2022 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
MODELING SIMULATION AND SOFTWARE (MS2) TECHNICAL SESSION
AUGUST 16-18, 2022 - NOVI, MICHIGAN

Real-time, Closed-Loop and Physics-based Modeling and Simulation System for Unmanned Ground Vehicles in Unstructured Terrain Environments

Samuel Misko[1], Arnold Free, PhD[2], Shiva Sivashankar, PhD[3], Torsten Kluge[4], Vladimir Vantsevich, PhD[5], Martin Hirshkorn, PhD[2], Andres Morales[1], James Michael Brascome[1], Shayla Rose[1], Nic Bowen[1], Siyan Zhang, PhD[1], Masood Ghasemi, PhD[1], Steven Gardner[1], Pierre Fiorini, PhD[2], Madhurima Maddela, PhD[1], Paramsothy Jayakumar, PhD[6], David Gorsich, PhD[6], Chris Manning[4], Matthias Thurau[4], Nico Rueddenklau[4], Gibin Zachariah[3], Eva Dennis[1], Ian Costello[2]

[1]University of Alabama at Birmingham, Birmingham, AL
[2]Naisense Solutions Inc., Montreal, Canada
[3]Siemens Industry Software Inc., Plano, TX
[4]dSPACE GmbH, Paderborn, Germany
[5]Worcester Polytechnic Institute, Worcester, MA
[6]Ground Vehicle Systems Center (GVSC), Warren, MI

ABSTRACT

To realize the full potential of simulation-based evaluation and validation of autonomous ground vehicle systems, the next generation of modeling and simulation (M&S) solutions must provide real-time closed-loop environments that feature the latest physics-based modeling approaches and simulation solvers. Real-time capabilities enable seamless integration of human-in/on-the-loop training and hardware-in-the-loop evaluation and validation studies. Using an open modular architecture to close the loop between the physics-based solvers and autonomy stack components allows for full simulation of unmanned ground vehicles (UGVs) for comprehensive development, training, and testing of artificial intelligence vehicle-based agents and their human team members.

This paper presents an introduction to a Proof of Concept for such a UGV M&S solution for severe terrain environments with a discussion of simulation results and future research directions. This conceptual approach features: 1) richly detailed severe terrain environments, 2) vehicle systems with multi-body dynamics, 3) Terramechanics-based tire-terrain interactions, 4) physics-based exteroceptive sensor models, 5) modular ROS autonomy components, 6) vehicle energy management and electric motor models, and 7) a user configurable dashboard for co-simulation coordination and model parameterization for automated M&S testing.

Citation: S.Misko, A. Free, S. Sivashankar, T. Kluge, V. Vantsevich, et al. "Real-time, Closed-Loop and Physics-based Modeling and Simulation System for Unmanned Ground Vehicles in Unstructured Terrain Environments," In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 16-18, 2022.

1. OVERVIEW

To realize the full potential of simulation-based evaluation and validation of autonomous ground vehicle systems on unstructured off-road terrain, the next generation of modeling and simulation (M&S) solutions must provide real-time, closed-loop environments that feature the latest physics-based modeling approaches and simulation solvers.

Why Real-time? Primarily, real-time simulation facilitates X-in-the-loop (X = Human, Software, Hardware, Sensor, etc.) testing and evaluation (T&E). This is particularly important for T&E and Verification and Validation (V&V) processes that require the unit under test to be stimulated with realistic dynamics. Human or Driver in-the-loop, or Human on-the-loop for the human-machine teaming applications are a perfect example of where simulation latency (<0.15s), time step (>60Hz), and fidelity must be sufficient to provide realistic operational dynamics and challenges.

Why Closed-Loop? Closed-loop simulation facilitates the use of one or more autonomous agents as well as to allow for X-in-the-loop T&E and V&V. These autonomous agents require multiple interconnected feedback loops and access to simulated sensor data for real-time inference and control operations, as well as ground truth for off-line and/or real-time training. Furthermore, the M&S system architecture must be modular to allow for modular integration of disparate X-in-loop systems.

Why Physics-based? All models/solvers must utilize physics-based approaches to accurately generate the widest possible range of simulated conditions, interactive and coupled dynamics, and environmental interactions. Modeling of highly diverse and complex situations are required for T&E and V&V for unmanned ground vehicles in military relevant operational scenarios based in off-road unstructured terrain environments.

Why M&S? M&S, when done right, provides on-demand access to the maximum number of military relevant operational conditions/environments for: training warfighters, to evaluate conceptual and fielded vehicle performance, to train and evaluate AI/ML approaches to autonomy, training/testing/evaluation of new human-machine teaming approaches.

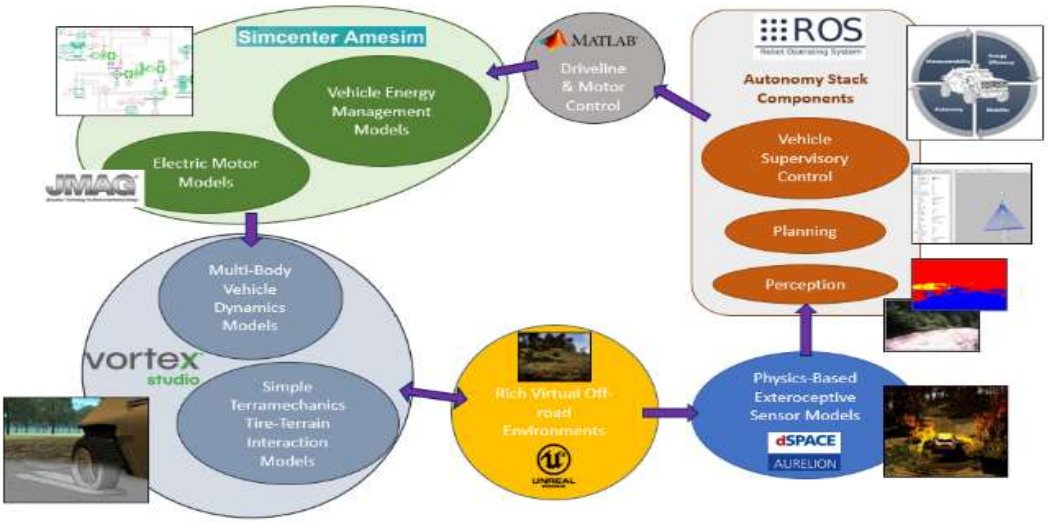

Figure 1: Overview of Proof-of-Concept Co-Simulation Architecture

This paper provides a overview of the UAB AVM team's effort to develop a proof-of-concept M&S system (Figure 1) that can be used to: (1) fully define the requirements and operations, and (2) provide a starting point for the full implementation of a modular, real-time, closed-loop simulator solution for manned and unmanned vehicle simulation in unstructured off-road environments.

2. SOFTWARE COMPONENTS

As shown in Figure 1, there are six main software components linked together to realize this comprehensive UGV M&S tool: Vortex Studio, dSPACE AURELION, Unreal Engine 4, Robotic Operating System, Matlab, and Simcenter Amesim. These modular software components are linked together with a co-simulation framework to enable real-time, closed-loop performance.

2.1. Vortex Studio: Vehicle Dynamics & Terramechanics Models

CM-Lab's Vortex Studio provides real-time multi-body dynamics and tire-terrain interaction modeling capabilities.

2.1.1 Vehicle Topology

The topology of a typical internal combustion engine (ICE) wheeled vehicle modeled with in Vortex Studio is shown below in Fig. 1, where the red lines represent fixed connections, and the blue lines represent power coupling through rotating shafts:

Using this topology with the standard components results in a vehicle with approximately 35 rigid bodies, which is be capable of being simulated between 60 to 500 Hz.

2.1.2 Wheel/Ground Modelling

Several different models can be used for wheel/ground interaction, which can be divided into hard and soft ground models.

The hard ground models (representing wheel behavior on rigid surfaces like pavement, concrete, or hard-packed gravel) include Magic Formula, Pacejka Magic Formula, Composite Slip, Fiala and Coulomb.

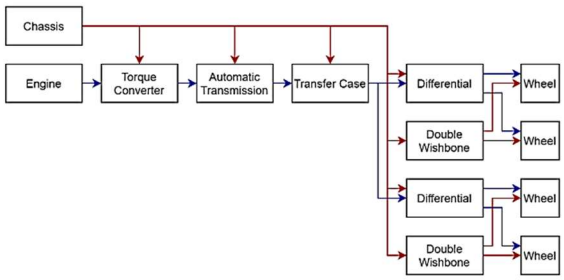

Figure 2: Typical ICE Vehicle Topology

Soft ground models (representing behavior on soft surfaces like soil, sand and snow) can be defined from a variety of models based on terramechanics, which contain two components, a pressure/sinkage model and a shear strain model. The available pressure sinkage models are, Bekker, Wong, Reece, Muskeg, Snow.

These soft ground models can also be used to permanently deform the ground when driving, so that subsequent wheels will pass through the ruts left by other wheels.

In addition to the included tire models, custom tire models can be defined and added and used in the simulation. These models use a provided interface to calculate the reaction force on the wheel given the position and velocity of the wheel and contacting surface.

2.1.3 Electric Vehicles

Electric vehicles can consist of electric motors with a wide variety of drive trains, ranging from single motors with a drive train that otherwise resembles an ICE, to hybrid drive trains of electric motors and ICEs to electric motors directly driving the wheels. It is possible to model any combination of motors and drive train components, such as gear reductions (planetary or axial gears) and power splitters (transfer cases or differentials).

There is currently no library of predefined electric motor models, but a model can easily be defined using Python scripts, C++ code or

co-simulation with an external modelling tool. Generally, the model will define the torque applied by the motor given the motor speed, demand signal, and vehicle energy management component conditions (e.g. temperature/state of various powertrain components).

2.1.4 Considerations for Terrain Models

In order to simulate the physics of the tire-terrain interaction with Vortex Studio, the topography of the terrain and the terrain's mechanical parameters must be defined and modeled to allow the Vortex Studio physics engine to perform the necessary calculation of resultant forces. At each time step, the volume of the intersection between the tire geometry and the terrain topography is calculated. This interference volume is then used to solve for the resultant forces (based on the selected hard or soft ground model) and applied to the vehicle's multi-body dynamics model at each time step. See Section 2.3 for more information on terrain modeling.

2.2. dSPACE AURELION: Exteroceptive Sensor Models

Modern exteroceptive sensors produce a large amount of data per timestep, and represent the sensory perception of the vehicle to observe its surroundings. AURELION provides real-time physics-based exteroceptive sensor modeling and simulation capabilities.

2.2.1 LiDAR Model

For LiDAR, light pulses are emitted outside the visible range and the reflections are sensed. The intensity of the reflection and the time interval between the transmission and reception processes provide information about the geometry and surface properties of the environment.

In order to achieve the broadest possible coverage of the LiDAR sensors available on the market, the simulation of a LiDAR is organized in two stages. In the first stage, the ray tracer determines the point cloud and stores it in an abstracted data structure. Both the execution of the ray tracer and the data structure itself are managed on the GPU, enabling real-time applications. This step is largely independent of technical details of the specific sensor. Only the pattern (firing directions, time intervals between the firings, initial intensities of the firings) and some further parameters (max. range, ...) are necessary.

In the second stage the output behavior of the sensor includes the implementation of the data output protocol, which is defined as interface between sensor and control unit. In the context of LiDAR, (automotive) ethernet is usually used as the communication hardware in order to support the high data rates. In AURELION this step is called postprocessing. To implement post-processing, a complete protocol specification is required, which is usually made freely available by the

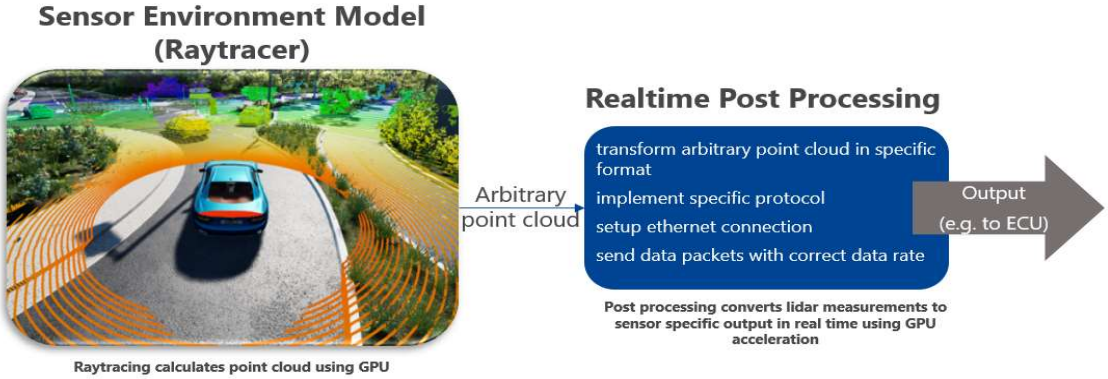

Figure 3: Simulation of LiDAR Sensors in AURELION

sensor manufacturer. Essential elements of

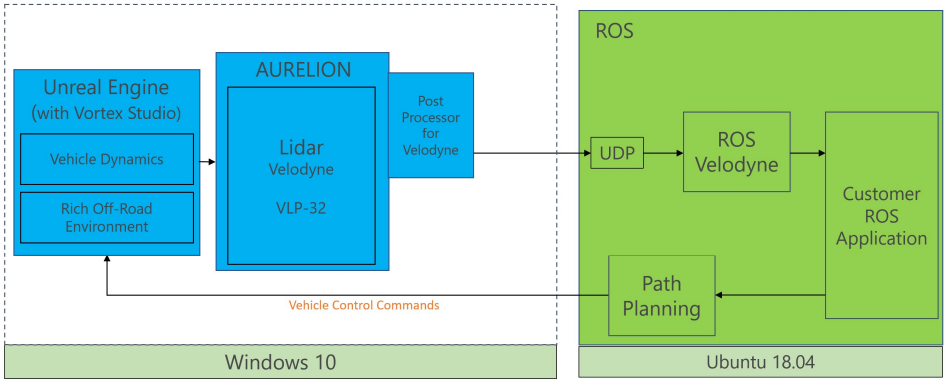

Figure 4. LiDAR Sensor Data Flow Architecture

this specification are the used transport protocols (e.g. TCP, UDP) and possible operation modes (e.g. Frames per Second, Single/Dual Return). To achieve real-time capabilities, the postprocessing may optionally be managed by the GPU, too. This system architecture is shown in Figure 3.

dSPACE AURELION, can simulate several different LiDAR, RADAR, and Camera sensors out of the box. In addition to the sensors that are already supported, other sensor models can be created, given the correct information. The Ouster OS-1 LiDAR sensor is an example of a sensor that is not included with AURELION but can be added alongside the Velodyne VLP-16, VLP-32C, and HDL-64E. To create an accurate sensor model, there are several sensor-specific attributes that need to be defined such as:

- Range
- Update Rate
- Field of View (Horizontal and Vertical)
- Resolution (Azimuth, Elevation)
- Output type

dSPACE then provides a new sensor model based on these attributes to the customer. The output of the new model can be in a few different formats including: an object list, a target list, and raw point clouds. For point cloud output, the data from the simulated sensors is converted to the format specified by the vendor (e.g. Ouster or Velodyne) in the post-processing step. Once converted, the output is indistinguishable from the output of a physical LiDAR sensor. The LiDAR sensors in AURELION are most commonly interfaced via a UDP connection.

In this application, the LiDAR data is helping to inform the autonomous local path planning of the autonomous vehicle based on an autonomy stack implemented in the Robotic Operating System (ROS) framework. To facilitate this, the UDP packets are then processed into a standardized ROS PointCloud2 message via a vendor-specific conversion block. Following this conversion into the PointCloud2 message type, the LiDAR data can be run through autonomous perception or thresholding algorithms for generation of costmaps and/or occupancy grids to inform the autonomous path planning (Figure 4).

2.2.2 Hardware-In-the-Loop for Stereo Camera Models

In some instances, it is not feasible to run all sensor model components in a virtual/SIL environment; such as in cases where real hardware effects are desired, a component is delivered as a closed subsystem from a supplier, or it is not easy to virtualize some components (such as FPGA based algorithms). In these cases, a Hardware-In-the-Loop (HIL) setup is required where the real sensor hardware is used to close the loop between simulation and the control stack.

dSPACE systems provide several ways to inject data generated by AURELION into the

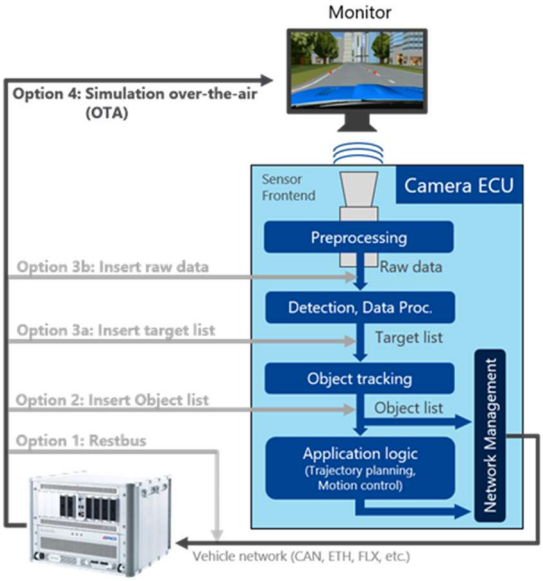

Figure 5: Options for injecting sensor data into the system

system under test (Figure 5). For determination of what level of simulation/data insertion would be utilized, the following needs to be considered:

- Physical/protocol interface requirements:
  - Is this a closed subsystem? Are there internal technical details to enable data injection at different internal points?
  - Are the protocol requirements known for different interface levels?
  - What are the physical layer specifications?
  - Does my test system support these?
- Desired system under test:
  - Is testing desired on the object detection algorithms? If so, the data needs to be injected before that operation.
  - If just the application logic needs to be tested, the object list information is injected.

Determining the data injection/simulation level for stereo cameras can be complicated. Especially in the case of this program, where it is expected that industry will bring proprietary stereo disparity algorithms to contribute. Two approaches fit well with these constraints. The first is an over the air (OTA) solution for stereo cameras. In this case, the test system needs to provide two different images – one each for the left and right cameras. This can be accomplished through an autostereoscopic screen so that the left and right cameras are viewing different pixels. In this case, the images can detect any changes in the simulated environment including lighting conditions, additional actors/objects in the scene, and different weather conditions. This OTA system overview is shown in Figure 6.

The second approach is a raw data injection method for both the left and right cameras. In this scenario, the test system must simulate the lens and imagers of the stereo cameras and feed raw data into the image processor. This can be done over a variety of protocols including GMSL2, FPD-Link III, MIPI A-

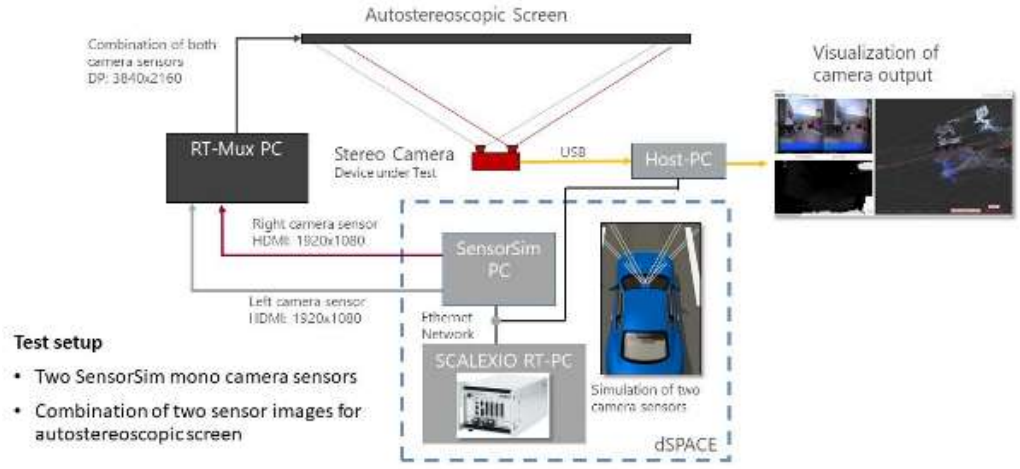

Figure 6: System overview for over the air (OTA) stereo camera data injection

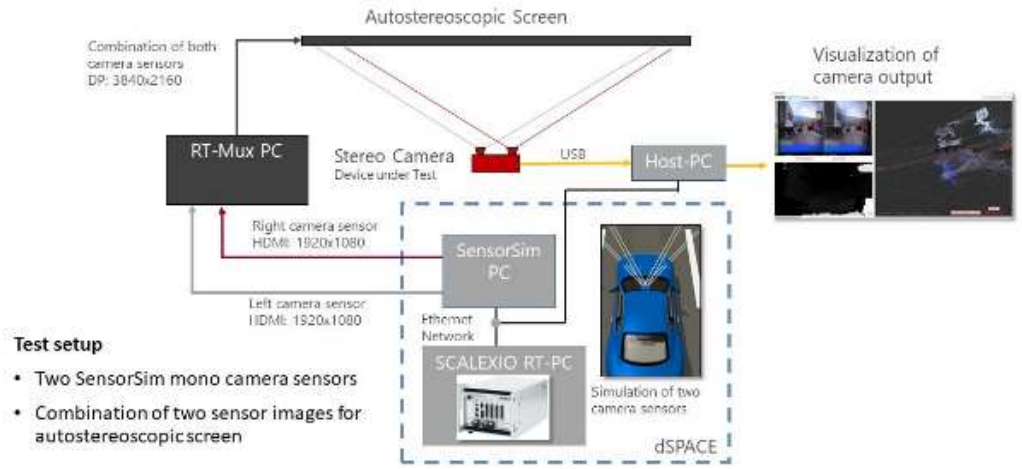

Figure 7: System overview for over the air (OTA) stereo camera data injection

PHY, CSI-2, etc. In these cases, a much more detailed knowledge of the camera specifications is required to develop the test system. But it also results in benefits for test cases. With a raw data interface, effects such as brightness/exposure, pixel errors/patterns, and other imager effects and failures can be simulated. This system diagram is shown in Figure 7.

2.2.3 AURELION Ingestion of Unreal Scenes

The simulation platform from dSPACE, called AURELION, includes sensor simulations for multiple types of sensors, such as configurable camera, LiDAR and RADAR sensors. The configuration defines the behavior of the sensor itself so that devices of specific vendors can be simulated as well as a post-processing to alter the signal as needed.

Such sensors interact with the environment they are in. For example, metallic objects give a vastly different sensor response than plastics, wood or glass for each sensor type. Unreal Engine 4 is the basis of the AURELION simulation platform. UE4 follows the state-of-the-art Physical Based Rendering (PBR) approach for the camera sensor image. This is basically what you see on the screen as a representation of what a camera or the human eye would see.

The camera PBR approach needs 3D objects consisting of geometry (polygons) and textures (images on the polygons). The PBR approach extends the traditional rendering approaches in which you would use a limited number of textures, such as a color-texture with additional texture-channels such as the metalness, specularity, roughness or emission. All the data is needed to create a realistic camera image.

The Unreal PBR approach must be extended further to support simulation of other sensor types. The texture-channels are still too limited and restricted to one-dimensional scalar values. A more realistic approach to describe the behavior of a material is, for example, to model it as a bidirectional reflectance distribution function (BRDF), which is hardly possible to encode in a texture image but needed to calculate a realistic LiDAR and RADAR sensor response. dSPACE AURELION therefore defines physical material properties externally and only links the material definition with a Material-ID, which is linked on the 3D Object similar to other texture channels in the PBR approach. The sensor simulation then can use raytracing approaches to find contact points on the 3D object geometry and combine that with the

physical properties linked through the Material-ID. This offers huge advantages such as flexibility in the number of used material descriptions and the ability to change those descriptions externally without touching the 3D scenes.

The workflow for generation of these UE4-based scenes for AURELION, requires first, the generation of a PBR-ready 3D scene. To produce such a scene, the following steps are required:

- Move 3D objects and scenes into a separated Unreal content plugin
- Add Material-IDs to 3D objects
- Runtime Virtual Texturing is not supported by AURELION
- The Level of Details (LODs, number of polygons) of 3D objects needed to be adjusted to achieve real-time capabilities. LiDAR and RADAR can use different LODs than the camera, so that the camera images are not affected.
- Unreal optimizes some systems for the camera, so that objects are not accessible by other sensors:
    - Convert "procedural foliage" to "hierarchical instanced static meshes"
    - Copy and convert "landscapes" to "static meshes" for LiDAR/RADAR

These UE4 scenes are then compiled into a PAK-file, which is an Unreal-specific workflow for encryption and optimization of simulation performance. As the scenes get larger, it is required that AURELION support an UE4 feature called level streaming, which is a feature currently under development. Level streaming allows for the division of very large scenes with extensive detail to be broken into smaller tiles. This feature optimizes the simulation performance by loading or removing each tile from the computer's memory during runtime to based on the current position of the vehicle. It is therefore possible to drive through large worlds with the high level of detail required for accurate sensor simulation.

2.3. Unreal Engine 4: Environment Models

Modeling of rich unstructured, off-road environments can best be accomplished Unreal Engine 4 (UE4) by leveraging a number of tools for procedural generation of meshes for vegetation, cultural objects, and static/dynamic object and actors (called "assets" in UE4); as well as, tools such as World Machine to accurately model geo-specific terrain profiles and guide the placement and distribution of the aforementioned procedurally generated assets. For modeling 3D assets, geometric primitives can be used to model basic shapes like boxes and cylinders. These can be useful to model structures like buildings, curbs and tree trunks that closely match the shapes. Terrains can be modelled as triangle meshes, height fields or geometric primitives, or a combination of the three.

Height fields are the only geometry type that supports real-time deformable terrain behavior, and can also be of high enough resolution to capture small terrain features. However, they have two main limitations:

1) fixed resolution (horizontal spacing) for the entire field – meaning if a high resolution is chosen to capture small details, that high resolution persists throughout the entire height field; and
2) can not model perfectly vertical terrain features – as you can not have vertices placed directly above others.

The optimal resolution of the terrain depends on the surface being modelled. Higher resolution generally represents the shape of small features in the terrain more accurately, but has a performance cost. Any resolution below about 1/4 of the size of the tire contact patch provides little improvement, and the high resolution is only needed in areas that have small features (rocks, bumps) that need to be considered; and provides no benefit for large, flat areas.

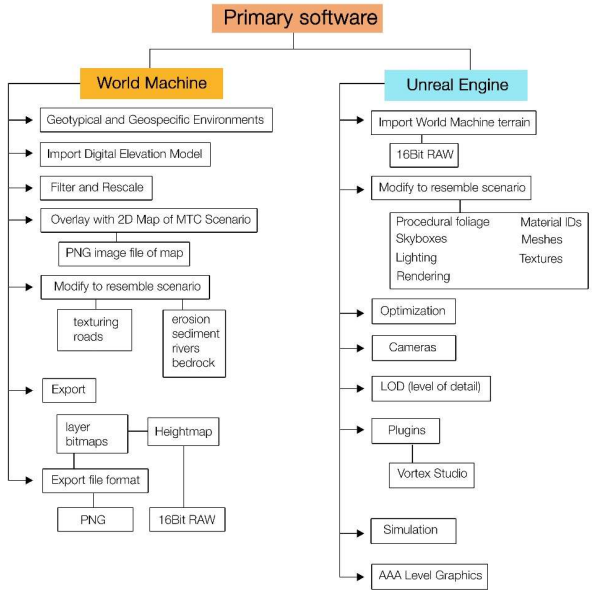

Figure 8. Toolchain for Environment Modeling

2.3.1 Workflow for Building Geotypical and Geospecific Environments

Figure 8 shows the summary toolchain for highly automated scene generation for the modeling of unstructured environments suitable for use in vehicle performance simulations.

World Machine is a robust tool for visualizing terrain data, and procedurally generating geotypical terrain. In software tools such as World Machine, existing elevation data can be pulled down from public sources as GIS data or entirely new terrains can be generated. In both cases, World Machine allows for the generation of natural topographic features based on estimations of complex natural processes such as slope erosion, river channeling, snowpack, and terracing.

Running these processes also generates output data, for example, a map (also called masks) of where rock debris was removed from and subsequently deposited during the course of the erosion feature generation, or where soil moisture is higher due to runoff gathering in concave regions. These maps can be used to differentiate terrain types in the final visualization environment using unique textures for forest loam, mossy stone, loose gravel, etc.

These same maps can be used to populate the bare terrain with natural objects ranging in scale from pebbles and ground vegetation to boulders and forests. Procedural placement tools are used to define the growth pattern and distribution/density of the foliage.

This workflow places an emphasis on procedural tools, where each step of the terrain generation produces output data that can be used by the next step. Procedural generation minimizes the labor of hand placement, allows for extreme scalability, and produces plausible environments that obey geological and biological principles.

Unreal Engine is the industry leader in real-time graphics for games, cinema, and simulation. It has a large marketplace for third-party content, which saves time on environment creation, and it includes plugin support for directly interacting with a range of external programs such as Vortex Studio.

For geospecific terrain modeling, elevation data can be drawn from publicly available point cloud scans where available, or existing digital elevation models (DEMs), such as the ones hosted by the US Geological Survey. GIS data representing foliage distribution, soil type, or other relevant terrain characteristics can be synced with the elevation model.

When generating geospecific terrain models from GIS data for vehicle performance simulations, it is important to down sample the resolution of this data from the typical 1-30m resolution down to approximately 5-25cm. This higher level of accuracy allows for the inclusion of more detailed topographic features and the appropriate level of surface roughness. Our research has shown

that the use of Kriging interpolation techniques for downsampling, in conjunction with roughness estimation and application based on an Fractal noise approaches provides optimal results for the modeling of off-road natural environments for vehicle performance simulation.

When generating geotypical terrain models, the terrain topography can be generated from noise, often a variant of Perlin gradient noise, which is capable of approximating the fractal qualities of real terrain across multiple scales. However, it is more common to use a hybrid approach, where existing elevation data can be used as a starting point which can then be modified as needed.

2.4. Autonomous Perception and Path Planning

The autonomous perception and planning portion of the simulator targets the identification and classification of unstructured, off-road environmental information such that occupancy grid and/or cost map data may be provided to perform autonomous route planning. This function remains a challenging active research domain for achieving true level-5 driverless cars [5]. Contextualizing the scene's dynamic and chaotic conditions quickly and accurately is critical for attaining targeted vehicle velocities safely and reliably.

The task of perception in the context of hostile off-road environments presents specific challenges. The environment is highly unstructured and changing, making differentiation between similar events, objects, and/or environment features very challenging. Other vehicles, people, and trafficable boundaries can be occluded in much more complex patterns than is the case with well-defined roadway environments. Beyond the challenges posed by the natural environment, there is a lack of pertinent datasets needed to meet the complete spectrum of scenarios. Most openly available datasets are built using urban settings, on-road operations, narrow ranges of anticipated events, and are recorded using the most common sensors, making training of customized sensor systems difficult.

2.4.1 Terrain and Object Perception:

Autonomous perception can be broken into sub-functions including object detection for the identification and positioning of elements in the scene, region classification for the evaluation of terrain conditions surrounding the vehicle, and object tracking for tactical decision-making. Each of the perception topics listed above are active research areas, notably in the context of autonomous driving, each with a large corpus of literature [6-8]. Semantic segmentation can be used to address the region classification task such that the complex terrain can be generalized and grouped into classifications representing the estimated trafficability of each section. Semantic segmentation of an image or 3D point cloud consists of labelling each pixel/point of the image/frame according to a set of predefined categories.

For example, a machine learning (ML) network (such as a convolutional neural net or echo state network (ESN)) can be used to perform semantic segmentation on LiDAR point clouds from the simulated sensor data which can then be processed into a occupancy grid for communication to a local planner for navigation and obstacle avoidance (Figure 9). In this example, the ML network can be trained

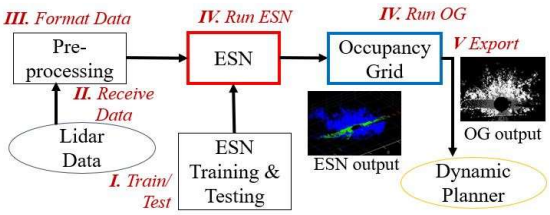

Figure 9. Perception node flow diagram. Simulated Lidar point clouds are inserted to the pre-trained perception algorithm, with the output of a cost map (occupancy grid) sent to the local planner.

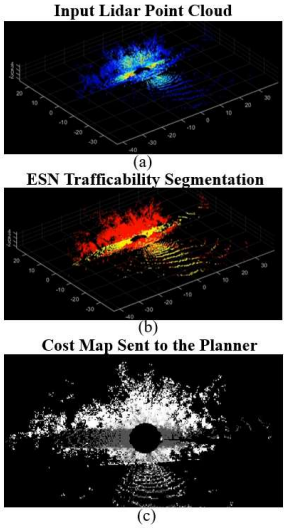

Figure 10. Example perception instance. (a) A LiDAR point cloud is sent to the perception algorithm. (b) The ESN receives the frame and classifies each pixel. (c) The segmented output is converted to a cost map.

to classify four grades of drivability: obstruction, low, moderate, and high trafficability. Figure 10 shows the raw LiDAR point cloud for this example, as well as the classified output of the ESN, and finally its respective cost map. The cost map is a 2D dataset with grayscale values representing the estimated trafficability for the terrain surrounding the vehicle.

2.4.2 Autonomous Planning:

The autonomous planner begins at the global level during initialization, with the current knowledge of the environment and obstacles inside it. It may then sample random points inside the environment and form nodes at these points. As these nodes are created, it constructs a path from the agent's initial location to its goal by connecting these two locations with a series of waypoints, prioritizing the shortest path that does not intersect any obstacles.

In a dynamic planning example, the vehicle then travels along this path by using Q-learning to learn the optimal control inputs to follow this path as closely as possible based on the vehicle's observed kinodynamic constraints. Through this approach, the path planning algorithm doesn't need to have previous knowledge of the vehicle model's dynamics. As the agent travels along this path, it also continues to sample the environment and adjusts the path as necessary if a more optimal one is found.

The agent may not follow its path perfectly. To account for this, the kinodynamic distance (the maximum distance that the agent has deviated from its planned path) is monitored. This kinodynamic distance is also used to artificially expand the size of all obstacles in the environment to ensure that the planned path never collides with the real obstacles. In consequence, the obstacle space changes dynamically, either by obstacles moving, being expanded, or new obstacles becoming visible. If the updated obstacles block the planned path, then the dynamic planner will instead construct a different path using an alternative set of nodes.

2.5. Vehicle Control System

The Vehicle Control System's responsibility is to use the high-level commands from the autonomous planning system to control the vehicle's motion for the following objectives:

- Follow the desired vehicle trajectory closely in terms of:
  - position coordinates
  - longitudinal speed along the path
  - orientation (or yaw) at each positions
- Optimize the performance of the vehicle mobility while maintaining the dynamic stability of the vehicle.
- Minimize powertrain energy utilization on an instantaneous basis or while driving on the desired trajectory.

The Vehicle Control System in the context of the overall simulation architecture is shown in Figure 11. This controller example is implemented in Simulink. Based on the commands and the vehicle state information from the vehicle proprioceptive sensors, the

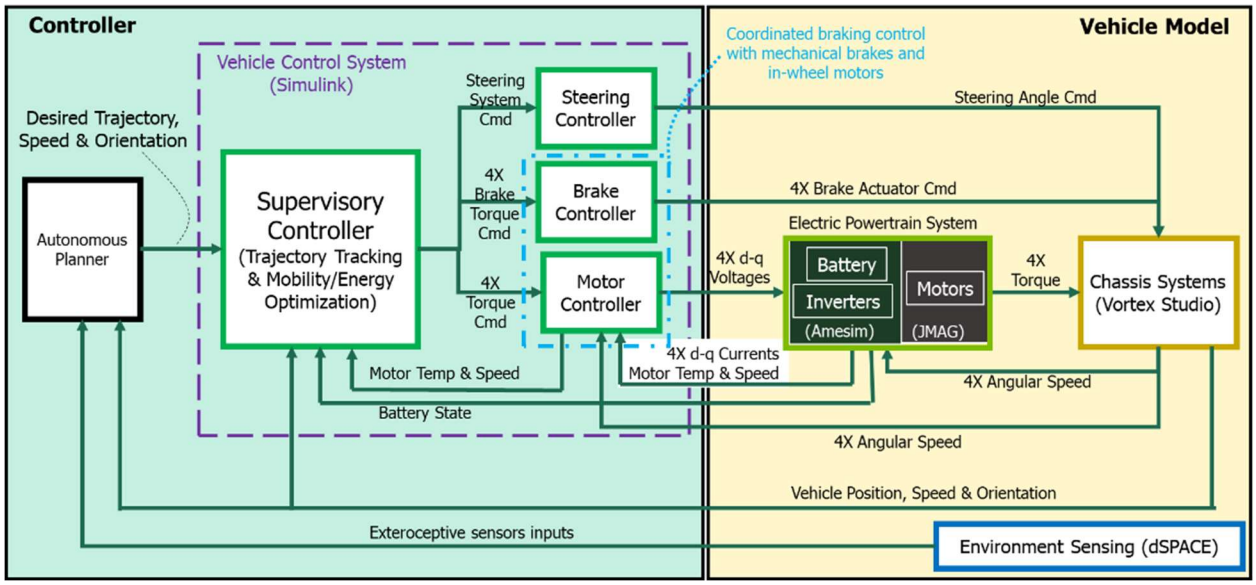

Figure 11: Autonomous Planner & Vehicle Control Architecture

Vehicle Control System maneuvers the vehicle. A two-level hierarchical architecture of the Vehicle Control System provides the means to accomplish the objectives.

- **Supervisory Controller**: Fulfils three main objectives:
  - **Trajectory Tracking**: Required propulsive acceleration vector of the vehicle due to tires' circumferential forces consisting of longitudinal and lateral linear accelerations as well as a yaw angular acceleration.
  - **Trajectory Tracking**: the desired steering angle to track orientation.
  - **Vehicle Energy Management**: Distribute the propulsive acceleration vector among the different in-wheel motors in terms of the required/ reference circumferential forces using an optimal distribution strategy. The goal is to minimize energy consumption from on-board energy storage to maximize range and maintain stable operation of the vehicle. Here, the stable operation of the vehicle refers to the dynamic stability of the chassis where the vehicle does not lose mobility. Furthermore, the supervisory controller monitors the battery temperature, the state-of-charge (SOC), the state-of-power, and decides accordingly how much of a braking torque should be provided by the mechanical brake subsystem to supplement the regenerative braking. For this purpose, it communicates with the mechanical brake controller and provides the reference share of brake torque to be generated by the mechanical brake actuator.
- **Subsystem Controllers:** the demanded maneuvering is accomplished by actuating lower-level vehicle subsystems consisting of:
  - **In-wheel motor controllers** to control the longitudinal acceleration, deceleration, and steering. The electric motors can apply different torques on four wheels for improved mobility and maneuverability, and can operate as generators during braking for Regenerative Braking for improved energy efficiency.
  - **Brake controller** to control the longitudinal deceleration through mechanical wheel braking. Mechanical

braking is used only if the regenerative braking is insufficient for the demanded deceleration. For this purpose, the two subsystems collaborate closely to ensure the deceleration demand is always satisfied efficiently.
    - **Steering control** for lateral motion and yaw rotation via steering system.
- **Wheel Torque Control:** The individual in-wheel motor is responsible for power delivery to/from the wheel and thus, adjusting the angular speed of the wheel. These functionalities are constantly monitored and controlled by the motor control module. The motor control module is designed to regulate the angular speed and torque requirements of an electric motor to power up a driving wheel. Furthermore, it is used to decelerate the wheel through regenerative braking. Finally, the motor control solution ensures that the operations of the electric motor along with other power electronics are optimized and yield optimal energy-efficient performance. The motor controller calculates the phase voltage inputs to the inverter based on torque command signals from the Supervisory Controller.

2.6. Simcenter Amesim: Electric Powertrain Model

In this example application a conceptual off-road vehicle with electric powertrain and four individual wheel motors was developed and based roughly on the US Army's FED Alpha demonstrator vehicle. We have unofficially coined this conceptual vehicle as the "FED Epsilon". The basic components of the electric powertrain of the FED Epsilon are a battery pack, IGBT inverters, PMSM motors, and motor controllers. The inverter is a standard 6-switch Voltage Source Inverter (VSI) that supplies a Pulse Width Modulation (PWM) signal to the motor. There are additional cooling systems modeled to simulate

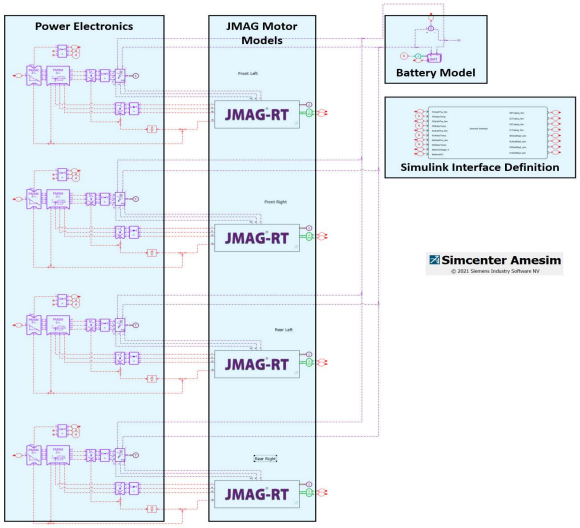

Figure 12. Model of Electric Powertrain with four Individual Wheel Motors (IWM)

the thermal impacts and proper thermal management of the battery and motors. The inverter and battery blocks are modeled using Simcenter Amesim.

The motor model is developed in a FEA/EMAG analysis software called JMAG. JMAG uses multi-physics modeling to develop the inductance characteristics, the current vs torque characteristics, rotation speed vs torque characteristics, iron loss / copper loss characteristics, etc., of a given motor. This JMAG motor model is compiled and exported such that it can be imported into Amesim and then combined in series with the inverter and battery models from pre-existing libraries available in Amesim.

The torque output from each motor is fed to the Vortex Studio; where the multi-body dynamics and tire-terrain interaction models of the FED Epsilon are managed. The Vortex Studio vehicle model provides the angular speed of each wheel as feedback to the individual motor controllers.

Different powertrain configurations such as single axle motor driven system, multi-axle motors, In-Wheel Motors (IWM), internal combustion engines (ICE), etc can be analyzed for different vehicle types and mission profiles. Additionally, the performance and

efficiency of different thermal management systems can be also be similarly modeled and analyzed for their effects.

Figure 12 shows the powertrain model developed in this example in Amesim. Since an Individual Wheel Motor (IWM) powertrain configuration is used for this vehicle, there are four motor and inverter models in the powertrain. The battery model captures the behavior of the battery in terms of the Open Circuit Voltage (OCV) and Internal Resistance which are usually functions of battery temperature and SOC. The required co-simulation interface with Simulink is also shown.

3. CO-SIM FRAMEWORK

3.1. Inter-software Interfaces

Inter-software communication happens through a pub-sub process based on the Robot Operating System (ROS) framework; as well as custom UDP pipelines. In ROS, Applications publish their output to topics, and other applications that take those topics as input subscribe to them. ROS also provides standardized message types so each application, or ROS node, is expecting the data in the same format as it is sent. This allows for the nodes to be modular so pieces can be replaced, so long as their inputs and output are formatted accordingly. The custom UDP pipelines are used to connect Vortex Studio outputs to dSPACE for updating the vehicle position and orientation at each simulation timestep; as well as to provide vehicle position, orientation, and wheel speed data to the vehicle controllers managed in Simulink.

3.2. Dashboard and Parameterization

Various simulation processes need to be stopped and restarted during development and testing. Furthermore, since the processes need to synchronize, the processes should

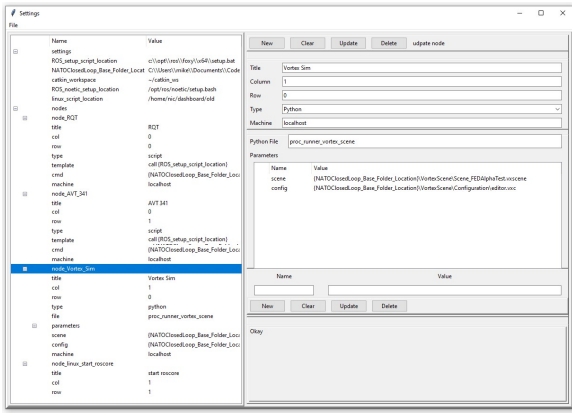

Figure 13. Dashboard User Interface for Process Parameter Configuration

start simultaneously. This can become difficult to perform manually as the number of simulation processes grows. It was advantageous then to develop an application for configuring, starting, stopping, and monitoring all processes.

Each process requires its own configuration and start-up parameters, which are used to define it. Centralizing those configurations simplifies the system management. The solution is a dashboard management system (Figure 13). The dashboard is a fully configurable, generic container that runs each simulation process on a separate thread. On the runtime user interface, each process is given a start and stop button and an output window (Figure 14). When the user clicks the start button for the process, the dashboard starts the process and passes its configuration information to it. The dashboard captures the output and displays it within that process`s panel as the

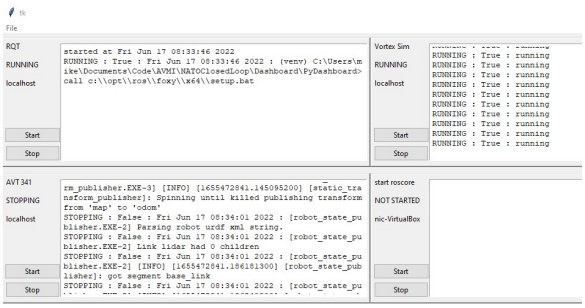

Figure 14. Co-Simulation Dashboard Runtime User Inteface

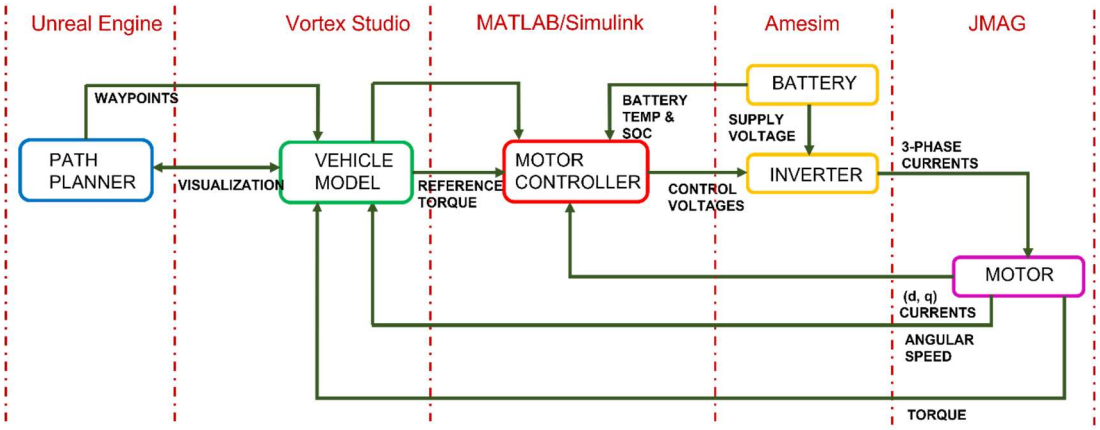

Figure 15. Co-simulation Framework for Closed-loop Electric Vehicle Demonstration

process runs. In addition to the dashboard, ROS also provides a means to quickly swap simulation settings between runs. ROS "Launch" files make it so number of sensors and different processing nodes can be added or removed.

4. PROOF-OF-CONCEPT SIMULATION EXAMPLE

The following proof-of-concept simulation example shows the conceptual electric vehicle with in-wheel motors (FED Epsilon) navigating through a rich off-road environment. This selected example focuses on the ability to provide closed-loop simulation of an electric vehicle with four in-wheel motors. Other proof-of-concept demonstration examples not discussed here, feature LiDAR and Camera sensor models based in AURELION with an autonomy stack based in ROS to control navigation of the vehicle through the landscape while avoiding static and dynamic obstacles, and choosing the optimal path between waypoints based on cost maps with terrain trafficability estimations. The team is actively working on merging these two proof-

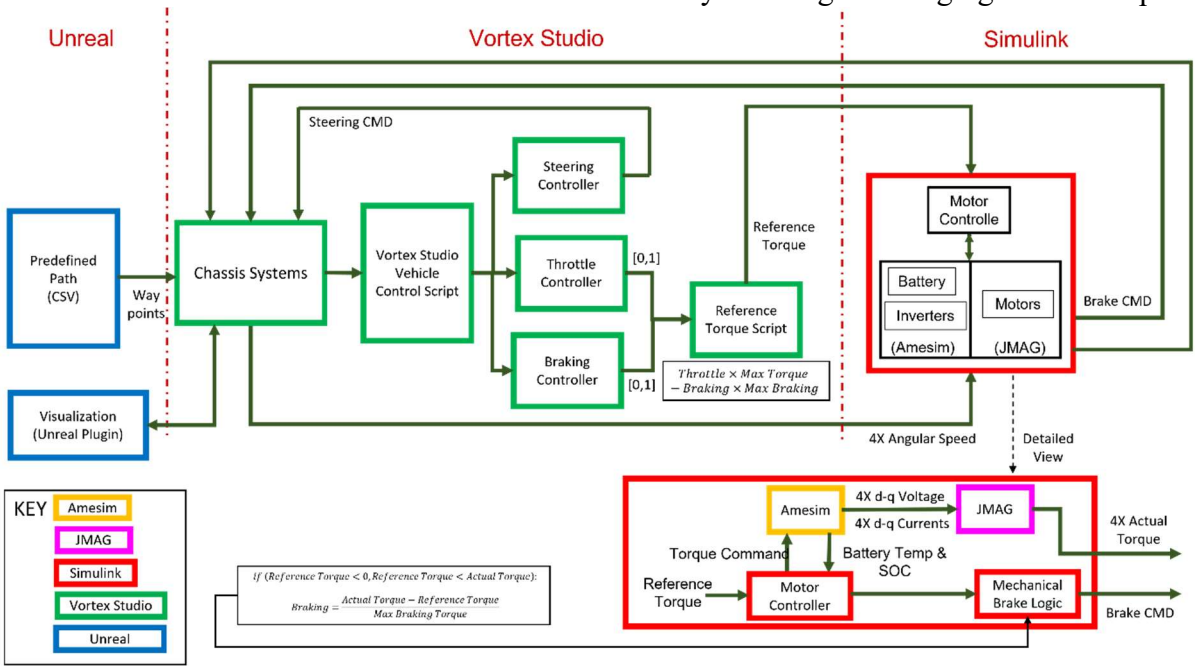

Figure 16: Co-simulation Architecture for Closed-loop Electric Vehicle Demonstration

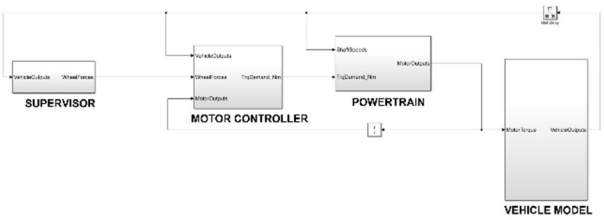

Figure 17. Simulink Master Block Diagram

of-concept demonstrations into the singular co-simulation solution discussed above.

The co-simulation framework and detailed architecture for this closed-loop electric vehicle demonstration has been shown in Figures 15 & 16. Unreal Engine 4 provides the model of the environment, user visualization, and holds the waypoint definitions along a predefined path (this is to be replaced by the output from the autonomous planner based in ROS for full-closed loop autonomy simulation). The Vortex Studio interface is implemented as a native UE4 plug-in, and uses the terrain model as an input to the physics solver for the tire-terrain interaction models that when combined with the multi-body dynamics vehicle model, generates the vehicle position, speed, and orientation at each time step. Also implemented in Vortex Studio are the vehicle level control scripts to provide appropriate inputs to the Steering, Throttle and Braking Controllers which together come up with a desired torque demand for each wheel. This desired (reference) torque command is provided to Simulink as a input, which includes a pre-compiled model of the Amesim vehicle energy management system (e.g. inverters, battery) and four JMAG motor models. These models calculated the actual torque available based on thermal and electrical supply constraints as well as each motors' angular speed provided by a UDP connection to Vortex Studio.

The Simulink Master model shown in Figure 17 shows the VehicleModel block through which houses the UDP communication protocol interface to Vortex Studio, and the PowerTrain block that houses the pre-compiled S-function for the Amesim/JMAG models.

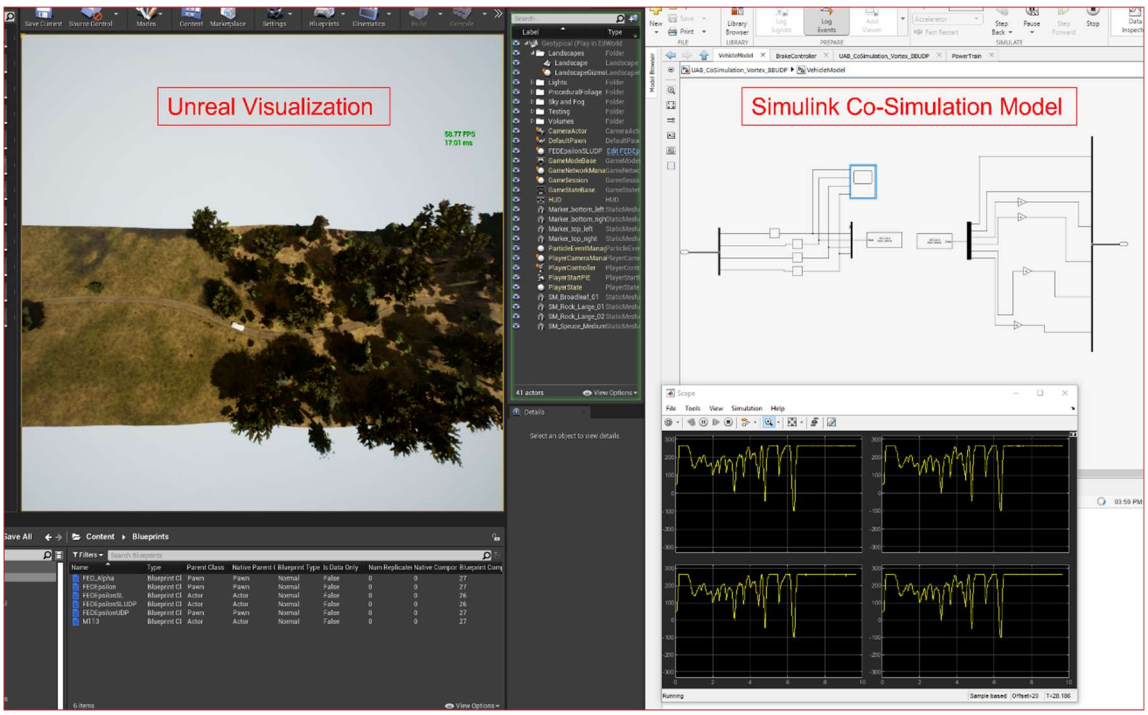

Figure 18: Simulink model and Unreal visualization running parallelly

In Figure 18 two parallel windows can be seen - to the right is the Simulink co-simulation model with real-time data output graphs.In Figure 19, the user can view the progression of the vehicle traversing the UE4 test environment. On the left, the movement of the vehicle can be visualized in Unreal from a birds eye perspective. The path taken by the vehicle in relation to the waypoints is shown in Figure 20, where it can be seen that the vehicle follows the waypoints faithfully. The velocity profile of the vehicle can be seen in Figure 21. The comparison between the torque demand and the actual torque at the wheel is shown in Figure 22. The depreciation of the battery's State of Charge (SoC) is shown in Figure 23.

Figure 19: FED Epsilon Traversing Path in UE4

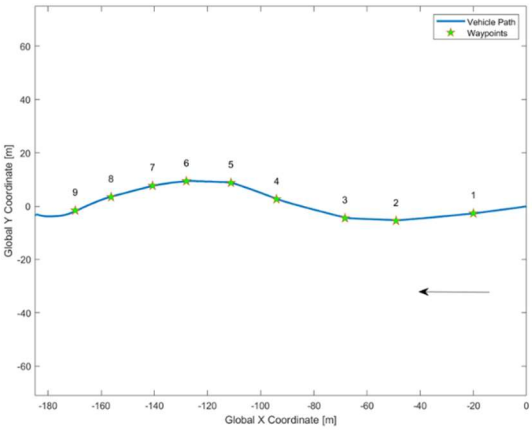

Figure 20: FED Epsilon's path and the waypoints

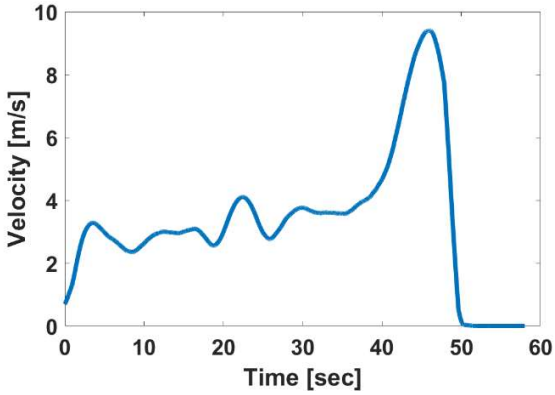

Figure 21: FED Epsilon's velocity

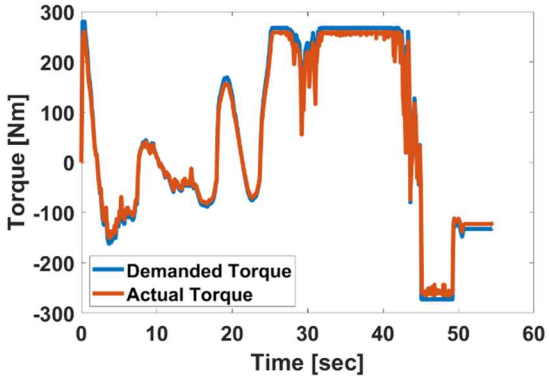

Figure 22: Torque Demand vs Provided

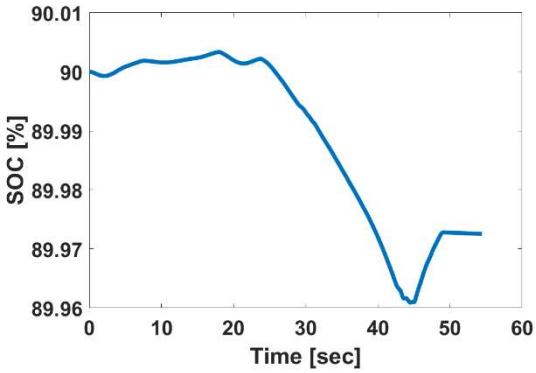

Figure 23: Battery State of Charge (SoC)

5. CONCLUSIONS

Our team is developing a comprehensive co-simulation solution for evaluating unmanned ground vehicle performance in complex unstructured environments using a suite of commercial and open software tools. The

current focus is on the development of a number of proof-of-concept demonstrations that serve as a basis for:

- proving baseline feasibility, and
- defining the requirements and conceptual operation of a deployable modeling and simulation tool.

These specifications and M&S tools are targeted for use by industry and government stakeholders throughout the various developmental and operational evaluation processes that take place as part of the larger acquisition, sustainment, and continuous improvement actions by the US Army's Ground Vehicle Systems Center.

This paper presents a portion of our work in this area, and serves as the introduction of our recent efforts to the larger scientific and governmental community. To date, we have largely proven the baseline feasibility of the larger co-simulation design shown at the beginning of the paper in Figure 1, and are continuously expanding and integrating our various closed-loop proof-of-concept demonstrations while optimizing for:

- real-time operation
- utilization of rich, large-scale unstructured environments
- proven physics-based M&S capabilities
- a closed-loop, modular architecture.

6. FUTURE DIRECTIONS

Next Gen Vehicle R&D. Facilitate development of hybrid and electric vehicles with highly sophisticated power devices, hierarchical control and cyber-physical dynamics requiring continuous, iterative improvement to real-time models to improve accuracy for wider range of conditions and reduce latency for higher bandwidth dynamics. Integrate additional capabilities for modeling of more traditional ICE powertrains with co-simulation interfaces and requirements defined for any additional M&S tools as required. Develop user interfaces for vehicle designers, analysts, etc. to allow for rapid iteration and evaluation of new conceptual vehicle designs optimized for off-road mobility, maneuverability, and energy efficiency.

Advanced Autonomy R&D. Incorporate increasingly detailed and wider range of physics-based models of environmental interactions (tire-terrain interactions, sensor-environment/material interactions) with robust approaches to introduction of noise/uncertainty into all feedback pathways to provide more stochastic/dynamic operating conditions. Develop additional visualizations and user interfaces as required to optimize utilization of the tool for testing, testing, and training autonomous perception, planning, and command/control systems.

Integration with Human User Interfaces and/or Training Environments. To allow for more robust warfighter training and warfighter evaluation of vehicles in many diverse operational scenarios, leveraging the latest in Graphical User Interface design and/or integration with existing synthetic training environments (STEs) can be leveraged to aid in the warfighter ability to quickly develop and execute these scenarios – including new user interfaces for next generation concepts for human-machine teaming.

7. REFERENCES

[1] K. Cakir and A. Sabanovic, "In-wheel Motor Design for Electric Vehicles", AMC'06-Istanbul, 2006, pp. 613-618.

[2] W. Chanpeng and P. Hachanont, "Design of Efficient In-Wheel Motor for Electric Vehicles", Energy Procedia, vol. 56, 2014, Pages 525-531, ISSN 1876-6102.

[3] R. Ford, "In-wheel motors: Beyond torque vectoring, the benefits of independent wheel torque control in non-performance applications", e-mobility Technology International – Powertrain Systems, May 2020.

[4] P. Bhatt, H. Mehar and M. Sahajwani, "Electrical Motors for Electric Vehicle – A Comparative Study", Proceedings of Recent Advances in Interdisciplinary Trends in Engineering & Applications (RAITEA), April 3, 2019.

[5] Jebamikyous and Kashef, "Autonomous Vehicles Perception (AVP) Using Deep Learning: Modeling, Assessment, and Challenges", ACM Computing Survey, vol. 10, 2022

[6] Boukerche and Hou, "Object Detection Using Deep Learning Methods in Traffic Scenarios", ACM Computing Survey, vol. 54, 2021

[7] Lateef and Ruichek, "Survey on Semantic Segmentation using Deep Learning Techniques", Neurocomputing, vol. 338, 201

[8] Cao and Bao, "A Survey On Image Semantic Segmentation Methods With Convolutional Neural Network", Proceedings of CISCE, 2020

[9] Wigness and al., "A RUGD Dataset for Autonomous Navigation and Visual Perception in Unstructured Outdoor Environments", Proceedings IROS, 2019

[10] Jiang and al., "RELLIS-3D Dataset: Data, Benchmarks and Analysis", arXiv, 2020

[11] Gardner, S., Haider, M. R., Smereka, J., Jayakumar, P., Gorsich, D., Moradi, L., & Vantsevich, V. (2021). Rapid High-Dimensional Semantic Segmentation with Echo State Networks. Ground Vehicle Systems Engineering Technology Symposium (GVSETS) - Autonomy, Artificial Intelligence, Robotics (AAIR).

[12] Gardner, S., Haider, M. R., Moradi, L., & Vantsevich, V. (2021). A Modified Echo State Network for Time Independent Image Classification. 64th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS).

[13] Yuan, Y., Chen, X., Chen, X., & Wang, J. (2019). Segmentation transformer: Object-contextual representations for semantic segmentation. arXiv preprint arXiv:1909.11065.

[14] Takikawa, T., Acuna, D., Jampani, V., & Fidler, S. (2019). Gated-SCNN: Gated Shape CNNs for Semantic Segmentation. ICCV.