KPIT

End-to-End Closed-Loop Validation of Automated Driving (AD) Systems

MathWorks Automotive Conference 2022

Deepika , Chinmayi, Srinivas, Bhagyashree

Contents

Need of the Autonomous Driving Industry

Speed

With the increasing need for Autonomous vehicle, advanced AD solutions needs to be brought quick to market.

Millions of scenarios

All AD solutions needs to be tested with millions of scenarios for making it safe and road ready.

Architecture

AD software is moving towards SDV/SOA, which demands for speedy validation process.

Mandatory

To expedite this process integrated SIL testing is mandatory.

Typical Challenges

Field Testing is Expensive & Inefficient

AD control solutions needs to be validated in closed loop & Sensor solutions in open loop. Simulation tools do not give end-to-end validation solution thus, most of the integrated AD solutions are validated in field testing which is expensive, not energy efficient & time consuming.

AD System Availability

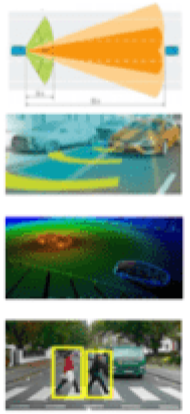
End-to-end validation is not possible till all the AD systems are available for validation

Need for SW Tools

To create a complete closed loop validation framework, different software and tools needs to work in conjunction.

Sensors & Sensor Configuration

Sensors and sensor configuration needs to be finalized before validation of software.



Compatibility Challenge

Compatibility of different tools, integration of software in different environment and execution in the integrated setup is a strenuous job.

High Volume of Data

To validate with real field data, data logging and replay are essential. This comes with high volumes of information capturing, replaying sensor data from multiple systems in parallel, synchronizing timestamps and simulating.

KPIT solution
End–to–end validation framework

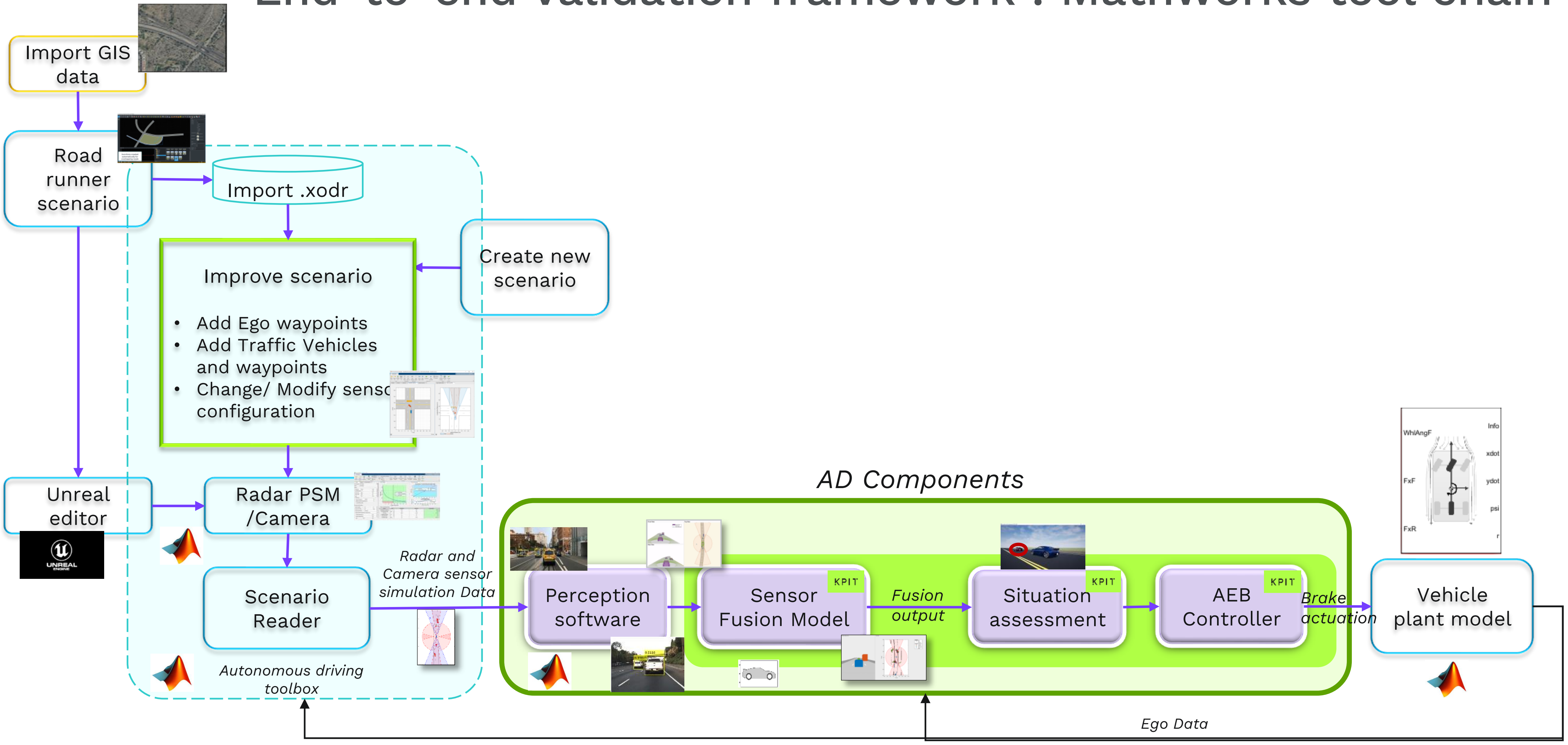
Using MathWorks tools mentioned below a complete closed loop validation framework has been created.

- *Autonomous Driving Toolbox,*
- *Radar toolbox,*
- *Simulation 3D Scene,*
- *Image processing toolbox*
- *Vehicle dynamics boxset*
- *Unreal engine and Road runner*

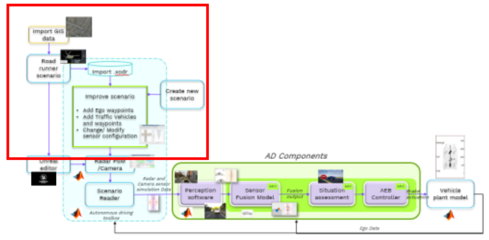
An end-to-end closed-loop framework in a single IDE helps to give;

✓ higher-quality products
✓ reduces validation cost,
✓ integration challenges
✓ delivers innovations faster

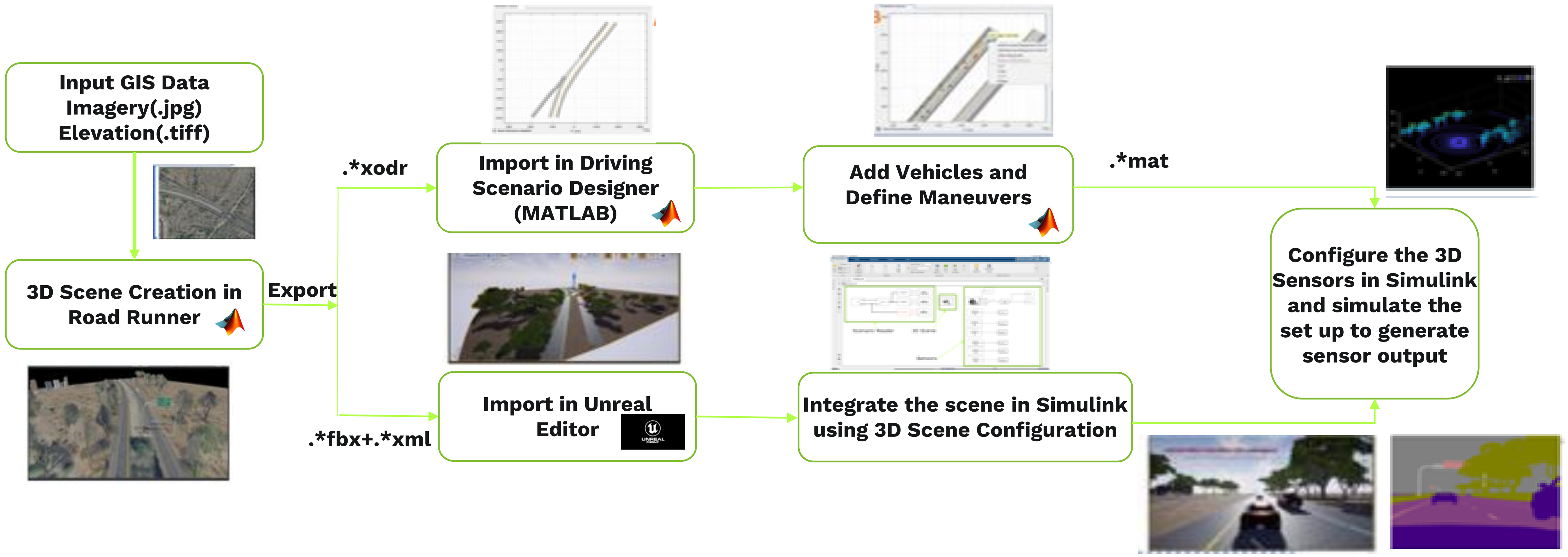
End-to-end validation framework : Mathworks tool chain



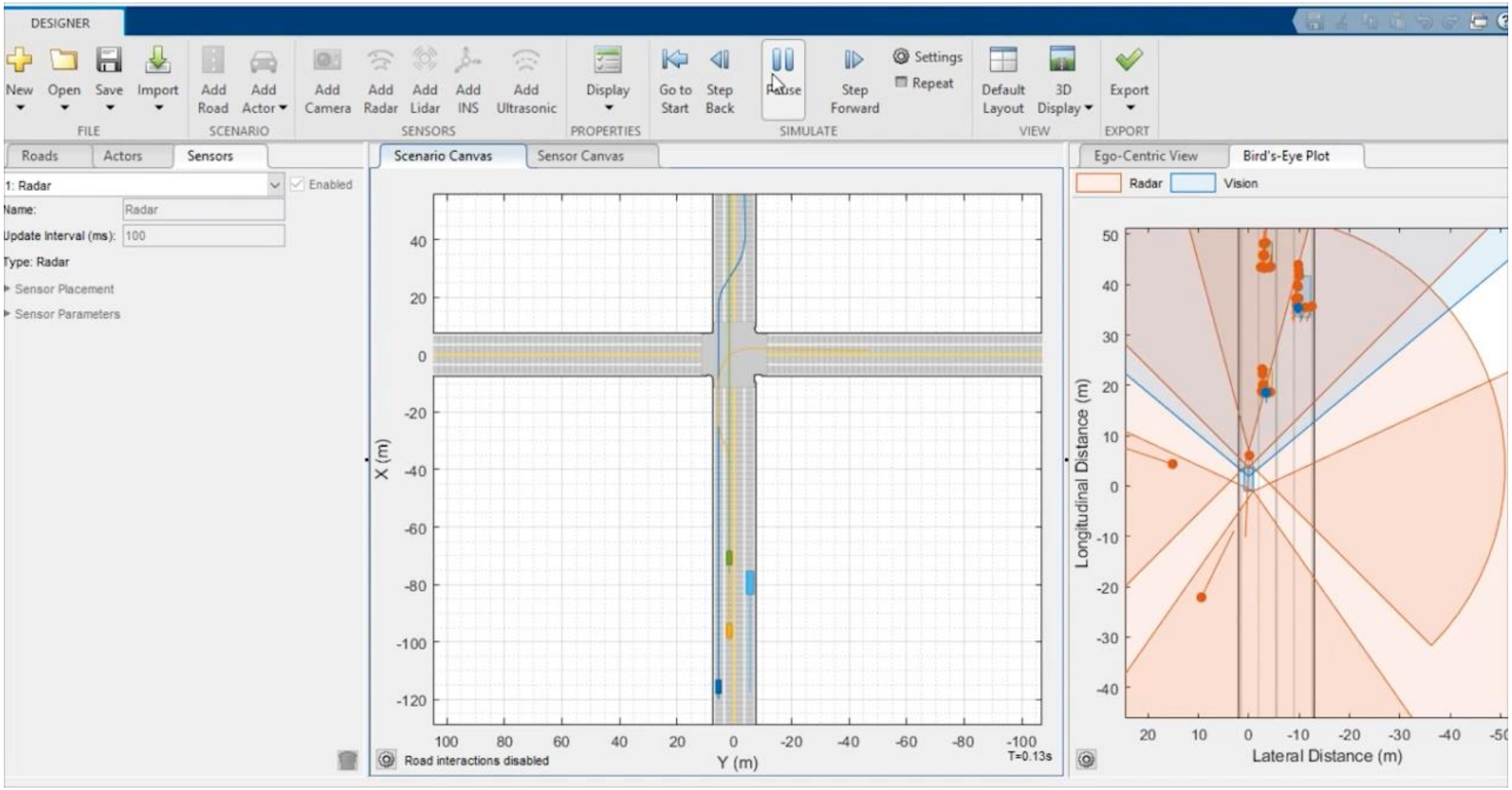
RoadRunner and Unreal Editor



- Scenario created in RoadRunner can be exported to Unreal Editor and to the driving scenario designer
- *.xodr : driving scenario designer
- *.fbx+xml : Unreal Engine


Input GIS Data
Imagery(.jpg)
Elevation(.tiff)
3D Scene Creation in Road Runner
Export
.*xodr
.*fbx+.*xml
Import in Driving Scenario Designer (MATLAB)
Import in Unreal Editor
Add Vehicles and Define Maneuvers
Integrate the scene in Simulink using 3D Scene Configuration
.*mat
Configure the 3D Sensors in Simulink and simulate the set up to generate sensor output

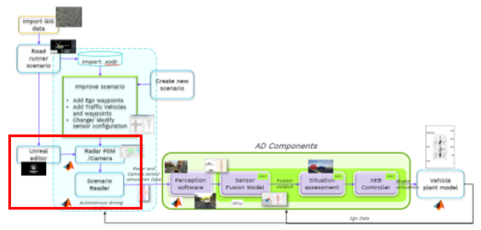
Closed Loop Demo – Input Scenario in DSD



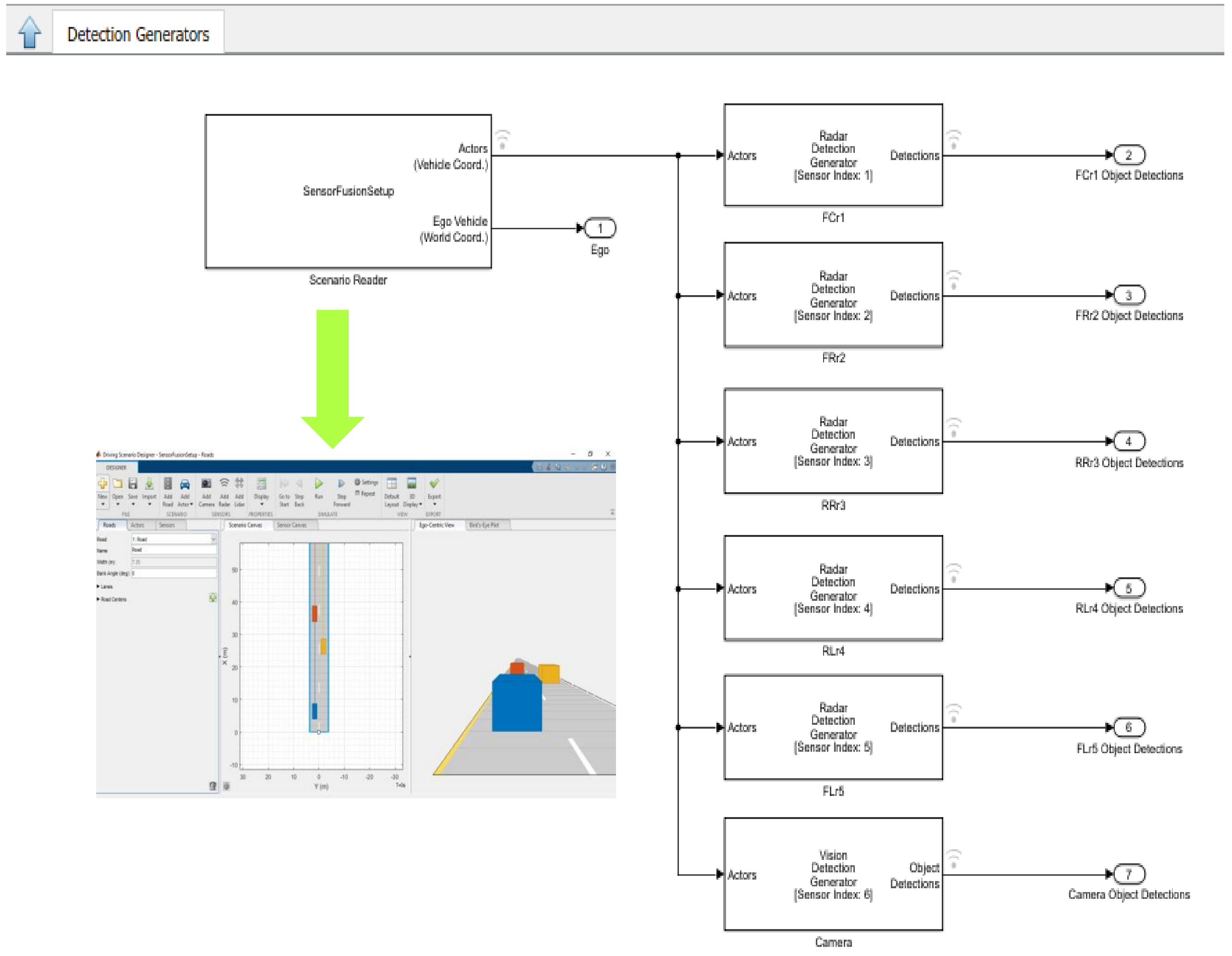
1: Radar
Enabled
Name: Radar
Update Interval (ms): 100
Type: Radar
Sensor Placement
Sensor Parameters

KPIT

Radar and Camera

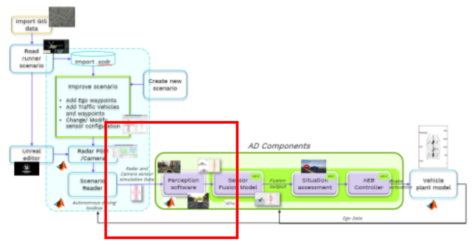


- For the case study, we have considered 5 radars and 1 camera configuration.
- The scenario data extracted from the Driving scenario designer is fed to the Radar detection generator and the vision detection generator blocks in the Simulink
- The outputs from these blocks are fed the perception algorithm and the sensor fusion model to obtain the fused output.



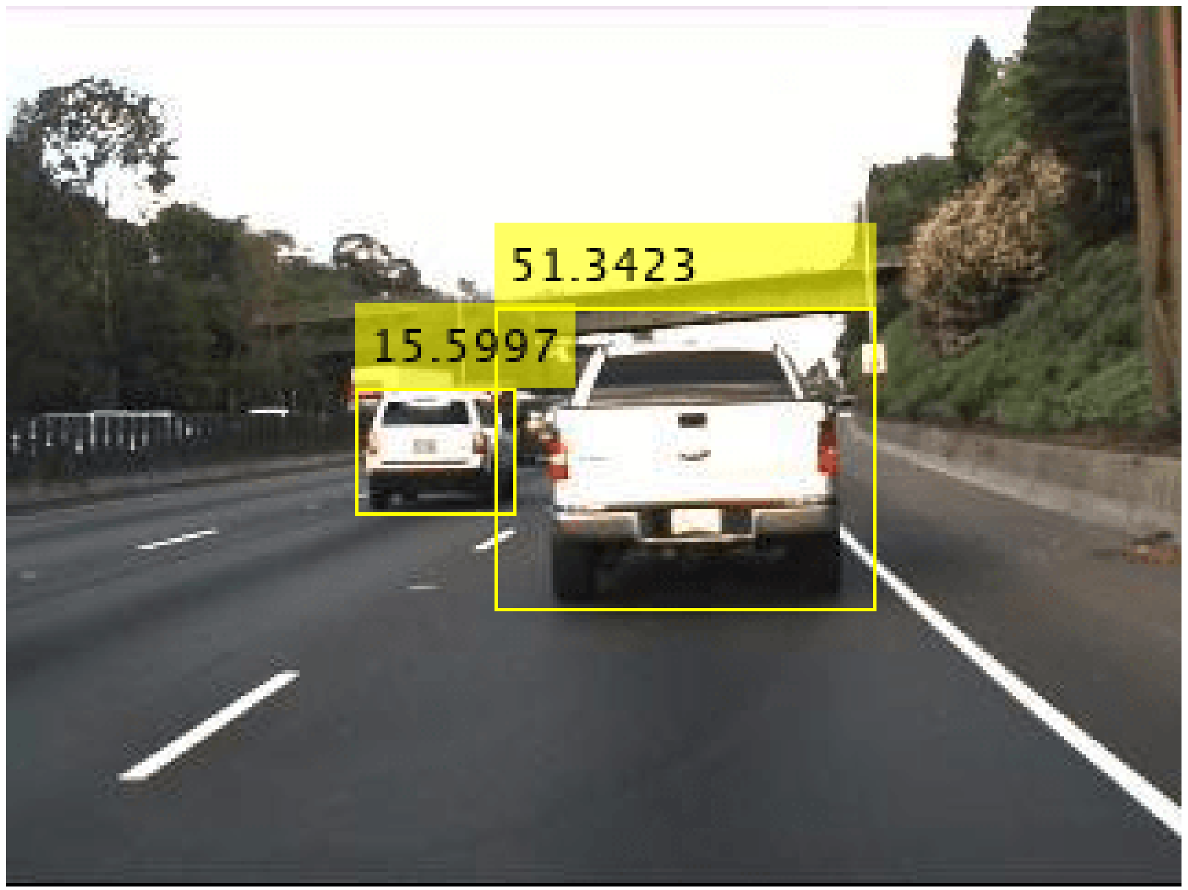
Detection Generators
SensorFusionSetup
Actors (Vehicle Coord.)
Ego Vehicle (World Coord.)
Scenario Reader
1 Ego
Actors
Radar Detection Generator [Sensor Index: 1]
Detections
FCr1
2 FCr1 Object Detections
Radar Detection Generator [Sensor Index: 2]
FRr2
3 FRr2 Object Detections
Radar Detection Generator [Sensor Index: 3]
RRr3
4 RRr3 Object Detections
Radar Detection Generator [Sensor Index: 4]
RLr4
5 RLr4 Object Detections
Radar Detection Generator [Sensor Index: 5]
FLr5
6 FLr5 Object Detections
Vision Detection Generator [Sensor Index: 6]
Object Detections
Camera
7 Camera Object Detections

Perception Software

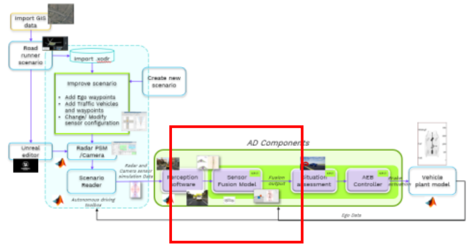


- For evaluation purpose, ACF (aggregated channel features) vehicle detector algorithm from MathWorks is used in ADT.
- The detector analyzes images of roads captured using a monocular camera sensor and returns information about the positions of different vehicles that are in the visible range of the camera.

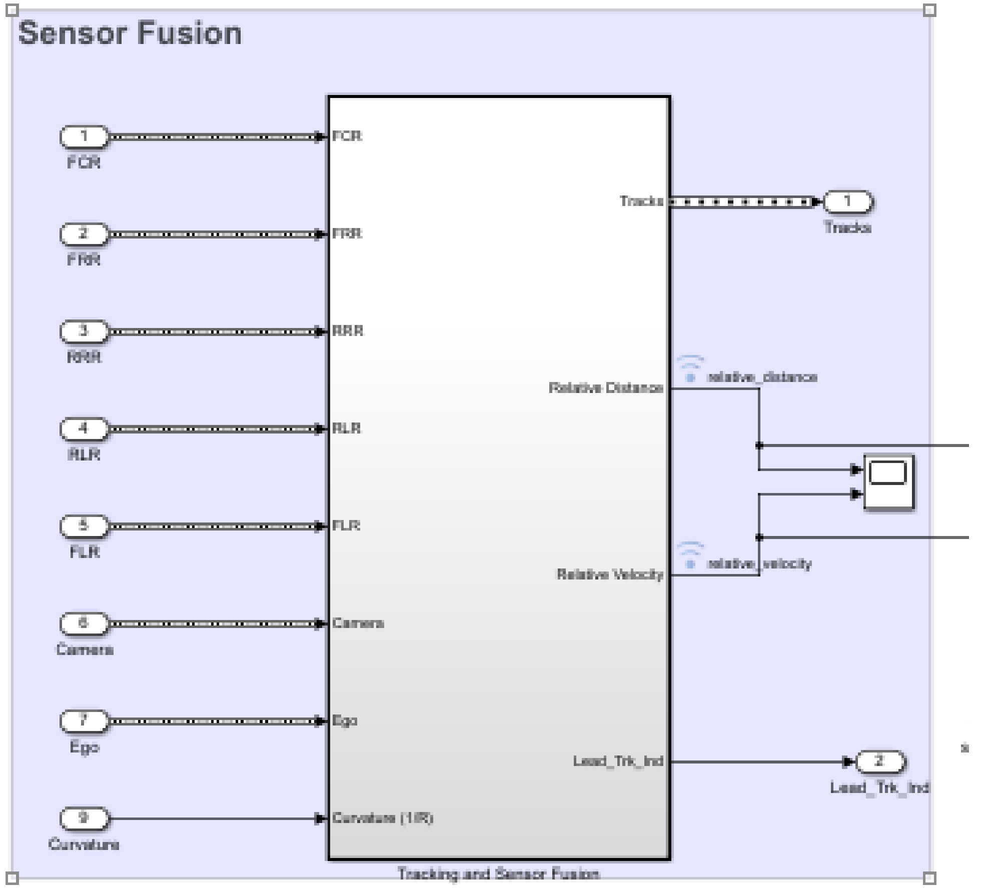
Detected Vehicles and Detection Scores

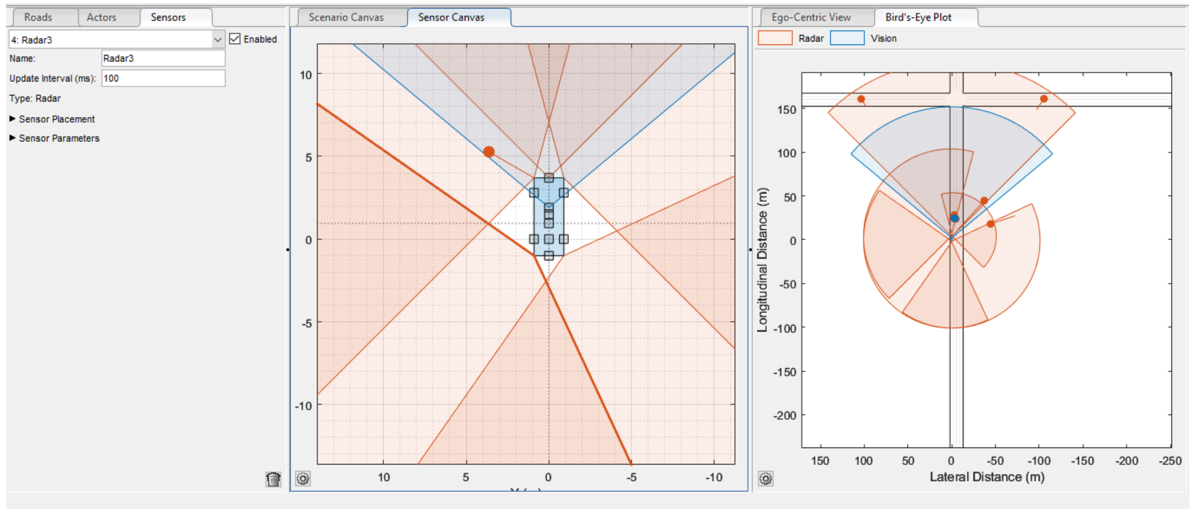


Sensor fusion

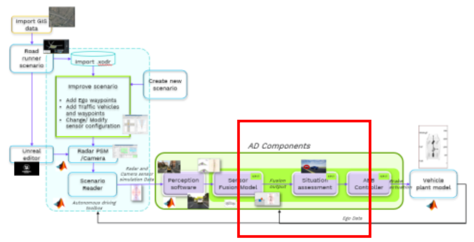


- The sensors data from ADT is fed to the sensor fusion module (KPIT assert)
- The sensor fusion performs association of camera and radar data and provides tracking of the objects.
- These objects are used for Situation assessment of the AD/ADAS system.



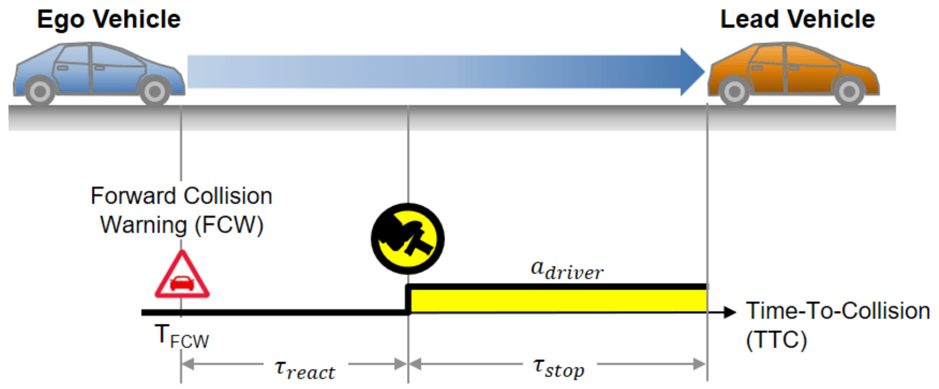


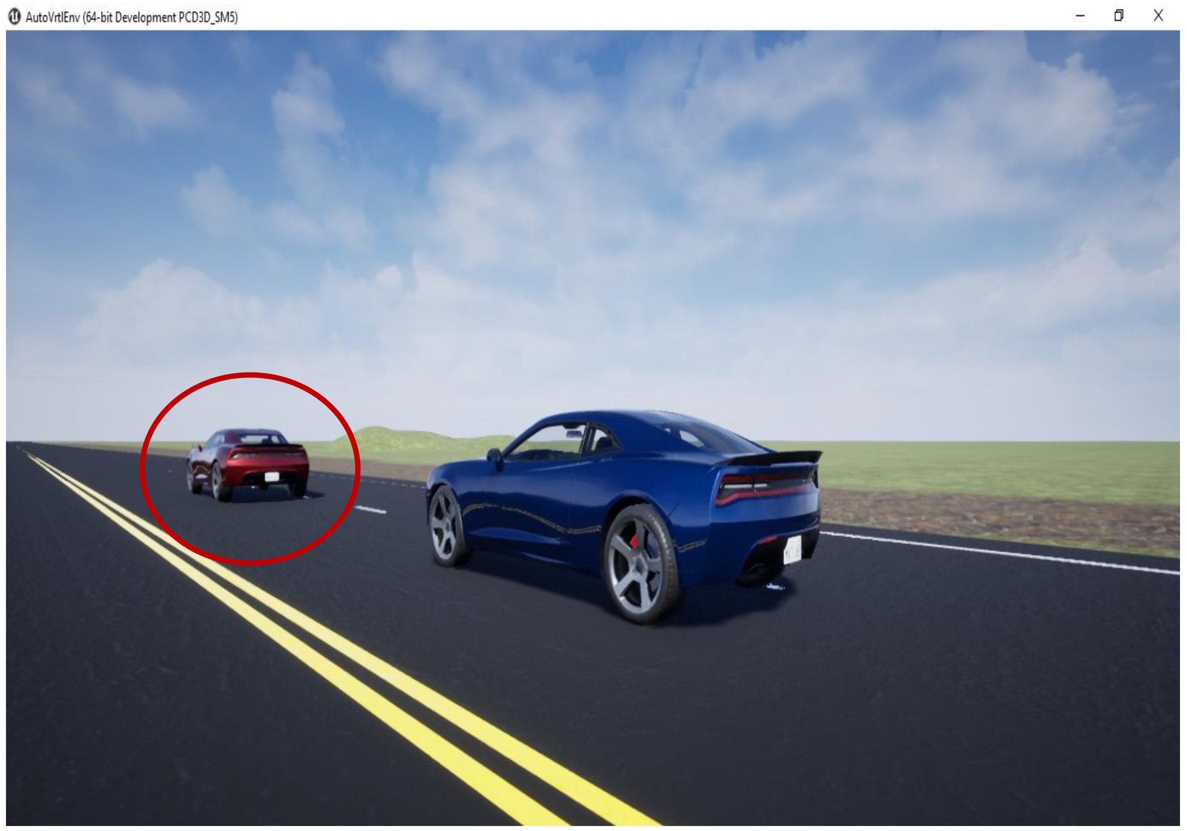
4: Radar3 ☑ Enabled
Name: Radar3
Update Interval (ms): 100
Type: Radar



Situation Assessment

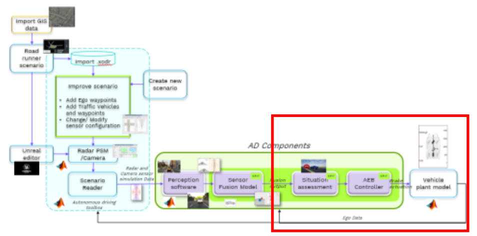
- The Situation Assessment module uses the sensor fusion track information, to assess the scene.
- In this case, depending on the track object information and ego vehicle information from vehicle dynamics block, Safety critical object is identified and TTC is computed.

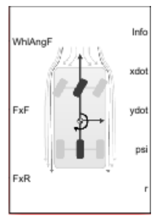




AEB Feature

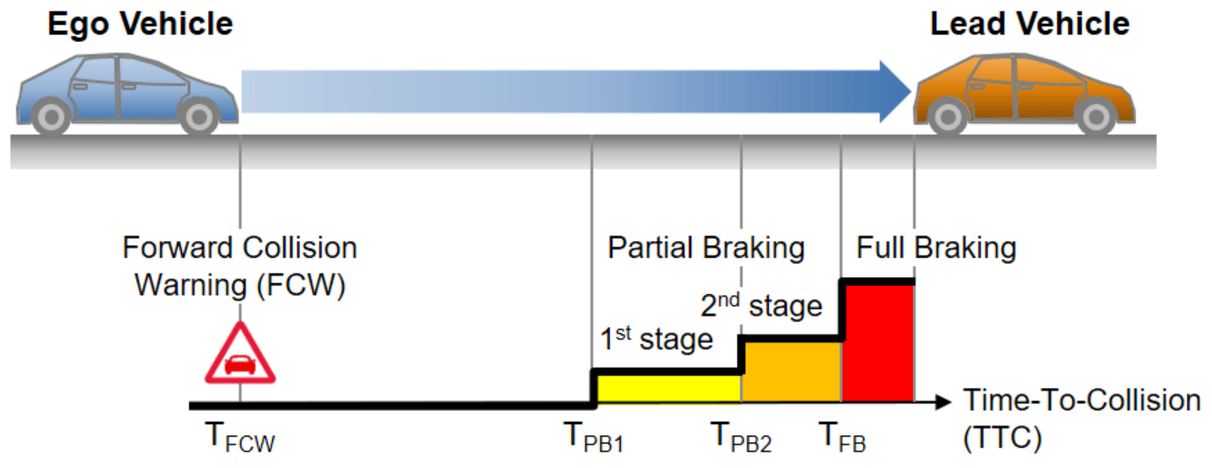
- In this case study, AEB feature (KPIT assert) was integrated in the closed loop.
- The selected object which is within the threshold of AEB TTC limit triggers AEB feature.
- The AEB feature controls the vehicle to come to a stop by generating brake actuation command.





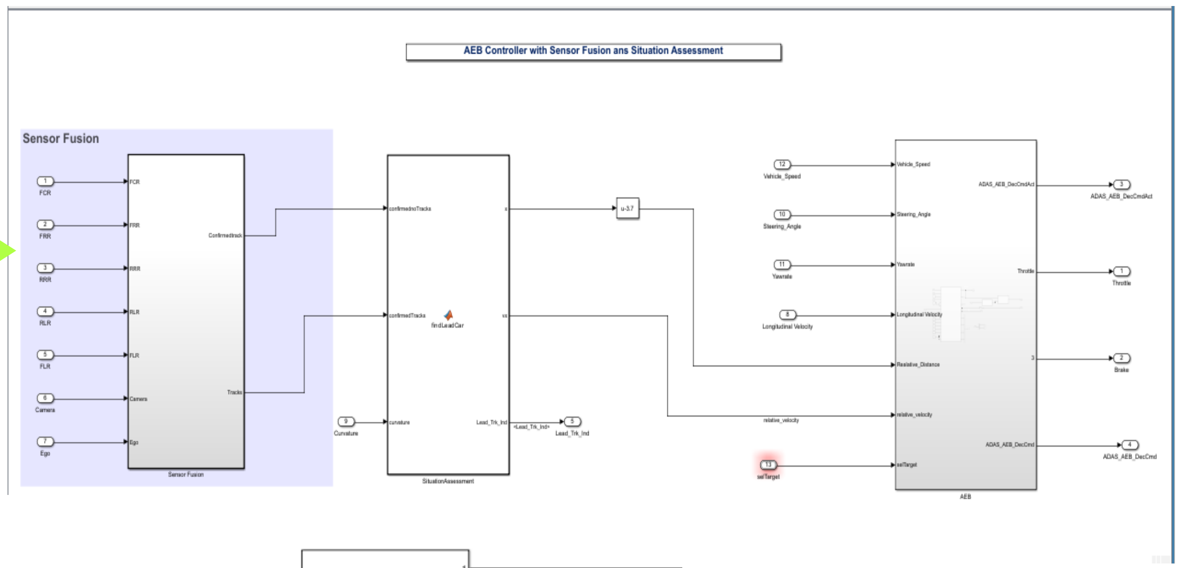
Vehicle dynamics model

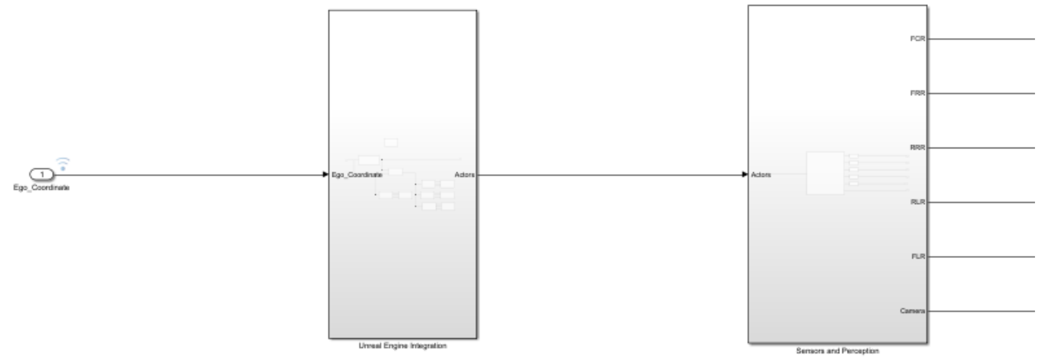
- The brake control command actuates the vehicle dynamics model.
- The ego vehicle states from plant model is looped back to the unreal editor to complete the closed loop framework
- For other AD features, control commands like acceleration/ engine torque or yaw is interfaced to vehicle dynamics model



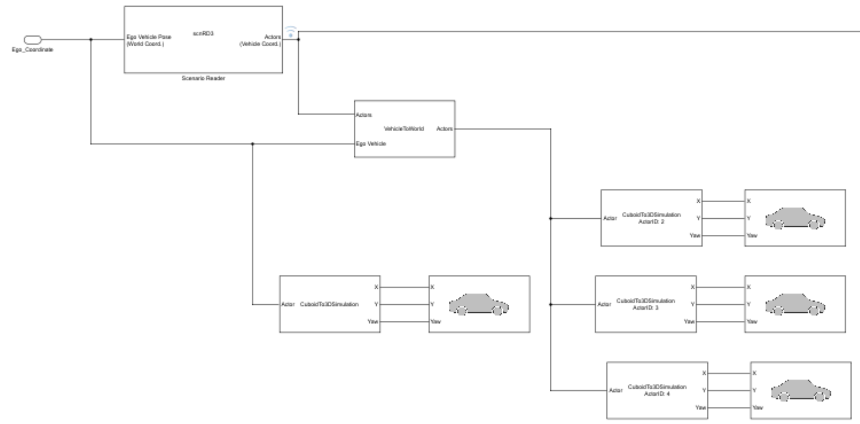
End-to-End Closed Loop Simulation with Autonomous Driving Toolbox

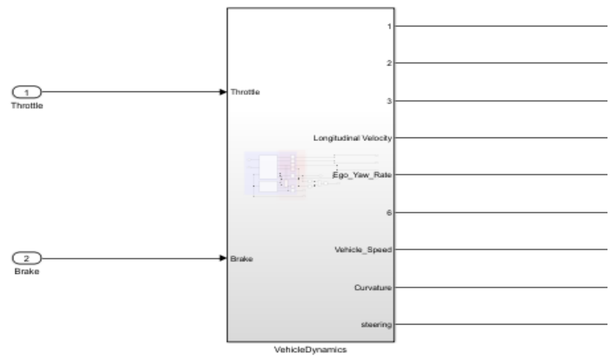
Sensor Fusion with Situation Assessment and AEB





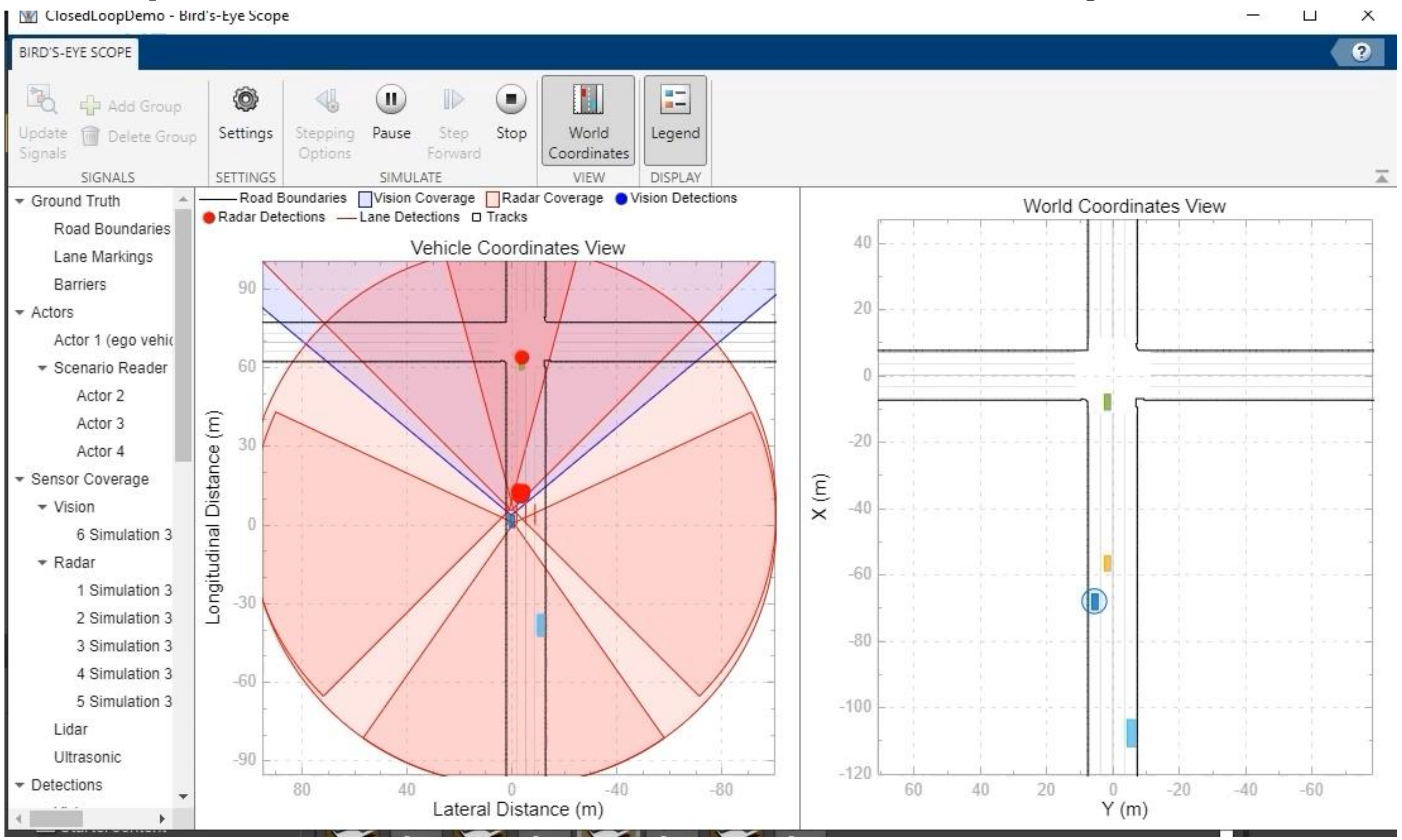
Unreal Engine Integration with Sensors and Perception Software





Vehicle Dynamics model

Closed Loop Demo – Output in 3D Simulation and BirdsEyePlot



LIGHTING NEEDS TO BE REBUILT (1,572 unbuilt objects)
'DisableAllScreenMessages' to suppress

KPIT

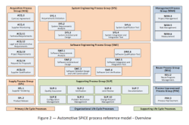
Validation framework advantages

AD components like - sensors, perception algorithm, sensor fusion, situation assessment, AD/ADAS control features, actuation controls are **integrated in a single framework**.

Since all the AD systems are **integrated in a single IDE**, the biggest challenge of integrating multiple tools working in harmony is handled.

This end-to-end validation framework helps in understanding the impact of one system over the other. The framework provides a **holistic understanding of the system functionality and performance** when working in harmony with other integrated systems.

The framework is made **flexible**, such that any AD component can be **plugged in for validation**. Working on an agile methodology, this **reduces the dependency on other systems**, enabling faster time to market.



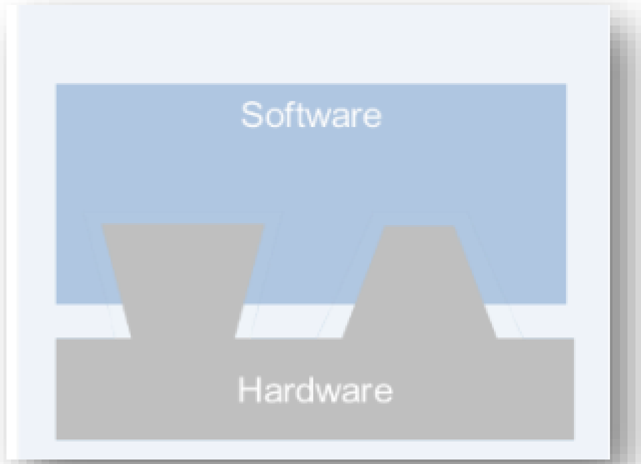
This framework acts as a platform for **ASPICE SWE.5 integrated tes**ting

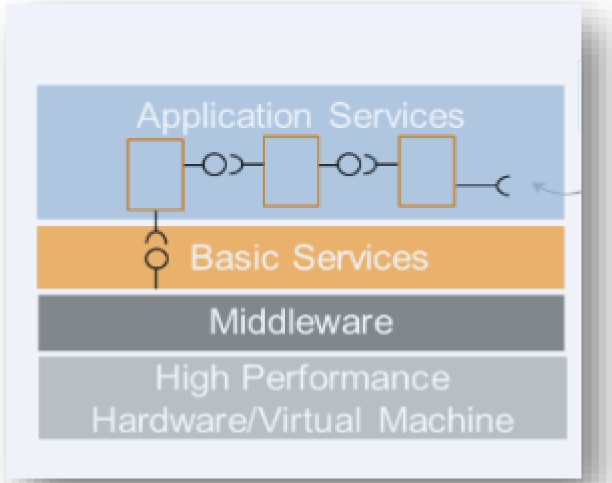
Eliminates 90% of software integration challenges.

Time and cost saving.

Legacy to SOA Integration Testing

Classic Vs Service oriented architecture





Challenges with Legacy architecture

- ✓ Classic software are highly coupled software and do not follow standard API's. Which makes ***it difficult for any software update or reusability***.
- ✓ Software in classic architecture communicates using ***signals***

Advantages of SOA Architecture

- ✓ The Software oriented architecture provides ***flexibility and scalability*** in processing distribution and compute resource allocations required for autonomous driving applications. This helps to update and upgrade adaptive ECU software securely even after its release.
- ✓ In SOA the software communication through ***services***
  - Events
  - Methods
  - Fields

Need for legacy + SOA configuration

- Migration from classic architecture to SOA is a **long process**.
- In most cases legacy component is **field proven**
- In most SDV cases, the **safety critical component is maintained as legacy**, till SOA is fully validated.
- **Reusing of Legacy** software is required till service-oriented application is developed.
- **Reduces application revalidation time** and brings product faster to the market.

SOA migration process

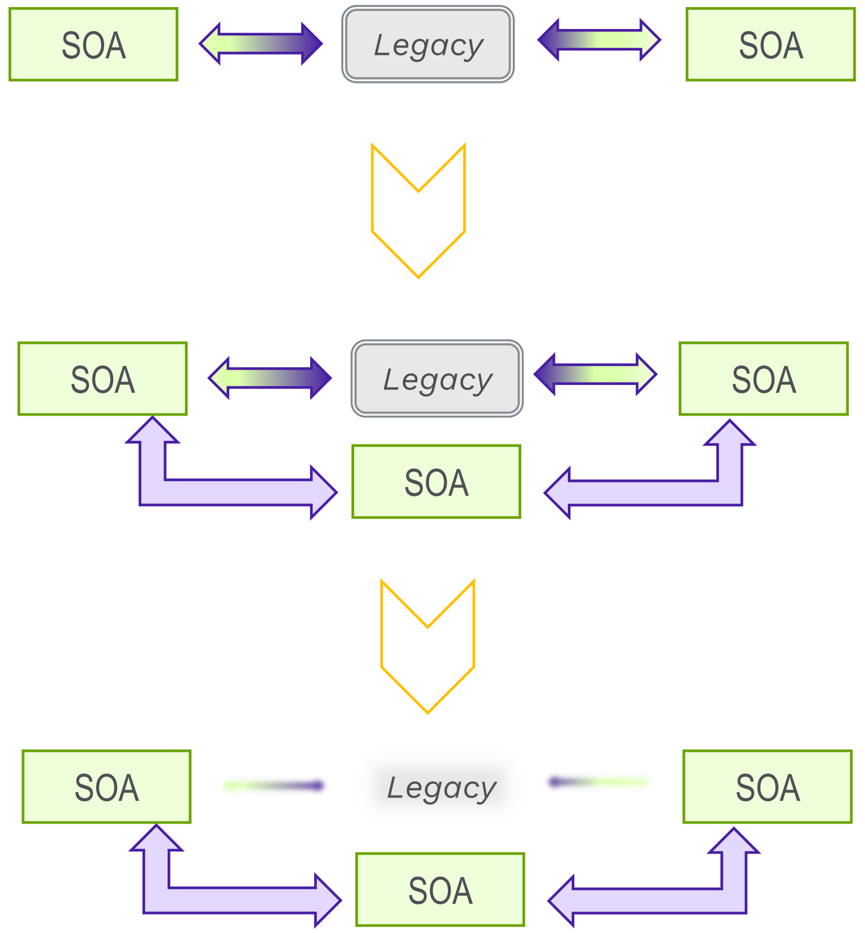
- SOA migration can be performed as **step-by-step process**;

  i. Legacy component to be converted as **service-based component**.

  ii. Co- existence of legacy component with SOA component needs to be validated.

  iii. Develop **equivalent SOA component and validate in shadow mode**

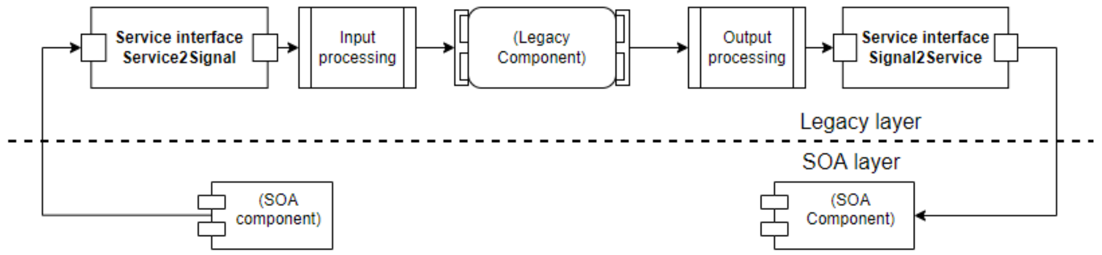
  iv. Replace legacy with SOA component



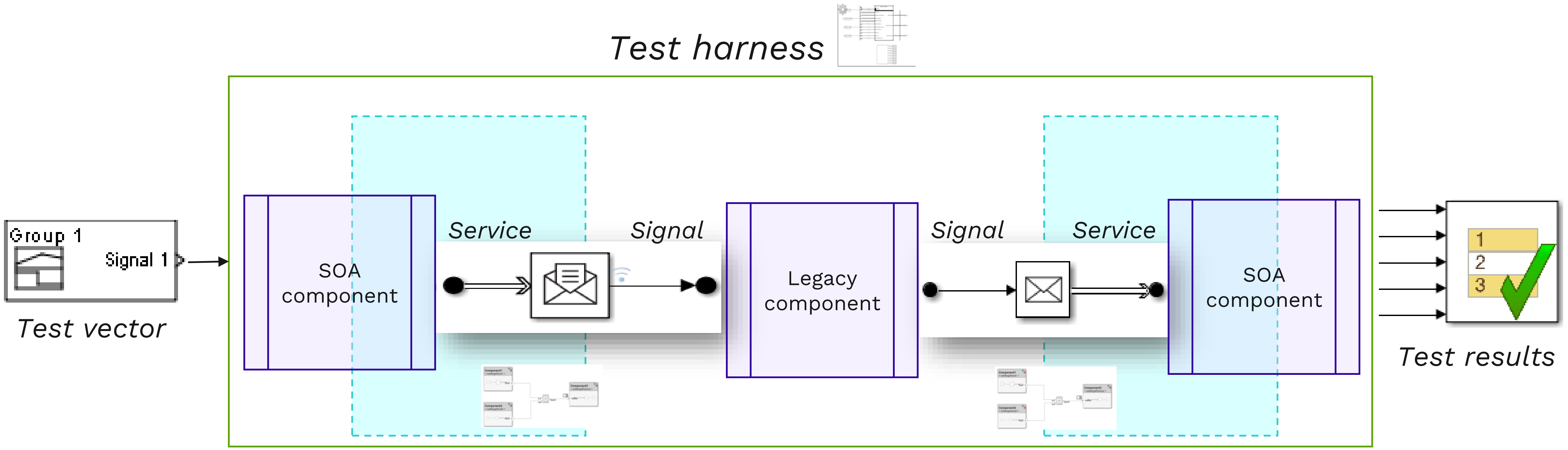
Co-simulation of legacy and SOA

- For **validation of legacy component and SOA in SIL**, **System composer from Mathworks** is used to integrate legacy software with SOA component.
- The services are converted as signals and vice versa while interfacing to legacy component.
- SOA and legacy component communicates through a virtual middleware, which is simulated in System composer environment.

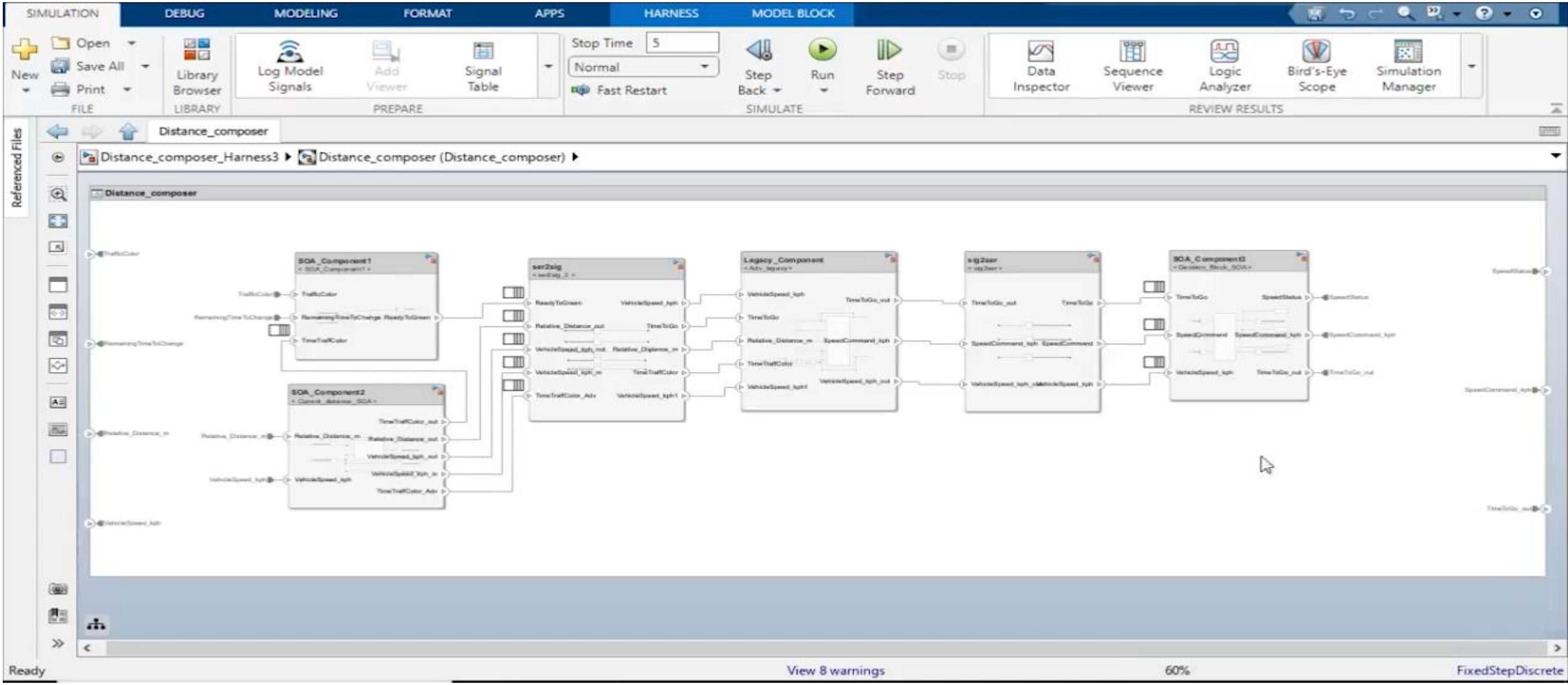
Legacy modules integration to SOA components



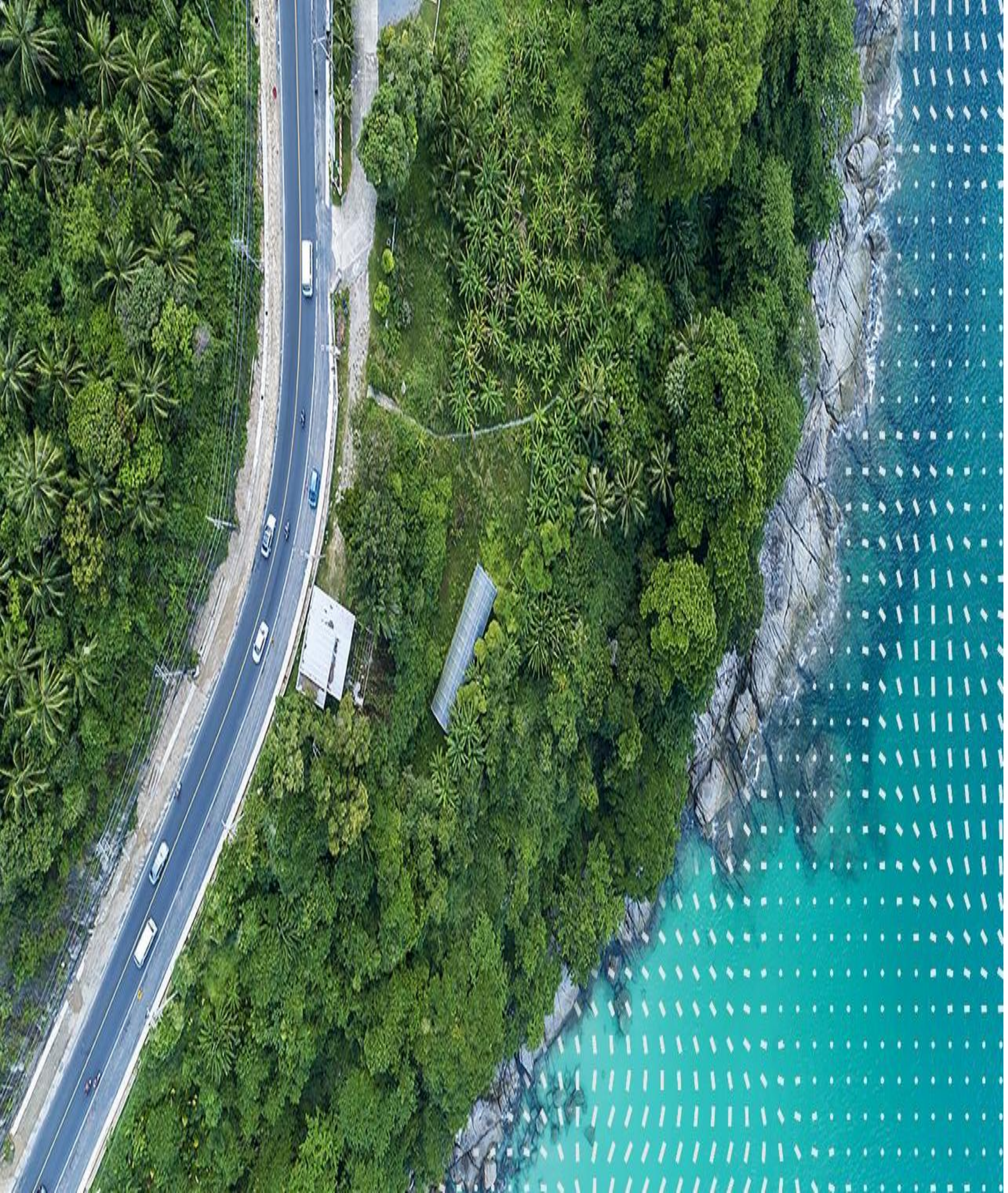
Co-simulation of legacy and SOA



Case study: POC of co-simulation



Queries ?



THANK YOU