) is an ANSI-C and C++ interface provided via a DLL.

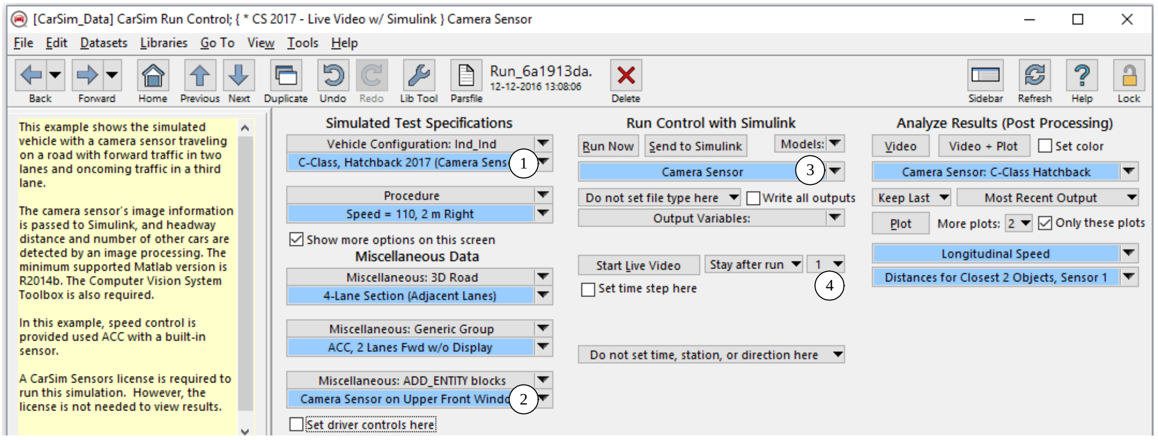
The VS Camera Sensor provides three types of information: color/visual, depth, and surface normal vectors. Camera data is computed per-pixel at arbitrary resolutions (independent of the main animation window).

Vehicle Detection Example

Camera Sensor Settings in the VS Datasets

CarSim includes an example simulation run named **Vehicle Detection (Simulink)** in the category **ADAS and Active Safety** of the **Run Control** screen (Figure 1). Similar examples are provided in TruckSim and BikeSim.

The camera sensor information is provided in two places. The location of the sensor is part of the vehicle dataset ① (Figure 1), where the coordinates are matched to the vehicle dimensions. The vehicle dataset (Figure 2) includes a link to an **Animator: Vehicles and Targets** dataset ①. The animator dataset (Figure 3) has a link ① to a dataset from the Generic VS Commands library with an `ADD_ENTITY` block for VS Visualizer (Figure 4).



[CarSim_Data] CarSim Run Control; { * CS 2017 - Live Video w/ Simulink } Camera Sensor
File Edit Datasets Libraries Go To View Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
Run_6a1913da.
12-12-2016 13:08:06
Delete Sidebar Refresh Help Lock
1
2
3
4

This example shows the simulated vehicle with a camera sensor traveling on a road with forward traffic in two lanes and oncoming traffic in a third lane.

The camera sensor's image information is passed to Simulink, and headway distance and number of other cars are detected by an image processing. The minimum supported Matlab version is R2014b. The Computer Vision System Toolbox is also required.

In this example, speed control is provided used ACC with a built-in sensor.

A CarSim Sensors license is required to run this simulation. However, the license is not needed to view results.

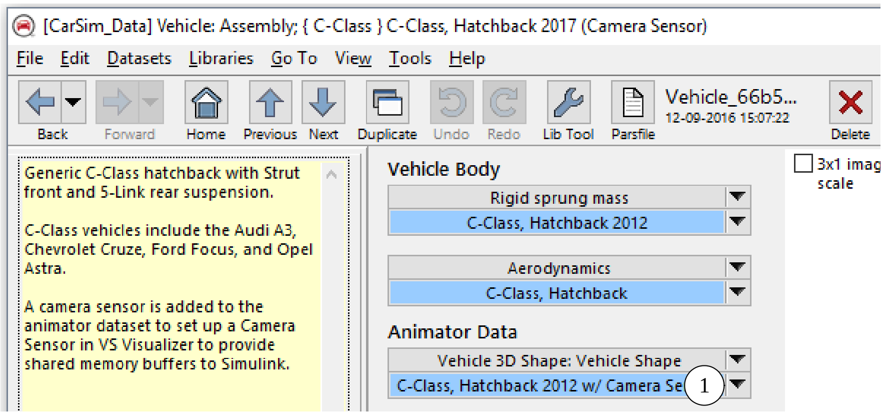
Simulated Test Specifications
- Vehicle Configuration: Ind_Ind — C-Class, Hatchback 2017 (Camera Sens
- Procedure — Speed = 110, 2 m Right
- ☑ Show more options on this screen

Miscellaneous Data
- Miscellaneous: 3D Road — 4-Lane Section (Adjacent Lanes)
- Miscellaneous: Generic Group — ACC, 2 Lanes Fwd w/o Display
- Miscellaneous: ADD_ENTITY blocks — Camera Sensor on Upper Front Windo
- ☐ Set driver controls here

Run Control with Simulink
- Run Now | Send to Simulink | Models:
- Camera Sensor
- Do not set file type here | ☐ Write all outputs
- Output Variables:
- Start Live Video | Stay after run | 1
- ☐ Set time step here
- Do not set time, station, or direction here

Analyze Results (Post Processing)
- Video | Video + Plot | ☐ Set color
- Camera Sensor: C-Class Hatchback
- Keep Last | Most Recent Output
- Plot | More plots: 2 | ☑ Only these plots
- Longitudinal Speed
- Distances for Closest 2 Objects, Sensor 1

Figure 1. Camera sensor example (Run Control screen).



[CarSim_Data] Vehicle: Assembly; { C-Class } C-Class, Hatchback 2017 (Camera Sensor)
File Edit Datasets Libraries Go To View Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
Vehicle_66b5...
12-09-2016 15:07:22
Delete
1

Generic C-Class hatchback with Strut front and 5-Link rear suspension.

C-Class vehicles include the Audi A3, Chevrolet Cruze, Ford Focus, and Opel Astra.

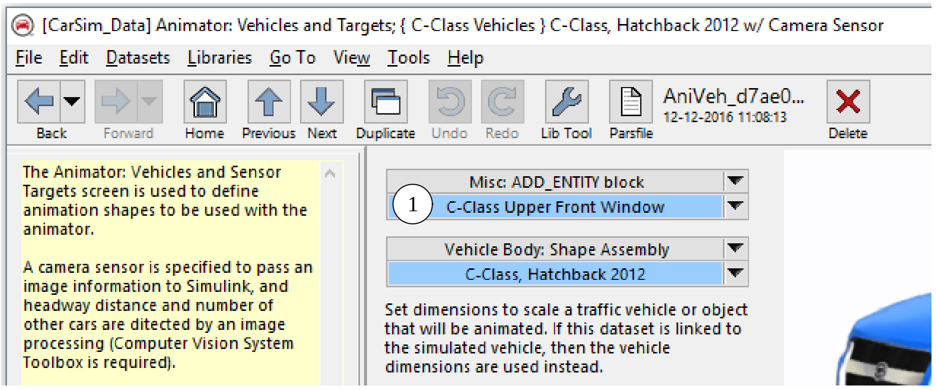
A camera sensor is added to the animator dataset to set up a Camera Sensor in VS Visualizer to provide shared memory buffers to Simulink.

Vehicle Body
- Rigid sprung mass — C-Class, Hatchback 2012
- Aerodynamics — C-Class, Hatchback

Animator Data
- Vehicle 3D Shape: Vehicle Shape — C-Class, Hatchback 2012 w/ Camera Se

☐ 3x1 imag scale

Figure 2. Vehicle: Assembly screen with link to Animator Data that locates a Camera Sensor.



[CarSim_Data] Animator: Vehicles and Targets; { C-Class Vehicles } C-Class, Hatchback 2012 w/ Camera Sensor
File Edit Datasets Libraries Go To View Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
AniVeh_d7ae0...
12-12-2016 11:08:13
Delete
1

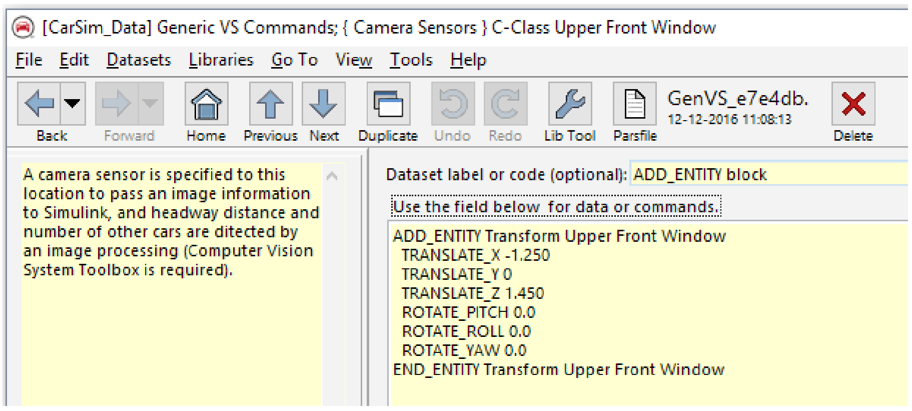
The Animator: Vehicles and Sensor Targets screen is used to define animation shapes to be used with the animator.

A camera sensor is specified to pass an image information to Simulink, and headway distance and number of other cars are detected by an image processing (Computer Vision System Toolbox is required).

- Misc: ADD_ENTITY block — C-Class Upper Front Window
- Vehicle Body: Shape Assembly — C-Class, Hatchback 2012

Set dimensions to scale a traffic vehicle or object that will be animated. If this dataset is linked to the simulated vehicle, then the vehicle dimensions are used instead.

Figure 3. Animator dataset with link to dataset with ADD_ENTITY blocks for VS Visualizer.


[CarSim_Data] Generic VS Commands; { Camera Sensors } C-Class Upper Front Window
File Edit Datasets Libraries Go To View Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile Delete
GenVS_e7e4db.
12-12-2016 11:08:13

A camera sensor is specified to this location to pass an image information to Simulink, and headway distance and number of other cars are detected by an image processing (Computer Vision System Toolbox is required).

Dataset label or code (optional): ADD_ENTITY block

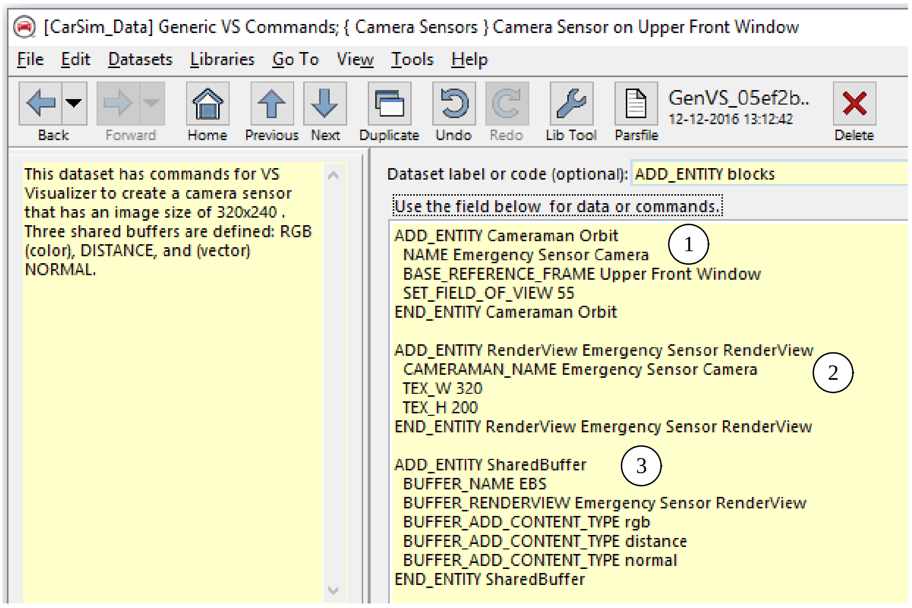
Use the field below for data or commands.

```
ADD_ENTITY Transform Upper Front Window
 TRANSLATE_X -1.250
 TRANSLATE_Y 0
 TRANSLATE_Z 1.450
 ROTATE_PITCH 0.0
 ROTATE_ROLL 0.0
 ROTATE_YAW 0.0
END_ENTITY Transform Upper Front Window
```

Figure 4. Transform (reference frame) for Camera Sensor.

The `ADD_ENTITY` block defines a Transform for VS Visualizer that makes use of the sprung mass reference frame, with relative coordinates for the camera set to a point at the top of the windshield, and pointing in the direction of the sprung mass X-axis.

A second dataset is used in this example to provide camera information that is not tied to the vehicle dimensions. The **Run Control** dataset (Figure 1) has a link to a set of `ADD_ENTITY` blocks (2) that are provided in another Generic VS Command dataset (Figure 5).


[CarSim_Data] Generic VS Commands; { Camera Sensors } Camera Sensor on Upper Front Window
File Edit Datasets Libraries Go To View Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile Delete
GenVS_05ef2b..
12-12-2016 13:12:42
1
2
3

This dataset has commands for VS Visualizer to create a camera sensor that has an image size of 320x240 . Three shared buffers are defined: RGB (color), DISTANCE, and (vector) NORMAL.

Dataset label or code (optional): ADD_ENTITY blocks

Use the field below for data or commands.

```
ADD_ENTITY Cameraman Orbit
 NAME Emergency Sensor Camera
 BASE_REFERENCE_FRAME Upper Front Window
 SET_FIELD_OF_VIEW 55
END_ENTITY Cameraman Orbit

ADD_ENTITY RenderView Emergency Sensor RenderView
 CAMERAMAN_NAME Emergency Sensor Camera
 TEX_W 320
 TEX_H 200
END_ENTITY RenderView Emergency Sensor RenderView

ADD_ENTITY SharedBuffer
 BUFFER_NAME EBS
 BUFFER_RENDERVIEW Emergency Sensor RenderView
 BUFFER_ADD_CONTENT_TYPE rgb
 BUFFER_ADD_CONTENT_TYPE distance
 BUFFER_ADD_CONTENT_TYPE normal
END_ENTITY SharedBuffer
```

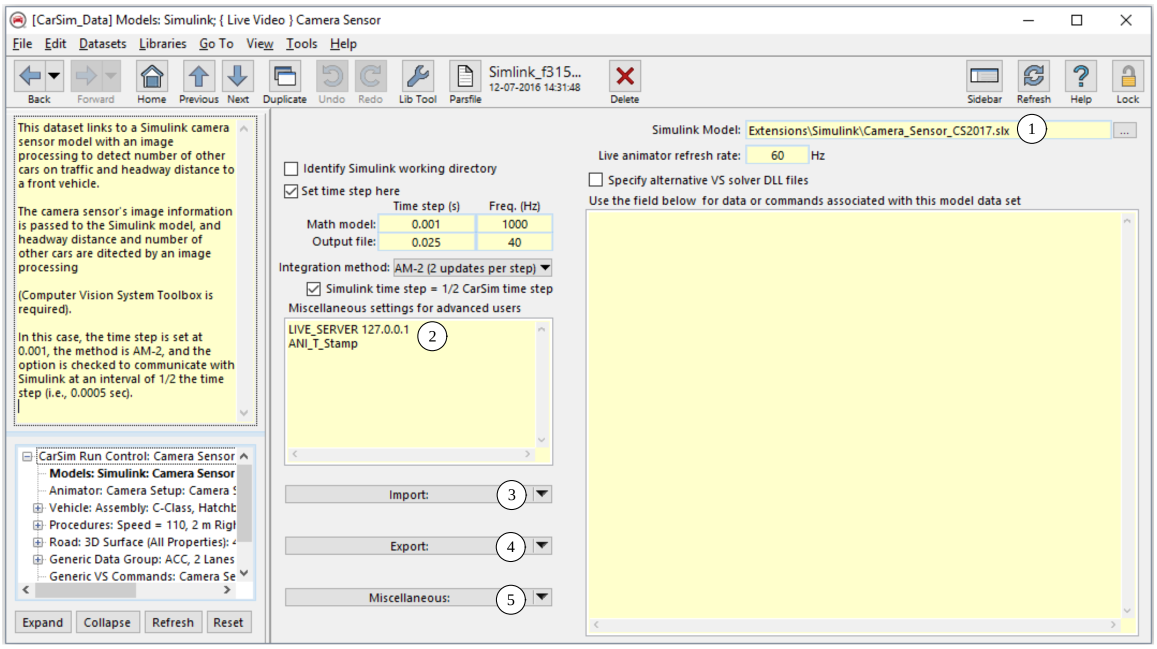
Figure 5. Camera sensor settings.

The camera sensor's field of view (1) and render size of image (width and height) (2) are specified. The render information is assigned to a buffer with a specified buffer name (root keyword: `BUFFER_NAME`) (3). The same buffer name (`EBS` for this example camera sensor) is used to identify the camera in Simulink. Therefore, multiple camera sensors are available with different buffer names.

The **Run Control** dataset (Figure 1, page 3) is set up for live animation. This option is available for all Simulink models. If you do not see the **Start Live Video** button and a drop-down control for the number of live cameras (4), then use the menu item **Tools > License Settings** to check the option for **Extra Live Animator**.

For this example, the number of live cameras should be set to 1.

The **Run Control** dataset has a link to a dataset from the **Models: Simulink** library (3), shown in Figure 6. Notice that there are no import/export variables linking the vehicle model to Simulink; the potential links ((3), (4), (5)) are not used. All information from the camera sensor to the Simulink model is automatic as part of the VS Camera S-Function.


[CarSim_Data] Models: Simulink; { Live Video } Camera Sensor

Simlink_f315...
12-07-2016 14:31:48

This dataset links to a Simulink camera sensor model with an image processing to detect number of other cars on traffic and headway distance to a front vehicle.

The camera sensor's image information is passed to the Simulink model, and headway distance and number of other cars are ditected by an image processing

(Computer Vision System Toolbox is required).

In this case, the time step is set at 0.001, the method is AM-2, and the option is checked to communicate with Simulink at an interval of 1/2 the time step (i.e., 0.0005 sec).

CarSim Run Control: Camera Sensor
Models: Simulink: Camera Sensor
Animator: Camera Setup: Camera S
Vehicle: Assembly: C-Class, Hatchb
Procedures: Speed = 110, 2 m Righ
Road: 3D Surface (All Properties): 4
Generic Data Group: ACC, 2 Lanes
Generic VS Commands: Camera Se

☐ Identify Simulink working directory
☑ Set time step here

	Time step (s)	Freq. (Hz)
Math model:	0.001	1000
Output file:	0.025	40

Integration method: AM-2 (2 updates per step)
☑ Simulink time step = 1/2 CarSim time step
Miscellaneous settings for advanced users

```
LIVE_SERVER 127.0.0.1
ANI_T_Stamp
```

Simulink Model: Extensions\Simulink\Camera_Sensor_CS2017.slx (1)
Live animator refresh rate: 60 Hz
☐ Specify alternative VS solver DLL files
Use the field below for data or commands associated with this model data set

Figure 6. Models: Simulink dataset.

This dataset provides two essential pieces of information. One is the pathname for the Simulink model (1), and the other is a command to connect the Simulink S-Function to VS Visualizer for live video. The connection is specified using the `LIVE_SERVER` command in the miscellaneous field (2):

```
LIVE_SERVER 127.0.0.1
```

Running the Simulink Model

View the Simulink model by clicking the **Send to Simulink** button (3) from the **Run Control** dataset. Because the simulation will involve live animation, the first window that appears is for VS Visualizer. Other windows appear for the Simulink model (Figure 7) and related windows. This example makes use of two S-Functions provided with CarSim, TruckSim, and BikeSim.

CarSim, TruckSim, and BikeSim each include a library of four S-Functions. Figure 8 shows the library for CarSim. The S-Functions for TruckSim and BikeSim have different icons, but are otherwise identical to the CarSim functions shown here.

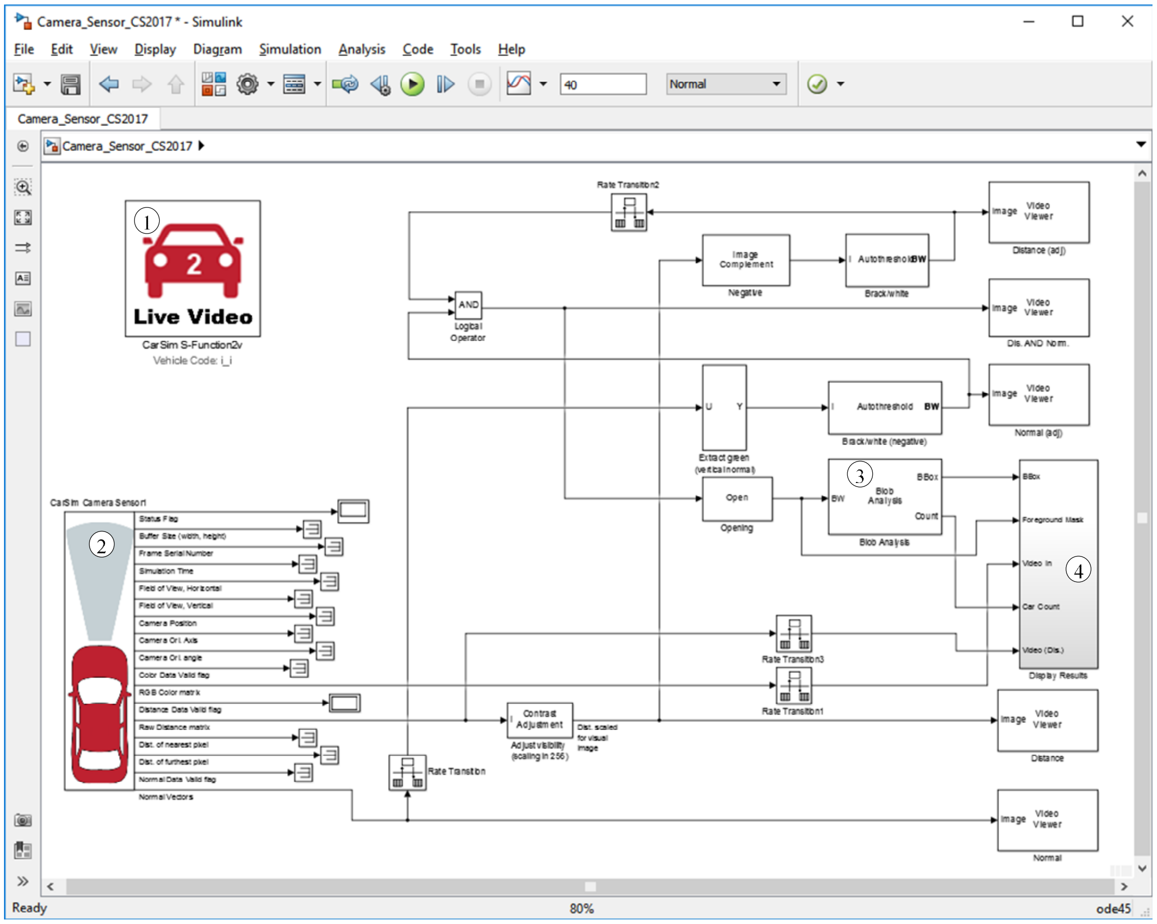


Figure 7. Example camera sensor model in Simulink (shown with CarSim).

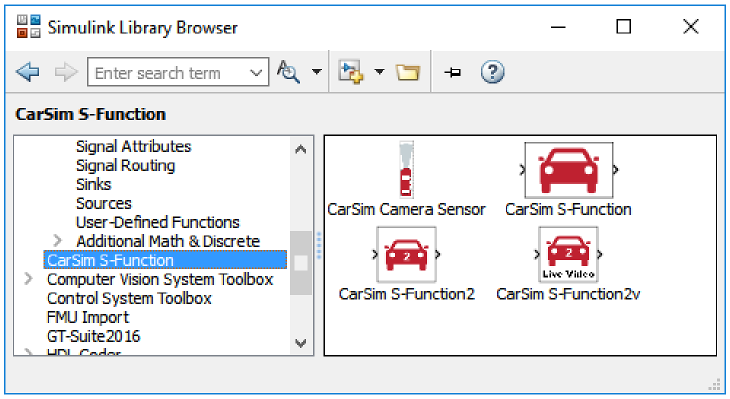


Figure 8. VS S-Function blocks in the Simulink Library Browser.

The S-Function whose name ends with "S-Function2v" (e.g., `CarSim S-Function2v` ①) connects the vehicle model to Simulink and supports a live connection with VS Visualizer, such that VS Visualizer receives motion variables from the vehicle models as needed to generate live video involving those motions.

The S-Function whose name ends with "Camera Sensor" (e.g., `CarSim Camera Sensor` ②) connects a shared memory buffer from VS Visualizer with Simulink.

The output signals from the VS Camera Sensor block are all hard-wired as described in a separate Technical Memo, VS Camera Sensor S-function.

Image Processing for a Camera Sensor

Click the **Start** button for the example Simulink model. In addition to the Simulink model window and the live video window that appeared earlier, there should be seven more animation windows (Figure 9). The Results window (Figure 10) shows the 320x200 camera image with detected cars indicated by green rectangles. It also displays the number of detected cars and headway distance to the front vehicle.

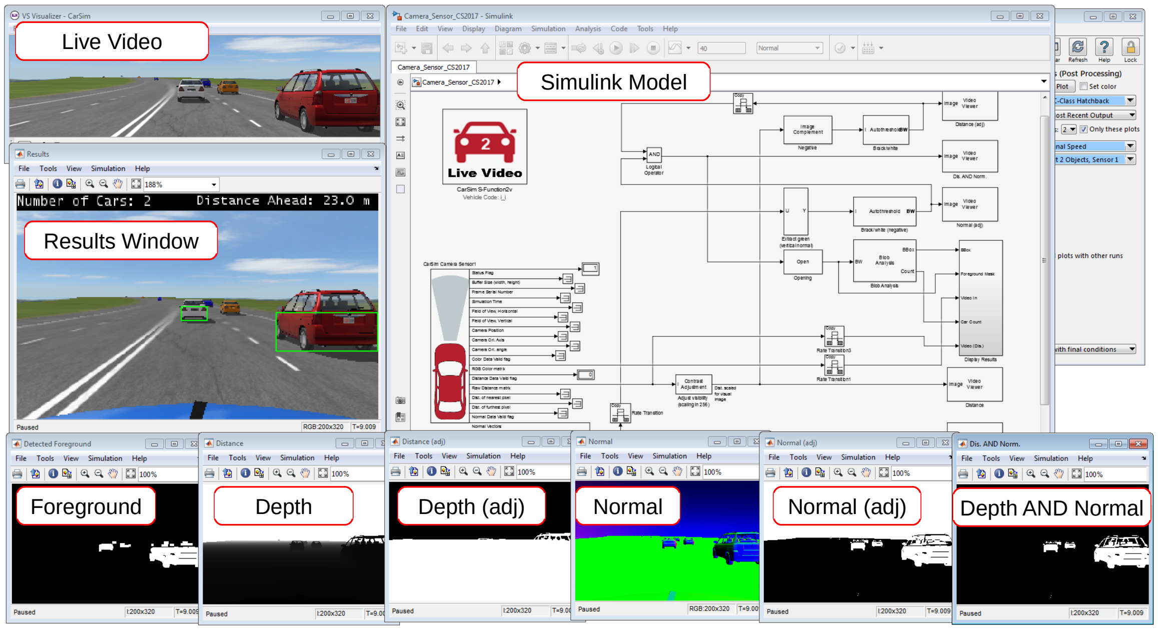


Figure 9. The Simulink model and eight animation windows.

In this example model, the RGB color matrix, raw depth matrix and normal vectors from the VS camera sensor block are used to get this image through several image processes. The image shown in Figure 10 is an overlay of the RGB color image with detected green areas and some other information.

The analysis involves six processed images, all shown in Figure 9:

1. The normal vectors from the VS camera sensor (Figure 11) are separated into vertical normal vector part and non-vertical normal part. This is done by simply extracting with the green color, and changing to a black/white image as shown in Figure 12. The image in Figure 12 implies anything facing the camera is white (1) and otherwise is black (0).

2. The depth matrix (Figure 13) is transformed to a negatively flipped black/white image (0 or 1) by a threshold as shown in Figure 14. This image implies anything in close depth is white (1) and further depth is black (0).

3. The adjusted depth image (Figure 14) and adjusted normal image (Figure 12) are combined with "AND" circuit (Figure 15); this combining implies anything in closer depth with facing to the camera is the traffic cars shown in white (1).

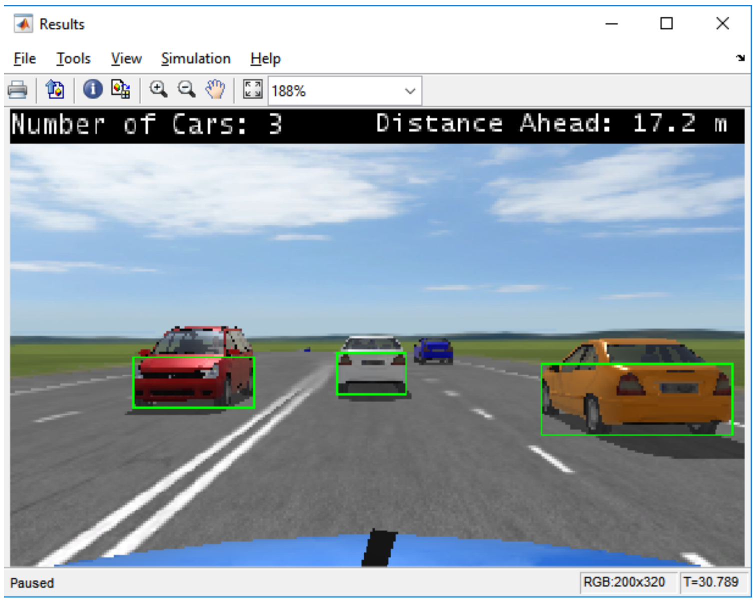

Results
File Tools View Simulation Help
188%
Number of Cars: 3 Distance Ahead: 17.2 m
Paused
RGB:200x320
T=30.789

Figure 10. Front camera image with detected traffic cars indicated by green rectangles.

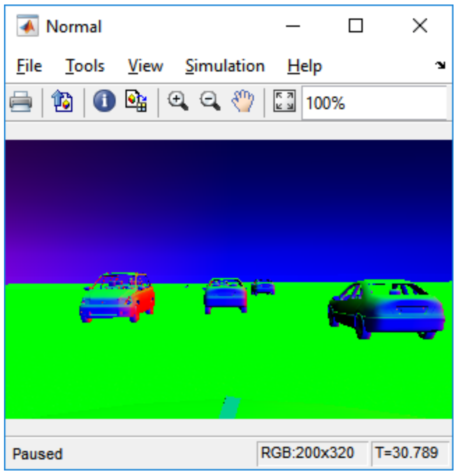


Figure 11. Normal vectors from the VS camera sensor.

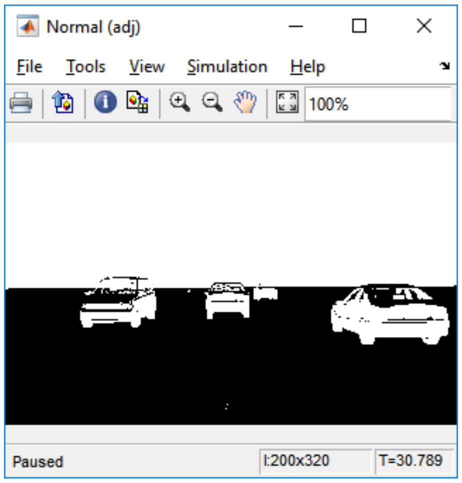


Figure 12. Black/white image showing vertical facing areas in white.

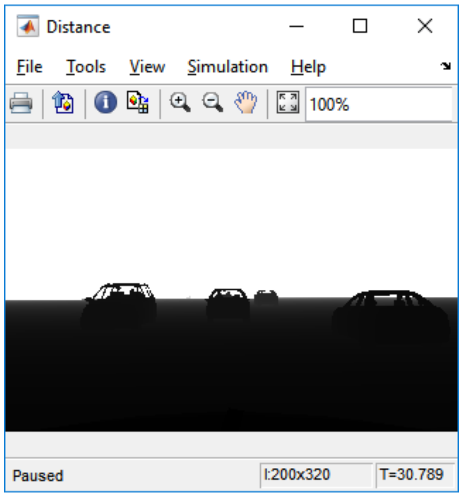


Figure 13. Original depth matrix scaling down to 256 levels.

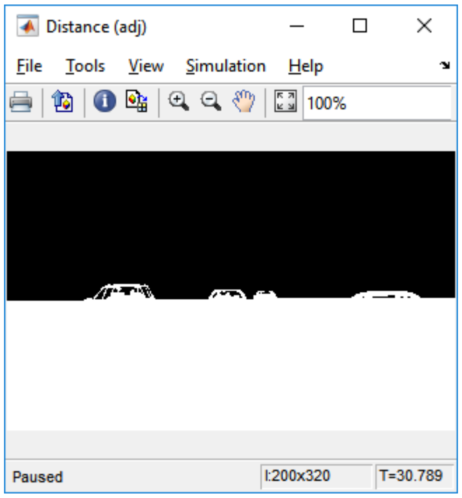


Figure 14. Negativelly flipped black/white image of the depth matrix.

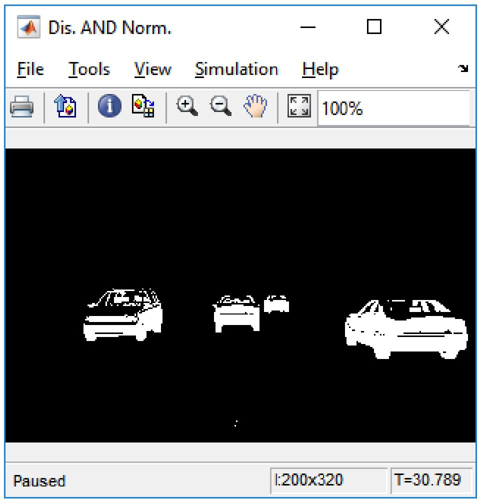


Figure 15. Detected traffic cars image.

4. The detected car areas are further simplified with an erosion operation through the **Opening** block to get an image shown in Figure 16, which is in turn passed to the **Blob Analysis** block (③ in Figure 7), available as part of the *Computer Vision System Toolbox* library. The Blob Analysis is based on a much simpler Boolean image (0 or 1 in each pixel), which shows only traffic vehicles in white (numerical value of 1) and other area is black (0).

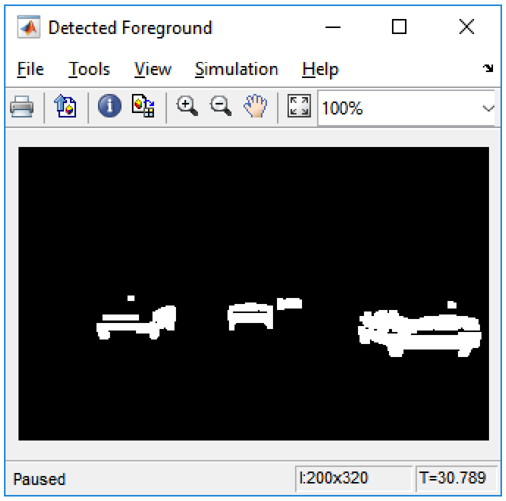


Figure 16. Boolean image highlighting traffic vehicles.

5. The detected rectangles information and video signals go in to the **Display Result** block ④. The inside of this block is shown in Figure 17.

6. Detected area information and depth image go in to **Distance detection** block ①, which detects a minimum longitudinal distance of each detected area whose lateral distance in the camera coordinate system is calculated by a simple geometric analysis. By knowing the longitudinal distance and lateral distance of each detected traffic vehicle, the block finds who is the front car and its headway distance is passed to the result image.

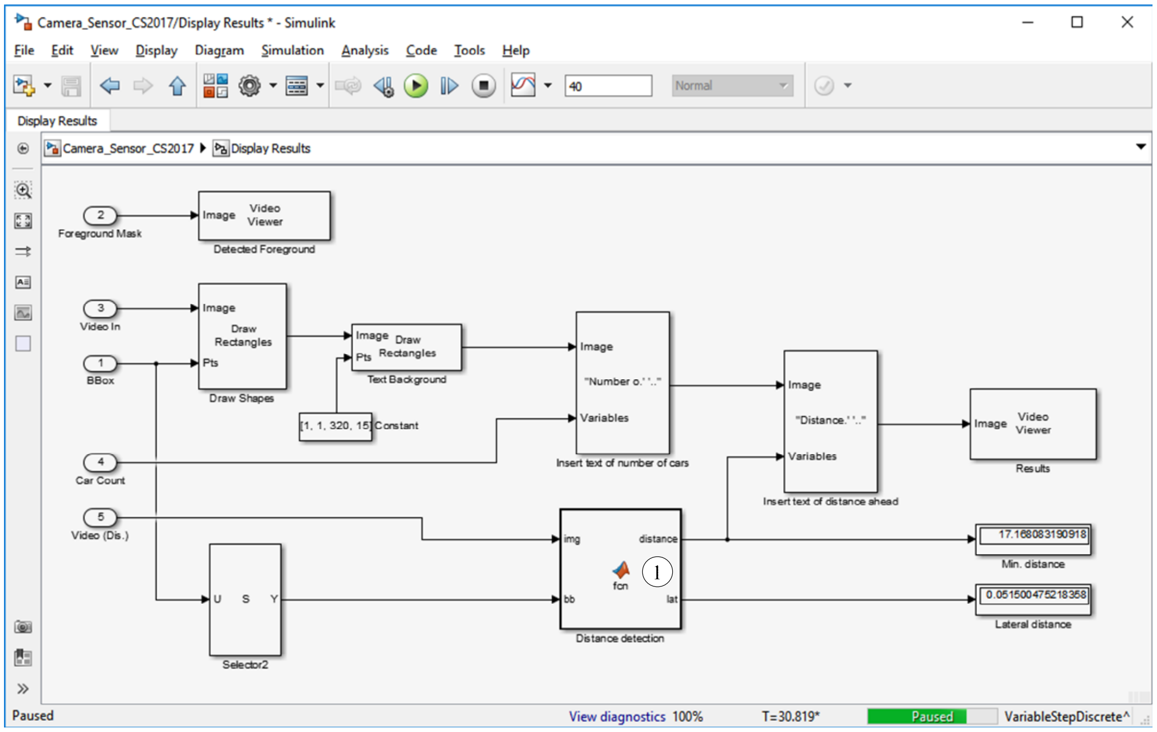


Figure 17. Display result block

The ego vehicle in this example is also equipped with the built-in sensor defined using the **ADAS Sensor for Range and Tracking** screen. The location of this sensor is as same as the camera sensor location, to allow comparison of the distance from each. Figure 18 shows that the distance calculated by the ADAS sensor is same as the headway distance derived through the image processing in Simulink.

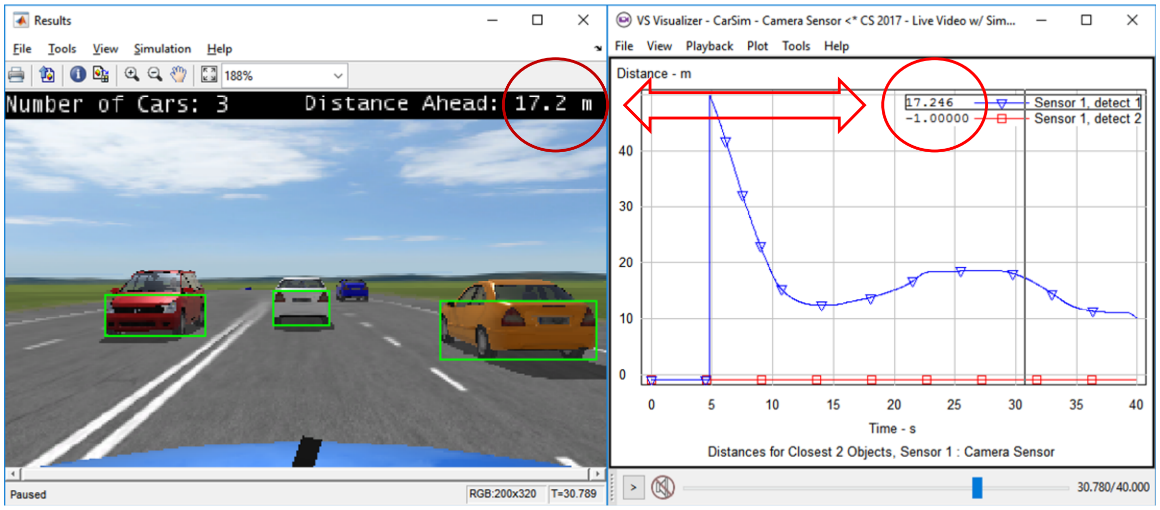

Results
File Tools View Simulation Help
188%
Number of Cars: 3
Distance Ahead: 17.2 m
Paused
RGB:200x320
T=30.789
VS Visualizer - CarSim - Camera Sensor <* CS 2017 - Live Video w/ Sim...
File View Playback Plot Tools Help
Distance - m
17.246 Sensor 1, detect 1
-1.00000 Sensor 1, detect 2
40
30
20
10
0
0 5 10 15 20 25 30 35 40
Time - s
Distances for Closest 2 Objects, Sensor 1 : Camera Sensor
30.780/40.000

Figure 18. Comparison between the distance calculation from the ADAS sensor and the camera sensor with image processing.

Highway Lane Detection Example (Simulink)

CarSim includes an example simulation run named **Highway Lane Detection (Simulink)** in the category **ADAS and Active Safety** of the Run Control screen. TruckSim includes a similar example.

Camera Sensor settings are the same as those described for the previous example (page 2). Running the Simulink model is also the same way as the one in the previous example.

In this case, the steering angle is selected as a CarSim import value. Figure 19 shows the Simulink model for the Highway Lane Detection example.

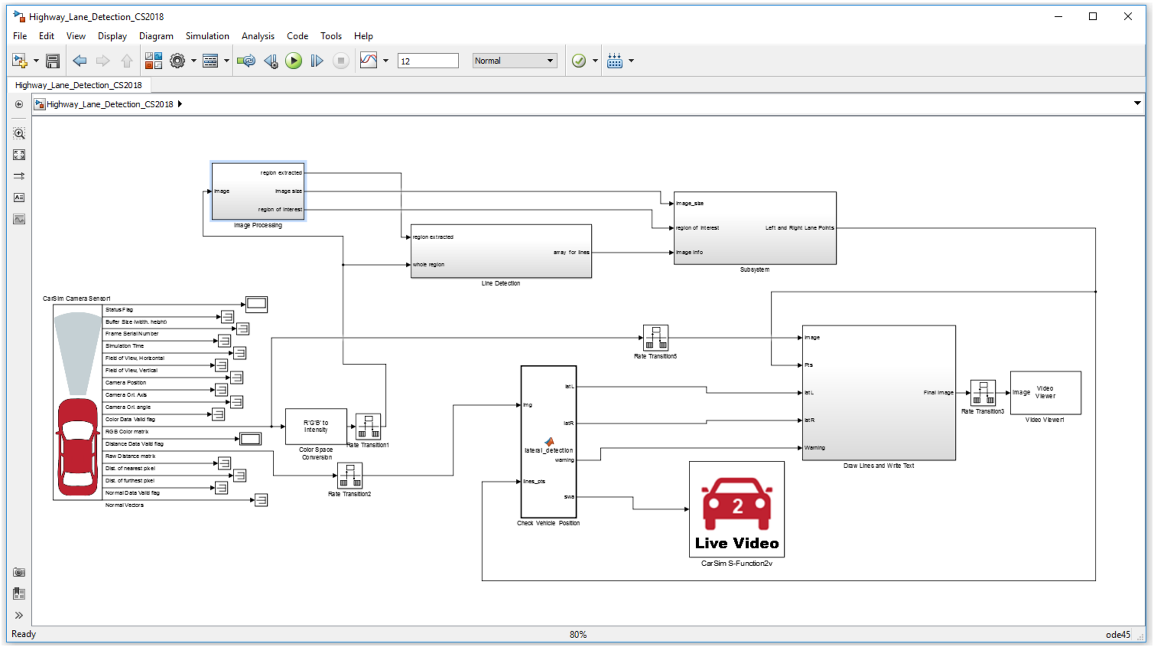
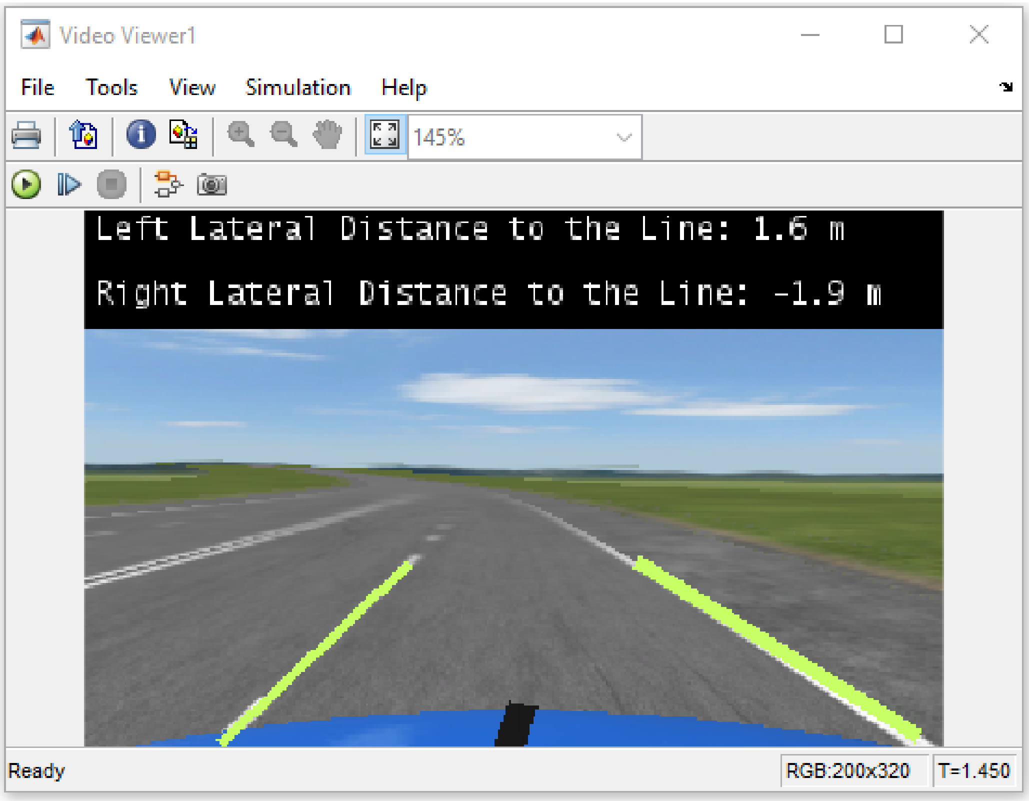


Figure 19. Highway Lane Detection Example Simulink model.

Click the **Start** button for the example Highway Lane Detection Example Simulink model. In addition to the Simulink model, a live animation window opens first, and the results window appears after that. The results window looks same as a live animation window before running the simulation.

After clicking the **Start** button from the Simulink model, the results window shows the 320x200 camera image which includes the lane detection information (Figure 20). The green lines are drawn where the lanes in both sides of the vehicle. On the top of the window, there are information about the lateral distances to each lane from the center of the screen. During the simulation, the vehicle veers and the message appears in the same window. When the vehicle drifts, the steering correction is applied.

In this example, RGB Color matrix and Raw Depth matrix were used from the VS camera sensor block. The following is the list of the steps taken in the analysis of the lane detection in the Simulink model.



```
Left Lateral Distance to the Line: 1.6 m

Right Lateral Distance to the Line: -1.9 m
```

Figure 20. Front camera image with highway lane detection indicated by green lines.

1. The RGB color image is converted to gray scale, and the edge detection method is applied. Isolate the region of interest (① in Figure 21). The region of interest is defined inside ① block. Depending on the road or the location of the camera sensor, the user can change the size of the area of interest.

2. The lanes are detected using Hough transformation ②.

3. Obtains the locations of those lanes ③. Here, once the lanes are detected in ②, it uses a simple geometry to get the upper and lower points on the right and left lanes around the region of the interest. In this example, it also retains those four points on the lanes for the three previous time steps, and if the lanes are not correctly detected in the current time step, it uses the points in the previous steps. This part of the calculation might not be necessary depending on the image of the lane.

4. Calculate the distance based on the field view information from CarSim ④. Using the depth matrix output and the geometry, the lateral distances to the both lanes from the center of the camera are calculated. In this example, the field of view of 55.0 deg. is used. This value needs to be matched with the one entered in CarSim. Inside ④ block, the field view is defined as `fov` variable. If the user changes the field of view value in CarSim, the `fov` variable needs to be updated. In this block, the steering wheel angle variable called `swa` is also hardcoded. This angle is used to turn the steering wheel opposite way when the vehicle drifts.

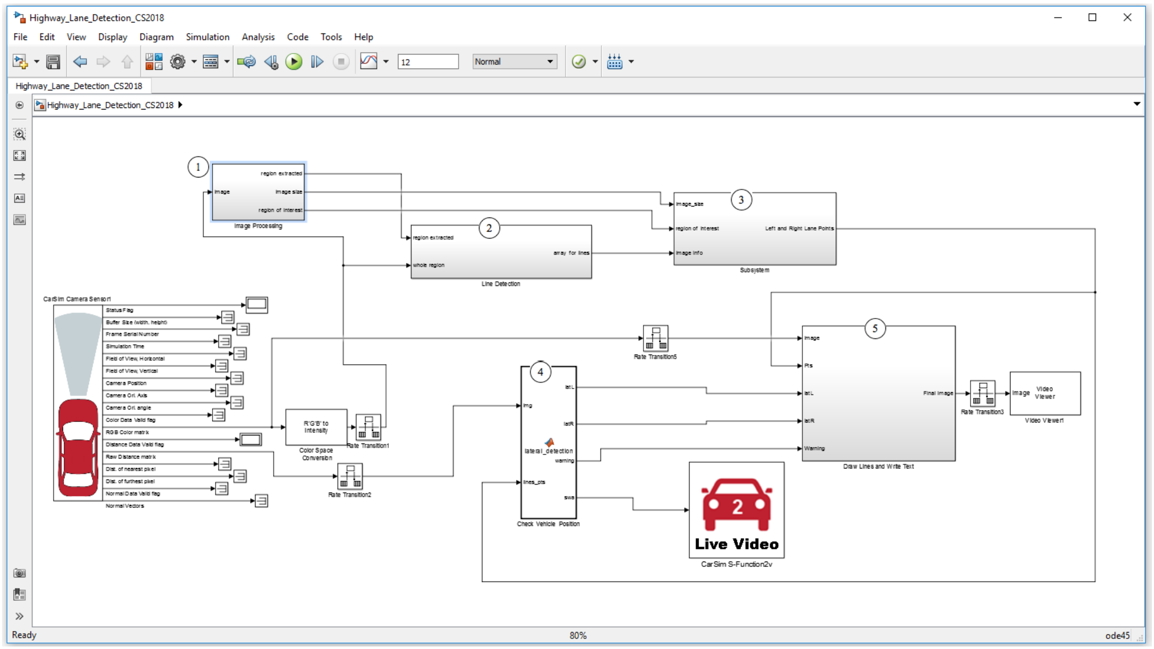


Figure 21. Highway Lane Detection Simulink Mode.

5. Green lines are drawn, and the lateral distance information text is written. Based on the lateral distance information, the steering angle is applied over a certain threshold ⑤.

Note that this is a simple example meant to merely show how to use a camera sensor using CarSim and the Simulink. This example only works with a certain camera angle and a certain type of the lane image. More detailed model is needed to be created if there are multiple lanes with multiple objects on the curvy road.

Highway Lane Detection Example (Python)

CarSim and TruckSim include an example named **Highway Lane Detection (Python)** in the Run Control category **ADAS and Active Safety**, available from the Run Control screen’s Datasets drop-down menu.

The primary goal is to show how to connect CarSim and TruckSim to Python libraries that also make use of NumPy and OpenCV in the context of Highway Lane Detection. The techniques demonstrated here could be used to extend this example to perform other image processing tasks.

Python 3.x is required to run this example; the following libraries are also required:

- NumPy (www.numpy.org)
- OpenCV (www.opencv.org)

Internally, this example was tested using:

- Python 3.7.2
- NumPy 1.18.1
- OpenCV 4.4.0

Camera Sensor and Simfile Location Settings

Camera Sensor settings are the same as those described in Vehicle Detection Example section (page 2). The default settings used in this example are:

- `FIELD_OF_VIEW = 55`
- `TEX_W = 320, TEX_H = 200`
- `BUFFER_NAME = EBS.`

The **Run Control** dataset has a link to a dataset from the **Models: Transfer to Local Windows Directory** called `simfile generator`(Figure 22).

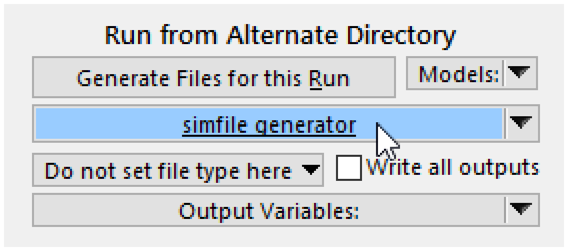
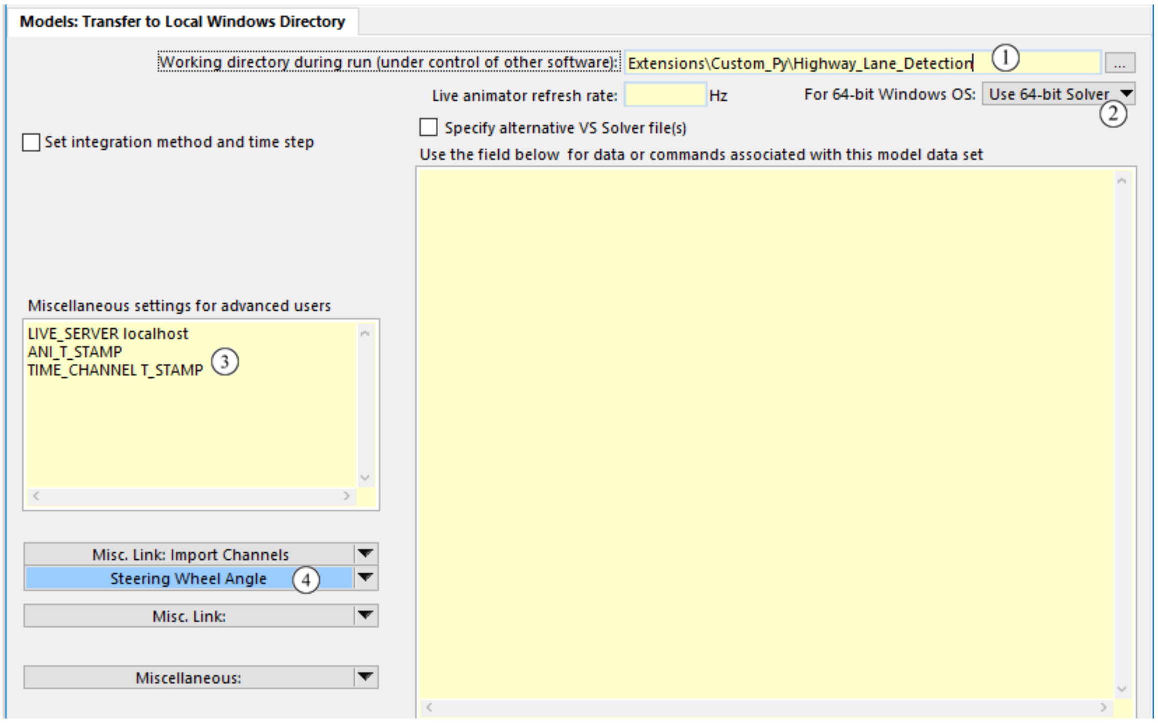

Run from Alternate Directory
Generate Files for this Run
Models:
simfile generator
Do not set file type here
Write all outputs
Output Variables:

Figure 22. Link to Models:Transfer to Local Windows Directory: simfile generator.

This dataset specifies a working directory during the simulation ① (Figure 23) and it should be set to `Extension\Custom_Py\Highway_Lane_Detection`. The solver is set to use 64-bit ② since Python 3.x is used. `ANI_T_STAMP` and `TIME_CHANNEL T_STAMP` commands are included for live animation ③. In this example, Steering Wheel Angle is imported.



Models: Transfer to Local Windows Directory

Working directory during run (under control of other software): Extensions\Custom_Py\Highway_Lane_Detection ①

Live animator refresh rate: Hz

For 64-bit Windows OS: Use 64-bit Solver ②

Set integration method and time step

Specify alternative VS Solver file(s)

Use the field below for data or commands associated with this model data set

Miscellaneous settings for advanced users

```
LIVE_SERVER localhost
ANI_T_STAMP
TIME_CHANNEL T_STAMP ③
```

Misc. Link: Import Channels

Steering Wheel Angle ④

Misc. Link:

Miscellaneous:

Figure 23. Simfile Generator Dataset.

How to Run the Example

1. Be sure NumPy and OpenCV are installed. See the **Appendix** for more details.

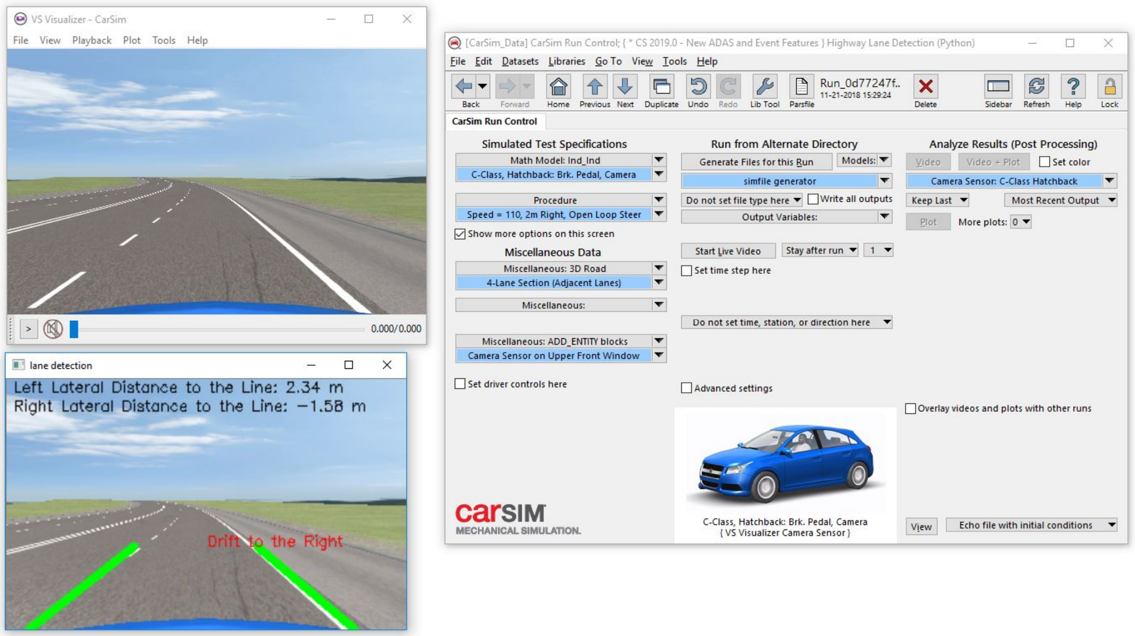
2. From the **Run Control** screen, click the button **Generate Files for this Run**. This will create a file named `simfile.sim` and save it in the database folder: `Extensions\Custom_Py\Highway_Lane_Detection`. The `Extensions` folder can be found at the root level of your CarSim / TruckSim database folder.

3. From the **Run Control** screen, open the VsV Live Animation window by clicking the **Start Live Video** button. Make sure to set **Number of live animators for this run** to 1.

4. Browse to the `Extensions\Custom_Py\Highway_Lane_Detection` directory and run `LaneDetection.py`. This can be accomplished using a Python IDLE environment or via the command line. For example, open the Command Prompt, browse to the directory listed above and type: `python LaneDetection.py`. For example:

```
D:\CarSim2021_Data\Extensions\Custom_Py\Highway_Lane_Detect
ion>python Lanedetection.py
```

When the `LaneDetection.py` script starts, it brings up a “lane detection” window (the bottom left image in Figure 24) containing the green lines, indicating the locations of the lanes and the lateral distance information. When the vehicle drifts more than 0.7 m, a warning is displayed (“Drift to the Right” or “Drift to the Left”) and the vehicle responds by adding or subtracting 25 deg. steering wheel from the base steering wheel angle of 5 deg. The steering wheel angle increment and the 0.7 m distance threshold are hardcoded in `LaneDetection.py`. End-users are encouraged to modify these thresholds to experiment with different control responses.


VS Visualizer - CarSim
File View Playback Plot Tools Help
0.000/0.000
lane detection
Drift to the Right
[CarSim_Data] CarSim Run Control; { * CS 2019.0 - New ADAS and Event Features } Highway Lane Detection (Python)
File Edit Datasets Libraries Go To View Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile Delete Sidebar Refresh Help Lock
Run_0d77247f..
11-21-2018 15:29:24
CarSim Run Control
Simulated Test Specifications
Run from Alternate Directory
Analyze Results (Post Processing)
Generate Files for this Run
Video
Video + Plot
Plot
Start Live Video
View
carSIM MECHANICAL SIMULATION

```
Left Lateral Distance to the Line: 2.34 m
Right Lateral Distance to the Line: -1.58 m
```

Math Model: Ind_Ind
C-Class, Hatchback: Brk. Pedal, Camera
Procedure
Speed = 110, 2m Right, Open Loop Steer
☑ Show more options on this screen
Miscellaneous Data
Miscellaneous: 3D Road
4-Lane Section (Adjacent Lanes)
Miscellaneous:
Miscellaneous: ADD_ENTITY blocks
Camera Sensor on Upper Front Window
☐ Set driver controls here
Models:
simfile generator
Do not set file type here
☐ Write all outputs
Output Variables:
Stay after run
1
☐ Set time step here
Do not set time, station, or direction here
☐ Advanced settings
C-Class, Hatchback: Brk. Pedal, Camera
{ VS Visualizer Camera Sensor }
☐ Set color
Camera Sensor: C-Class Hatchback
Keep Last
Most Recent Output
More plots: 0
☐ Overlay videos and plots with other runs
Echo file with initial conditions

Figure 24. Running LaneDetection.py.

Explanations of Python Scripts

There are two DLL files and three Python files in `Extensions\Custom_Py\Highway_Lane_Detection:`

- `vs_sb_64.dll`. Dynamic Link Library for Shared Buffer. Used to grab shared buffer information from the graphics card.
- `vs_lv_ds_x64.dll`. Dynamic Link Library for VS Solver including Live Animation.
- `GetSharedBufferInfo.py`. Uses `vs_sb_64.dll` to read the shared buffer information.
- `Simulation_with_LiveAnimation.py`. Includes the call to several VS Solver API functions.
- `LaneDetection.py`. Calls both `GetSharedBufferInfo.py` and `Simulation_with_LiveAnimation.py`, gets necessary shared buffer information for image analysis, detects the highway lanes, and runs the solver with live animation.

`LaneDetection.py` is the primary file for this example and executes the following tasks:

1. Runs the CarSim / TruckSim VS Solver and VsV live animation.
2. Gets the color and the depth information for each pixel from the shared buffer.
3. Converts the image to gray-scale and conducts edge detection analysis.
4. Limits the region of the interest and obtains the upper and lower points of the left and right lanes.
5. Calculates lateral distances to the left and the right lanes. If the distance is more than the threshold, a predetermined steering wheel angle of applied in the other direction to bring the vehicle back to the center of the lane.
6. Terminates the simulation.

Customizing the Lane Detection Python Scripts

As mentioned earlier, the purpose of this example is to show how to connect the Python script to CarSim / TruckSim and run an image analysis by accessing shared buffer information from the graphics card, also displayed via the VsV Live Animator data stream. The example contains a simplified image analysis and the scripts are used with a simple road animation. However, end-users can use their own Python scripts.

There are several things users need to be aware of when using their own Python scripts:

- The default shared buffer name is “EBS” which can be set in CarSim (see: **Camera Sensor on Upper Front Window** dataset). This name, EBS, is hardcoded inside `GetSharedBufferInfo.py` (`bufferName = ‘EBS’`). If the shared buffer name is modified, it is necessary to update `GetSharedBufferInfo.py`
- The field of view (FOV), the steering wheel angle (SWA), and the distance threshold (`if abs(abs(latL)-abs(latR)) > 0.7`) are all hardcoded. Users need to edit them in `LaneDetection.py` if different thresholds are desired.

- The two DLLs contained in the `Highway_Lane_Detection` folder need to be in the same folder as the Python scripts.
- When testing a custom script and an error is encountered before the simulation is over, the shared buffer information will be locked and cannot be accessed immediately. If that happens, the shared buffer information will be locked for 180 seconds. To resolve this issue, users can:
  - Open the License Settings window from CarSim / TruckSim (**Tools** > **License Settings**).
  - Uncheck the license feature `Extra Live Animator`.
  - Click **Select** to click accept the changes and close the License Settings window
  - Reopen the License Settings window.
  - Reselect the Extra Live Animator license feature.
  - Click **Select** to accept the changes.

Pedestrian Detection Example

CarSim, TruckSim, and BikeSim include a **Pedestrian Detection (Simulink)** example found under the **Camera Sensors** or **ADAS and Active Safety** category of Run Control screens.

Before using this example, please read the Vehicle Detection Example (pg. 2 of this memo), paying special attention to sections ‘Camera Sensor Settings in the VS Datasets’ and ‘Running the Simulink Model.’

Just like two other camera sensor examples in this documentation, this example requires the Computer Vision System Toolbox. The example uses a built-in Simulink function called `peopleDetectorACF` which returns a pretrained people detector. For more information about the training data or the detection function, please refer to Mathworks website. Since `peopleDetectorACF` function is a new function, use Matlab 2017a or newer to run this example.

To run this example, make sure the Live Video selected is ‘1’ from the drop-down menu on the Run Control screen, and press the ‘Send to Simulink’ button. When the Simulink model is open, it should also open the video viewer window. If the video viewer window is not open, click the Video Viewer block in the Simulink model to open it. Figure 25 shows the Simulink model for the Pedestrian Detection example. The distance to a pedestrian, the brake pressure, and a variable indicating when a pedestrian is detected are selected as CarSim import values and the speed is used as an export value. The pedestrian-detection variable is used to provide a HUD alert in VS Visualizer.

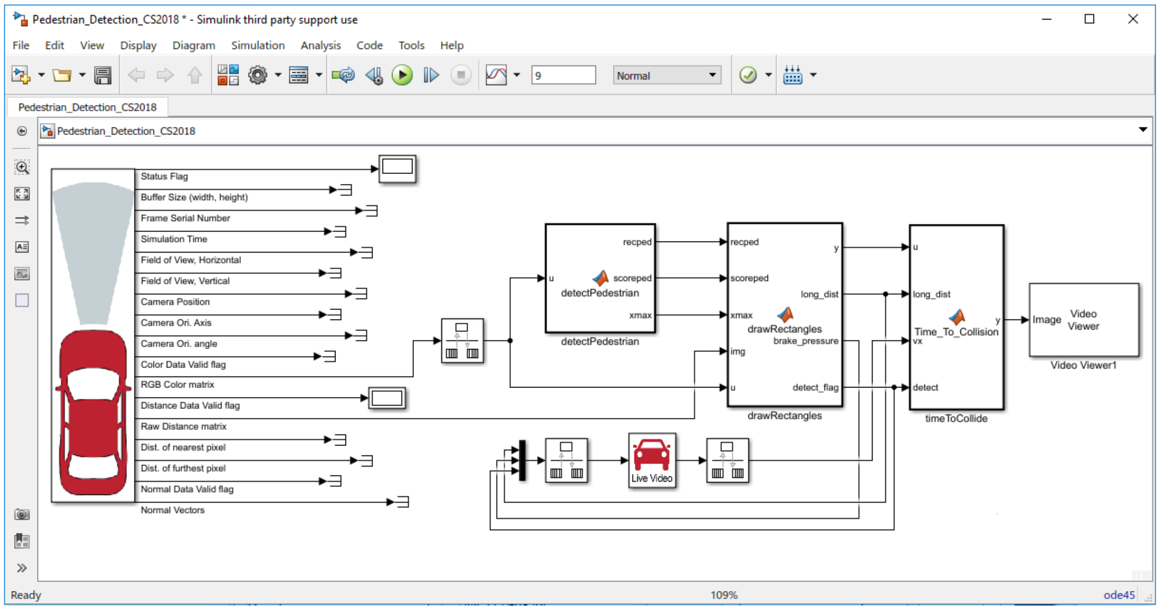


Figure 25. Pedestrian Detection Simulink Model.

After clicking the **Start** button from the Simulink model, the video viewer window shows the 320x200 camera image which includes a pedestrian walking from the right side of the screen. When a camera sensor detects the pedestrian, the distance to the pedestrian and time to collision information are displayed (Figure 26).

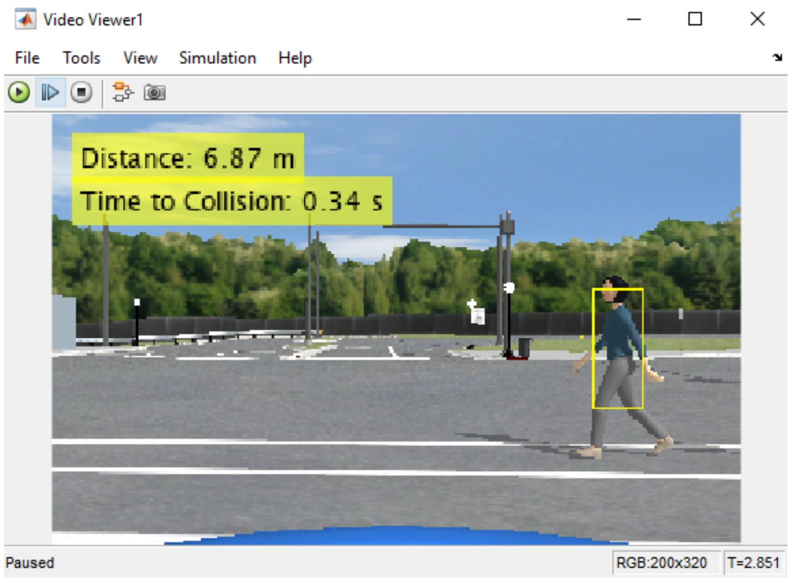


Figure 26. Front Camera Image with a Pedestrian.

The distance is calculated using the Raw Depth matrix from the VS camera sensor block, and when the distance to the vehicle becomes less than a certain length, the brake is applied.

The following is the list of the steps taken in the analysis of the pedestrian detection in the Simulink model.

1. The RGB color image is read, and a pedestrian is detected using the `peopleDetectorACF` function which returns a pretrained upright people detection (① in Figure 27). The user can adjust the size of the region of interest in block ①.
2. Once a pedestrian is detected, block ② draws a rectangle on the pedestrian. Using the Raw Depth matrix from the Camera Sensor S-Function, it calculates the distance from the vehicle to the pedestrian. The vehicle axle track (1.675m) and the field of vision (55.0 deg.) were obtained from CarSim and hardcoded in *.m file. The user can adjust these values if using a different vehicle. The brake pressure and the distance to apply the braking are also hardcoded and can be edited. Block ② assumes to detect only one pedestrian. The user may modify this if they want to detect more than one pedestrian.
3. Block ③ calculates the time to collision using the distance to a pedestrian and the vehicle velocity.

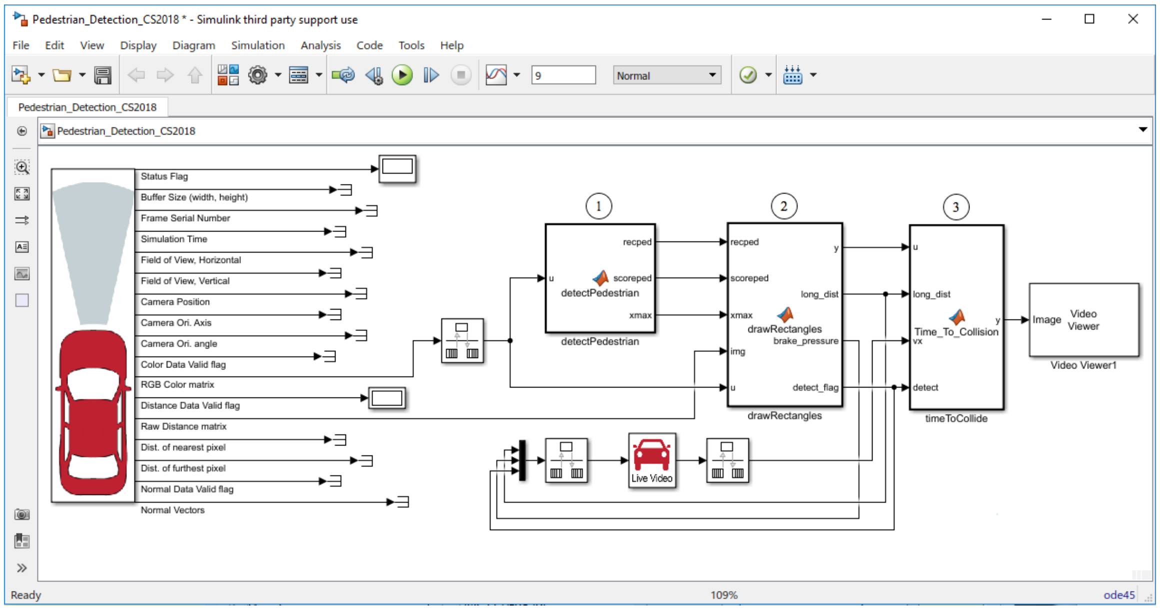


Figure 27. Pedestrian Detection Simulink Model.

As with the highway lane detection example, the purpose of this example is to show a simple usage of a camera sensor with CarSim/TruckSim/BikeSim and Simulink. We expect our users to edit the Simulink model to use for their own analysis.

Reference

1. Kageyama, I. and Watanabe, Y., “Basic Study on an Image Processing Method for Autonomous Vehicle”, Transactions of the Japan Society of Mechanical Engineers Series C, Vol. 62, 1996, No. 593, pp83-88, http://doi.org/10.1299/kikaic.62.83.

Appendix

This section includes installation information for NumPy and OpenCV in support of the Python-based Highway Lane Detection example. Since a multitude of online resources are available in regard to these software, this section is intended to serve as a general guideline rather than an authoritative set of instructions. Both NumPy and OpenCV are installed via the Command Prompt; from our experience, NumPy should be installed before OpenCV.

NumPy

Download the version of NumPy (.zip and .whl files) that matches your Python and Windows OS installations. As of the CarSim / TruckSim 2021.0 release, the examples were tested using the file: `numpy-1.18.1-cp37-cp37m-win32.whl.`

Use the Command Prompt to browse to your root Python folder and execute the installation using the `pip` command:

```
C:\Program Files\Python37>pip install numpy==1.18.1
```

Optional Content

In setting up the working environment for these examples (and before installing OpenCV), we found it useful to also install the following libraries:

```
C:\Program Files\Python37>pip install scipy

C:\Program Files\Python37>pip install matplotlib

C:\Program Files\Python37>pip install scikit-learn

C:\Program Files\Python37>pip install scikit-image
```

OpenCV

Due to its open-source nature, there are many places on-line where the OpenCV `.whl` files may be found. The most important thing to look for is the `.whl` file that matches your Python installation and Windows OS. As of the CarSim / TruckSim 2021.0 release, the examples were tested using the file `opencv_python-4.4.0-cp37-cp37m-win_amd64.whl.`

After downloading the `.whl` file, use the Command Prompt to browse to the folder location containing it and execute the following `pip` command:

```
pip install .\opencv_python-4.4.0-cp37-cp37m-win_amd64.whl
```

In other words, `C:\My_Files\pip install .\opencv_python`...

Be careful with spaces: in the line, above, there is a space after "install" and before ".\". If no space is used, the following message will appear:

```
ERROR: unknown command "install.\opencv_python-4.4.0-cp37-cp37m-win_amd64.whl"
```