REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
13-07-2021	Final Report	1-Nov-2020 - 31-May-2021

4. TITLE AND SUBTITLE
Final Report: Development of an Autonomous Off-Road Ground Vehicle Simulator

5a. CONTRACT NUMBER

5b. GRANT NUMBER
W911NF-21-P-0002

5c. PROGRAM ELEMENT NUMBER
665502

6. AUTHORS

5d. PROJECT NUMBER
665502
665502
665502

5e. TASK NUMBER
665502

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES

Advanced Science and Automation Corp.
9714 Oakhaven Ct.

Indianapolis, IN 46256 -1111

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
77857-MA-ST1.1

12. DISTRIBUTION AVAILIBILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Hatem Wasfy
UU	UU	UU	UU		19b. TELEPHONE NUMBER 217-417-3677

RPPR Final Report
as of 14-Jul-2021

Agency Code: 21XD

Proposal Number: 77857MAST1 **Agreement Number: W911NF-21-P-0002**
INVESTIGATOR(S):

Name: Hatem Wasfy
Email: hatemwasfy@ascience.com
Phone Number: 2174173677
Principal: Y

Organization: **Advanced Science and Automation Corp.**
Address: 9714 Oakhaven Ct., Indianapolis, IN 462561111
Country: USA

DUNS Number: 839819179 EIN:
Report Date: 31-May-2021 Date Received: 13-Jul-2021
Final Report for Period Beginning 01-Nov-2020 and Ending 31-May-2021
Title: Development of an Autonomous Off-Road Ground Vehicle Simulator
Begin Performance Period: 01-Nov-2020 **End Performance Period:** 31-May-2021
Report Term: 0-Other

Submitted By: Hatem Wasfy Email: hatemwasfy@ascience.com
Phone: (217) 417-3677

Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: **STEM Participants:** 2

Major Goals: 1. Perform a literature review
ASA will perform a literature review of existing simulation environments for single and multiple agent testing. Examples of these simulation environments are the Automated Driving Toolbox (ADT) developed by MathWorks, the Autonomous Vehicle software developed by Siemens, CARLA, rFpro, metamoto, Orchestra, cognata, ANVEL and CHRONO. Studying these and other solutions in detail will offer insights into better ways of handling various aspects of the AVSP. Furthermore valuable partnerships can be established with market leaders in this field that are identified through the literature review.

2. Produce a prototype of the AVSP
2.1 Develop a prototype of the AVSP as an open-source tool: The AVSP will be developed as an open-source software tool that can easily be used and augmented by other parties. The prototype that will be developed in Phase I of this project will run under the Windows operating system. It will include the user-interface for setting up the simulation and a general networking interface for integrating the various sub-components of the software.
2.2 Integrate the Unreal engine into the AVSP: The Unreal engine will be integrated into the AVSP. The Unreal engine will be used to generate a sample multiple level-of-detail terrain with various features such a hills, mountains, vegetation (grass, shrubs, and trees), rocks, trails, urban obstacles/structures, etc. The objective is to show that the AVSP can produce a photorealistic environment that can mimic a real off-road environment.
2.3 Integrate sensor models into the AVSP: The visible light mono camera and radar sensor models will be integrated into the AVSP. The ray tracing capability of the Unreal engine will be used to simulate those sensors.
2.4 Integrate simple terramechanics in the AVSP: A simple terramechanics capability will be added to the AVSP.
2.5 Simulate 4 vehicles moving and communicating in the AVSP using parallel computing: The AVSP will be used to simulate an autonomous team of 4 vehicles moving along pre-determined paths in a realistic virtual environment. The simulation will make use of parallel computing to demonstrate that tens of vehicles can be simulated in real-time given enough computational power. Each vehicle will communicate with the other vehicles its location, speed, and heading.

3. Write Phase I report: The report will document the developments and results of Phase I.

Accomplishments: The following goals were achieved:
1. Perform a literature review
ASA will perform a literature review of existing simulation environments for single and multiple agent testing. Examples of these simulation environments are the Automated Driving Toolbox (ADT) developed by MathWorks, the Autonomous Vehicle software developed by Siemens, CARLA, rFpro, metamoto, Orchestra, cognata, ANVEL

and CHRONO. Studying these and other solutions in detail will offer insights into better ways of handling various aspects of the AVSP. Furthermore valuable partnerships can be established with market leaders in this field that are identified through the literature review.

2. Produce a prototype of the AVSP
2.1 Develop a prototype of the AVSP as an open-source tool: The AVSP will be developed as an open-source software tool that can easily be used and augmented by other parties. The prototype that will be developed in Phase I of this project will run under the Windows operating system. It will include the user-interface for setting up the simulation and a general networking interface for integrating the various sub-components of the software.
2.2 Integrate the Unreal engine into the AVSP: The Unreal engine will be integrated into the AVSP. The Unreal engine will be used to generate a sample multiple level-of-detail terrain with various features such a hills, mountains, vegetation (grass, shrubs, and trees), rocks, trails, urban obstacles/structures, etc. The objective is to show that the AVSP can produce a photorealistic environment that can mimic a real off-road environment.
2.3 Integrate sensor models into the AVSP: The visible light mono camera and radar sensor models will be integrated into the AVSP. The ray tracing capability of the Unreal engine will be used to simulate those sensors.
2.4 Integrate simple terramechanics in the AVSP: A simple terramechanics capability will be added to the AVSP.
2.5 Simulate 4 vehicles moving and communicating in the AVSP using parallel computing: The AVSP will be used to simulate an autonomous team of 4 vehicles moving along pre-determined paths in a realistic virtual environment. The simulation will make use of parallel computing to demonstrate that tens of vehicles can be simulated in real-time given enough computational power. Each vehicle will communicate with the other vehicles its location, speed, and heading.

3. Write Phase I report: The report will document the developments and results of Phase I.

Training Opportunities: Nothing to Report

Results Dissemination: Nothing to Report

Honors and Awards: Nothing to Report

Protocol Activity Status:

Technology Transfer: Nothing to Report

PARTICIPANTS:

Participant Type: PD/PI
Participant: Hatem Wasfy
Person Months Worked: 5.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Tamer Wasfy
Person Months Worked: 3.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Omar Elmaraghi
Person Months Worked: 1.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Faculty
Participant: Sohel Anwar
Person Months Worked: 1.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Faculty
Participant: Hazim El-Mounayri
Person Months Worked: 1.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Tayabali Kesury
Person Months Worked: 3.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Chris Cardoza
Person Months Worked: 3.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Partners

,

RPPR Final Report

as of 14-Jul-2021

I certify that the information in the report is complete and accurate:
Signature: Hatem Wasfy
Signature Date: 7/13/21 12:26AM

Advanced Science and Automation Corp.

www.ascience.com

9714 Oakhaven Ct., Indianapolis, IN 46256

Army Phase I STTR Contract #: **W911NF-21-P-0002**
Topic number: **A20B-T006**
Proposal Number: **A20B-T006-0305**
Contract Performance Period: **Nov. 1st, 2020 to May 31st 2021**
Total Contract Amount: **$166,496.14**
Amount of funds paid by DFAS to date: **$166,496.14**
Total amount expended to date: **$166,496.14**
Total amount invoiced to date: **$166,496.14**
Number of employees working on the project: **3 at ASA and 4 at IUPUI**

Development of an Autonomous Off-Road Ground Vehicle Simulator

Phase I – Final Report

November 1st, 2020 to May 31st, 2021

Prepared by:
Hatem M. Wasfy (217-417-3677; hatemwasfy@ascience.com) (PI)
Tamer M. Wasfy (757-373-2296; tamerwasfy@ascience.com) (Key Person)

Date: **7/12/2021**

Issuing Government Activity: **US ARMY ACC-APG-RTP W911NF**
Project COR: **Dr. Joseph Myers**
E-mail: **joseph.d.myers8.civ@mail.mil**
Phone: **(919) 549-4245**
Address: **U.S. Army CCDC Army Research Laboratory**
800 Park Office Drive, Suite 4229
Research Triangle Park, NC 27709

Contents

List of Tables

List of Figures

1. Project Objectives

In this STTR project, Advanced Science and Automation Corp. (ASA) will develop a software platform for virtually testing how teams of interconnected automated ground vehicles interact with each other and their environment. The platform will be called the Autonomous Vehicle Simulation Platform (AVSP). The AVSP will be developed as an open source software to allow others to improve upon it and add new capabilities over time. Following are the project tasks that will be performed in Phase I:

1. Perform a literature review

ASA will perform a literature review of existing simulation environments for single and multiple agent testing. Studying these and other solutions in detail will offer insights into better ways of handling various aspects of the AVSP. Furthermore, valuable partnerships can be established with market leaders in this field that will be identified through the literature review.

2. Produce a prototype of the AVSP

2.1 Develop a prototype of the AVSP as an open-source tool: The AVSP will be developed as an open-source software tool that can easily be used and augmented by other parties. The prototype that will be developed in Phase I of this project will run under the Windows operating system. It will include the user-interface for setting up the simulation and a general networking interface for integrating the various sub-components of the software including IVRESS/DIS and the Unreal engine.

2.2 Integrate the Unreal engine into the AVSP: The Unreal engine [1] will be integrated into the AVSP. The Unreal engine will be used to generate a sample multi-level-of-detail terrain with various features such a hills, mountains, vegetation (grass, shrubs, and trees), rocks, trails, urban obstacles/structures, etc. The objective is to show that the AVSP can produce a photorealistic environment that can mimic a real off-road environment.

2.3 Integrate sensor models into the AVSP: The visible light mono camera and radar sensor models will be integrated into the AVSP. The ray tracing capability of the Unreal engine will be used to simulate those sensors.

2.4 Integrate simple terramechanics in the AVSP: A simple terramechanics capability will be added to the AVSP.

2.5 Simulate 4 vehicles moving and communicating in the AVSP using parallel computing: The AVSP will be used to simulate an autonomous team of 4 vehicles moving along pre-determined paths in a realistic virtual environment. The simulation will make use of parallel computing to demonstrate that tens of vehicles can be simulated in real-time given enough computational power. Each vehicle will communicate with the other vehicles its location, speed, and heading.

3. Write Phase I report: The report will document the developments and results of Phase I.

2. Tasks completed to Date

Task 1: Perform a Literature Review

This review will focus on the Tools and platforms that are used to simulate autonomous on-road and off-road vehicles. A simulation platform for autonomous vehicles is defined as a virtual environment for testing and visualizing how autonomous vehicles interact with each other and their surroundings. The interaction of the vehicle with its surrounding can be physical such as the vehicle driving over the terrain and colliding with various objects in the environment such as shrubs, trees, structures, and other vehicles. The vehicle's interaction with its surrounding can also be through the suit of sensors that the vehicle carries such as cameras, LIDAR and Radar. All autonomous vehicle simulation platforms use certain tools and methods to simulate and visualize these interactions. A combination of these tools within the proposed Autonomous Vehicle Simulation Platform (AVSP) will allow the platform to mimic the behavior of many types of autonomous vehicles that are equipped with various types of sensors with different configurations.

Hence this review will attempt to identify the best available tools and methods for simulating the various vehicle sensors. The main capabilities, strengths and weaknesses of the most prominent simulation environments will also be explored in order to identify and include the most useful features of these environments. There are many 3D graphics engines that can be used to generate the Virtual Environment (VE) within which the simulated autonomous vehicles operate, but the most realistic looking ones are the ones that are generated by high end 3D gaming engines. The two most widely used 3D gaming engines are the Unreal [1] and Unity [2] gaming engines. Unreal and Unity have very similar visual capabilities, but while Unreal is open source which makes it easier to interface with and modify, Unity is not. Furthermore, although this might be subject to personal preference, many believe that Unreal is easier to use and learn as compared to Unity.

1.1 Overview of Sensor Simulation Models

Autonomous vehicles typically employ several sensors to help them navigate their environment. These sensors can include cameras, LIDAR, Radar, ultrasonic sensors, GPS, IMU, magnetometer, and sensors for detecting the state of the vehicle's suspension and drivetrain.

1.1.1 Camera

The camera sensor is simulated using a viewport of the virtual environment in which the autonomous vehicle operates. The position, orientation, and zoom of the camera can be adjusted by adjusting that viewport (Figure 1). The camera view(s) can be attached to the vehicle to simulate the sensor suit that is mounted on the vehicle.

Figure 1 Autonomous vehicle simulated using the AVSP showing 3 camera views that are attached to the roof of the vehicle.

a) Mono Camera

This is the simplest type of camera to implement. Its output is generated from the view of a single virtual reality viewport. It should be noted that a given autonomous vehicle can have several mono cameras pointing in different directions (Figure 1).

b) Stereo Camera

A stereo camera uses two mono cameras, also called a stereo pair, with slightly different positions and orientations to obtain a stereo image on the virtual environment. The stereo image can be used by the autonomous vehicle to estimate the relative depth of various objects within the scene [3]. The depth estimates are obtained using a stereo disparity map which is generated by matching corresponding points in the stereo pair. The two images are rectified so that corresponding points in both images are found in the same row in order to reduce the 2D stereo correspondence problem to a 1D problem (Figure 2) [4].

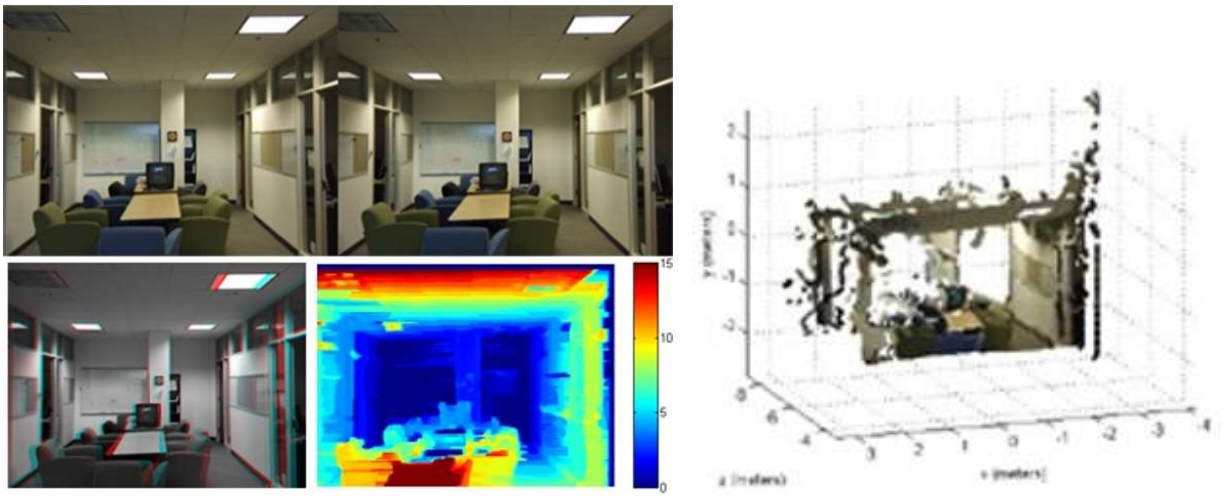
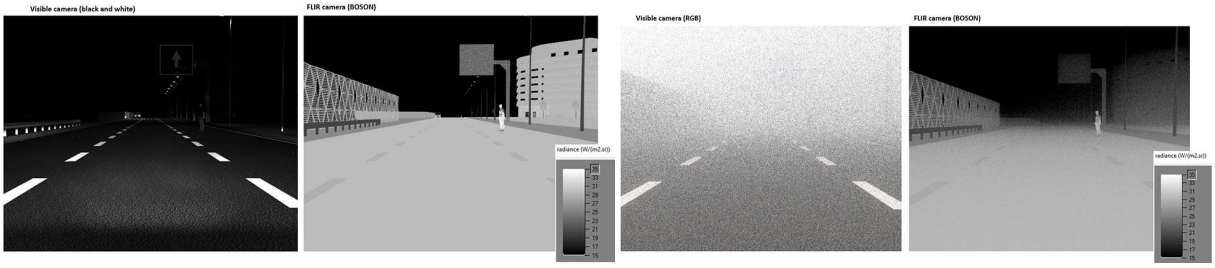


Figure 2 A pair of stereo images (top left) is used to generate a composite image (bottom left). A disparity map of the scene (bottom middle), and a 3D estimated rendering of the scene (right) are shown [4].

c) Infrared Camera

Since all objects that are above zero Kelvin emit light in the infrared part of the spectrum, infrared (IR) or thermal cameras are well suited for driving in dimly lit environments and for seeing through glare and fog. Furthermore since an infrared camera detects heat, it is can better detect people and animals under poorly lit conditions (Figure 3) [5]. The image displayed by the simulated IR camera can be textured using artistic emissivity textures that emulate real IR images [6]. However, in order to correctly simulate the output of an IR camera, the temperatures and emissivities of the different elements of the Virtual Environment (VE) need to be taken into account. A monochromatic (for example grey or green) scale texture can then be used to visualize the IR camera's output. VIRsuite [7] is an Unreal engine plugin that assigns spectral material properties to object models. The Unreal scene can then be rendered in the visible and/or IR band. VIRsuite can also be used to predict the steady-state surface temperatures of the scene objects or their real-time transient temperatures. This is achieved using the specified spectral material properties, as well as radiation and reflections from the various objects and surfaces in virtual scene.



(a) (b)

Figure 3 (a) Comparison of the outputs of a black and white camera under poorly lit conditions to the output of an infrared camera. (b) Comparison of the outputs of a black and white camera in dense fog conditions to the output of an infrared camera [5].

d) Fisheye Lens

Fisheye lenses has been used to capture ultra wide-angle images. However, it introduces significant visual distortions in order to produce a wide panoramic view. The fisheye lens projection follows an approximately linear relation between the incidence angle of an object point's beam and the distance from the corresponding image point to the principle point. Schwalbe [8] developed a projection model that was aimed at improving the distortion effects. A calibration of fisheye lens camera systems was performed where the reference object coordinates of the targets were determined with high precision. The calibration itself is done by spatial resection from a single image of the calibration room. The calibration parameters include interior orientation, exterior orientation and lens distortion. They offered a geometric model for fisheye lenses.

e) Omni-Directional 360° Lens

360-degree panoramic view a scene can be achieved by Dioptric cameras (e.g. fisheye lens), Catadioptric cameras, or Polydioptric cameras. These cameras optimally combine mirrors and conventional cameras, or employing purely dioptric fish-eye lenses. Omnidirectional cameras can be classified into two, central and non-central. Central catadioptric systems can be built by combining an orthographic camera with a parabolic mirror, or a perspective camera with either a hyperbolic or elliptical mirror. Panoramic cameras using fish-eye lenses cannot in general be considered as central systems, but the single viewpoint property holds approximately true for some camera models. Toepfer and Ehlgen [9] developed an omnidirectional camera model that can deal with parabolic and hyperbolic mirrors in combination with distorting lenses. It allows both, precision and robustness of the calibration process, and was successfully used to model a driver assistance system for light and heavy trucks.

1.1.2 LIDAR

LIDAR (light detection and ranging OR laser imaging, detection, and ranging) is a method used for measuring distances by illuminating the object with laser light and measuring the reflection with a sensor. Differences in laser return times are used to make digital 3-D representations of the environment knowing the speed of light with which the laser pulses propagate. The simplest LIDAR model to implement uses ray traces to simulate in a Virtual Environment the light pulses that are sent out by the LIDAR. The ray traces are limited to a maximum distance. A hit result is returned if an object intersects with the line trace before it reaches the maximum distance. The result contains the spherical coordinate of the hit point in terms of a distance and two angles.

More accurate LIDAR models account for beam divergence effects by using several ray traces to simulate each Lidar pulse [10]. The necessary number of ray traces is calculated as:

$$N = \frac{2\left[r + R_{max} tan\left(\frac{\theta}{2}\right)\right]}{x}$$

where x is the scale of the target's spatial variation, r is the receiver's aperture radius, R_{max} is the maximum range of the LIDAR, and θ is the beam's divergence. The return intensities of the N ray traces are summed to produce the return pulse.

1.1.3 Radar

A radar sensor works by transmitting electromagnetic radio frequency (RF) wave pulses while scanning a given region of the space surrounding the sensor. The sensor then detects the reflected RF energy or echo from any object in the path of the pulse and uses the difference in timing between the transmission and detection of the signal to determine the range to the detected object. The frequency or Doppler shift of the reflected wave can be used to determine the relative speed between the transmitter and the object.

The simplest Radar models use ray tracing similar to LIDAR to detect objects in the Virtual Environment. The clustering of points within a given volume is used to detect extended objects. If a threshold number of points $N_{\varepsilon,min}$ are found within a distance ε from a given point then this is considered a core point. Points that are within the ε neighborhood of a core point without being core points themselves are called border points. The following algorithm that is used by Mathworks Radar Toolbox [11] can be used to detect extended objects from ray tracing generated point clusters:

1. Identify all core points.
2. Core points that are within each other's ε neighborhood along with their associated border points are all considered density-connected points and part of the same point cluster. A maximal set of density connected points defines a cluster. Figure 4 shows an example of a 2D radar point cluster with an arbitrary shape.
3. Points than are neither core points nor border points are noise points and are ignored.

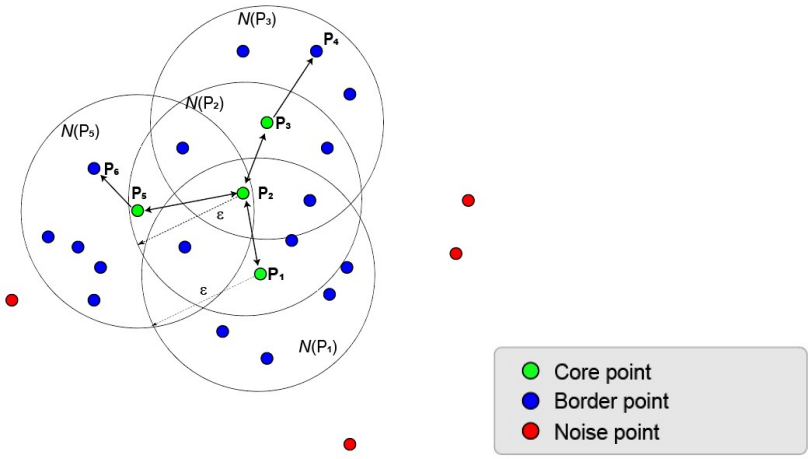


Figure 4 A Radar point cluster of arbitrary shape [11].

1.1.4 IMU

The IMU sensor is simulated by calculating the linear and angular accelerations of a given point on the vehicle.

1.1.5 GPS

The simulated position of the sensor takes place in the cartesian co-ordinate system and needs to be converted to GPS co-ordinate system. This is achieved by taking a reference GPS location which shows where the origin of the simulation is, these reference points are mapped on earth with the positive X-direction corresponding to East and Y-direction corresponding to North and Z-direction corresponding to altitude of the sensor. The measurements can be converted to the current global position reading in geodetic coordinates in terms of degrees latitude, degrees longitude, and meters of altitude. Additional noise models can also be added to the sensor model for realistic simulation.

1.1.6 Ultrasonic Sensor

The ultrasonic sensor works to detect obstacles that are very close to the vehicle in a given direction. It can be simulated in a similar way to the Radar sensor.

1.1.7 Vehicle Suspension and drivetrain state

The suspension travel and the state of the vehicle's drivetrain can be obtained directly from the vehicle's multibody dynamics simulation.

1.1.8 Magnetometer

In a GPS denied environment, a magnetometer could be the best available way for an autonomous vehicle to determine its heading. The magnetometer gives the direction of Earth's magnetic North relative to the vehicle.

1.2 Overview of Communication Standards

1.2.1 ROS

The Robot Operating System (ROS) [12] is an open-source framework that helps researchers and developers build and reuse code between robotics applications. ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS is used across numerous industries from autonomous vehicles, agriculture to medical devices to vacuum cleaners but is spreading to include different automation and software-defined dynamic use-cases.

1.2.2 ROS-M

ROS-Military or ROS-M [13] is tailored to the unique needs of military robots. ROS-M program aims to create a central registry of defense-related robotic components surrounded by a community of practice with common processes, systems, and standards. These software components serve as a foundation for future autonomy efforts to

build from, both fulfilling the potential of broad collaboration and greatly increasing the pace of development of autonomous platforms

1.2.3 JAUS

The Joint Architecture for Unmanned Systems (JAUS) [14] is a standard that was released by the Society of Automotive Engineers (SAE) for sensing, control, and computational communication of components for unmanned systems. JAUS defines a set of reusable components and their interfaces. In order to ensure that the JAUS architecture will be applicable to the entire domain of unmanned mobile systems, the following four characteristics were considered throughout its development: vehicle platform independence, mission isolation, hardware independence, and technology independence.

1.2.4 DSRC

Vehicles equipped with vehicle-to-vehicle (V2V) technology are able to identify the speed, location and direction of other vehicles within a proximity of about 300 meters [15]. This can help avoid accidents through improved situational awareness at blind spots and intersections, in heavy traffic and in other scenarios where drivers might be unaware of impending danger because of obstructions, weather or terrain. V2V technology enables vehicles to receive omni-directional messages, which will help create a 360-degree awareness of other vehicles in the proximity. Vehicles equipped with appropriate software can use the messages from surrounding vehicles to determine potential crash threats as they develop. The technology can then employ visual, tactile, and audible alerts or, a combination of these alerts to warn drivers.

V2X is a communication system that allows vehicles to communicate with other vehicles and infrastructure around them. V2X refers to the Vehicle to Everything Communication. V2X systems are intended to improve the safety, comfort and convenience of vehicles by allowing them to communicate with virtually anything that can be instrumented. This increased situational awareness enables more intelligent and connected solutions. In a V2X scenario, vehicles can receive wireless transmissions from other moving vehicles, from traffic signals, from weather reporting networks, and even from bicyclists and pedestrians to better understand and interact with the world around them.

Dedicated Short Range Communication (DSRC) is a system in which information is shared between vehicles (V2V) and between vehicles and infrastructure (V2X) [16]. In general, this technology is intended to aid the flow of anonymized information on driving conditions. Every vehicle broadcasts its location, heading and speed 10 times per second in a secure and anonymous manner. All surrounding vehicles receive the message, and each estimates the risk imposed by the transmitting vehicle. This allows a perception, detection and assessment of dangerous situations (road obstacles and potential collisions with road users) before they can be noticed visually. In a military off-road setting, DSRC can be used for inter-vehicle communication between members of an autonomous vehicle team.

1.3 Review of Existing Autonomous Vehicle Simulation Platforms

There are numerous platforms that can be used for simulating autonomous vehicles. A comparison of the most prominent ones is presented in this section. The platforms that the AVSP will be compared against are: rFpro [17], cognata [18], CARLA [19], Foretify [20], the Automated Driving Toolbox (ADT) developed by MathWorks [21], ANVEL [22], CHRONO [23], Siemens Simcenter Prescan [24], AirSim [25], VRXPERIENCE developed by Ansys [26], NVIDIA DRIVE Constellation [27], Gazebo [28], VANE [29], and DeepDrive [30]. The platforms are compared based on the capabilities of their latest versions. The main comparison points are:

1. Sensor integration: all autonomous vehicles use sensors to navigate their environment. Along with visible light cameras, four widely used sensors are infrared (IR) cameras, LIDAR, Radar and GPS. The complexity and fidelity with which these sensors are simulated can greatly differ among different platforms. In this comparison, we will simply focus on whether or not a given sensor is simulated in a given platform.
2. Photorealism of the Virtual Environment (VE): All autonomous vehicle simulation platforms develop a visual representation of the environment within which the vehicle operates. In this comparison we assess whether or not this visual representation closely matches a real environment. This criteria is essential for the correct simulation of camera sensors.
3. Weather simulation: weather conditions such as rain, snow, fog, and wind can affect the simulation in a number of ways. Weather conditions such as rain, snow, cloud cover and fog can affect the visual representation of the VE. Furthermore, the performance of sensors such as cameras, LIDAR and Radar can be influenced by weather

conditions. Wind can also induce forces on the autonomous vehicle. In this comparison we will focus on whether or not different weather conditions can be simulated by a given platform regardless of fidelity.

4. Dynamic time of day simulation: The real-time run time of a typical off-road autonomous vehicle scenario can be several hours. Thus it is not sufficient for the simulation platform to simply have the ability to change lighting conditions or the position of the sun at the start of the simulation. For a higher level of fidelity, the user should be able to specify the time of day when the simulation starts. The lighting conditions should then change dynamically based on the simulation time.
5. Dynamic soil properties: The properties of the soil can change during the simulation due to a number of factors. This includes rain, snow, temperature changes, and soil drying over time.
6. Simulation of 10s of vehicles: In order to be able to simulate 10s of vehicle within a single off-road scenario, the platform must be able to distribute the simulation load over several interconnected computational nodes.
7. Detailed vehicle dynamics: This comparison criteria concerns whether or not the vehicle's suspension, steering, and drivetrain are simulated using detailed models and physics.
8. Inter-vehicle communication: This comparison criteria concerns whether or not the vehicles in the simulation can communicate their positions, speeds, and sensor outputs to each other.
9. Simulate tracked vehicles: This includes vehicles with segmented and continuous tracks.
10. Simulate legged vehicles: This includes bipedal, quadrupedal robots.
11. Real-time simple terramechanics: Simple terramechanics capability using a semi-empirical model such as the Bekker-Wong model [Wong, 2010].
12. Terrain rutting: Vehicles passing over the soil can also dynamically alter the soil properties (through compaction and/or tilling) and terrain topography (through rutting).
13. Simulate interaction with vegetation: Vegetation includes grass, crops, shrubs, and trees. Small vegetation such as grass, crops, and small shrubs can increase resistance to the vehicle's motion. Small shrubs and trees can be toppled by the vehicle.
14. Simulate interaction with structures: Poles and fences can be toppled by the vehicle depending on their size.
15. Complex terramechanics with a Moving Terrain Patch (MTP): Complex terramechanics simulates the soil using the Discrete Element Method (DEM). The MTP approach allows the simulation of arbitrarily long terrain traverses with a limited number of DEM particle. This type of simulation can be performed as a postprocessing step or if the simulation is not run in real-time.
16. Scenario editor: The scenario editor allows the user to easily setup the scenario parameters.
17. Hardware in the Loop (HiL) connectivity: Allows the platform to connect to system hardware components in order to test how the AI interacts with them.
18. Open source platform: Is the platform open source or commercial.

Table 1 Comparison of autonomous vehicle simulation platforms. The Table shows the expected capabilities of the AVSP at the end of Phase II.

#	Feature	AVSP	rFpro	cognata	CARLA	Foretify	ADT	ANVEL	Chrono	Prescan	AirSim	ANSYS	Constellation	Gazebo	VANE	DeepDrive
1.1	LIDAR sensor simulation	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
1.2	Radar sensor simulation	Yes	No	Yes	No	No	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes
1.3	IR camera sensor simulation	Yes	No	No	No	No	Yes	No	No	Yes	No	No	Yes	Yes	No	No
1.4	GPS sensor simulation	Yes	No	No	Yes	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes	No	Yes
2	Photorealistic VE	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	No	Yes	No	No	Yes
3	Weather simulation	Yes	Yes	No	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes
4	Dynamic time of day simulation	Yes	Yes	No	No	No	No	No	No	No	No	No	Yes	No	No	Yes
5	Dynamic soil properties	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
6	Simulation for 10s of vehicles	Yes	Yes	Yes	No	No	No	No	Yes	No	No	No	No	No	No	No
7	Detailed vehicle dynamics	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No	No	No	Yes	No	No
8	Inter-vehicle communication	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No
9	Simulate tracked vehicles	Yes	No	No	No	No	No	Yes	Yes	No	No	No	No	Yes	No	No
10	Simulate Legged vehicles	Yes	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No
11	Real time simple terramechanics	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No
12	Terrain rutting	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No
13	Simulate interaction with vegetation	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
14	Simulate interaction with structures	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
15	Complex terramechanics with MTP	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
16	Scenario editor	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes	No	No	Yes	No	No	Yes
17	HiL connectivity	Yes	Yes	No	No	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No	No	Yes
18	Open source platform	Yes	No	No	Yes	No	No	No	Yes	No	No	No	No	Yes	No	Yes

Task 2.1: Develop AVSP Prototype

2.1.1 Develop IVRESS - Unreal Co-Simulation API

A general purpose client - server networking co-simulation API was developed for integrating the main two components of the AVSP, namely, ASA's IVRESS multibody dynamics / finite element (MBD/FEM) solver and the Unreal Engine. This API can be used with any visualization engine (such as Unreal or Unity) and any MBD (or in general computational mechanics) software. This visualization API called 'IVRESScoSimAPI' can be used for:

- Sending the positions and rotations of rigid bodies from the MBD software to the visualization engine.
- Sending a mesh that represents as rigid or deformable body/surface from the computational mechanics software to the visualization engine.

The server part of the API takes as input at the start of the simulation the network port number, the number of rigid bodies, rigid body names in the visualization engine, number of surfaces, and surface names in the visualization engine. Each time step (frame) during the simulation, the server sends to the client the new positions and rotations of the rigid bodies and the new point positions and/or connectivity of the surfaces (if they change). The server functions along with their input and output variables and function descriptions are shown below:

```
//Initialize the Visualization server
//Returns a reference to the server number or 0 if an error occurred.
//numRigidBoies: Number of rigid bodies to send to the client
//rigidBodyNames: Pointer to an array of strings containing the rigid body names in the visualization engine (Unreal or Unity)
//numSurfaces: Number of surfaces
//surfaceNames: Pointer to an array of strings containing the surface names in the visualization engine (Unreal or Unity)
int IVRESSvisServerInitialize(int port, int numRigidBoies, char **rigidBodyNames, int numSurfaces, char **surfaceNames);
```

```
//Attempts to start the visualization server
//Return 0 if the visualization server starts with no errors.
int IVRESSvisServerStart(int serverNum);
```

```
//Returns 1 if the Visualization server has started, and 0 otherwise.
char IVRESSvisServerStarted(int serverNum);
```

```
//Send positions and rotations of the rigid bodies to the visualization client
//Returns 0 if no error
//buffer: Buffer containing the positions and rotations of the rigid bodies and/or the surfaces point positions and connectivities in the following format:
//    action# data action# data ...
//IF:
//action# = IVvisClientSendRBpos
//     px1 py1 pz1 px2 py2 pz2 .... pxx pyx pzx  qx1 qy1 qz1 qw1 qx2 qy2 qz2 qw2  ..... qxn qyn qzn qwn
//action# = IVvisClientSendSurfacePos
//     surface# numPoints px1 py1 pz1 px2 py2 pz2 ..... pxn pyn pzn
//     n = numPoints
//action# = IVvisClientSendSurfaceConn
//     surface# numTriangles c11 c12 c13 c21 c22 c23 ..... cn1 cn2 cn3
//     n = numTriangles
//
//bufferLen: buffer length
int IVRESSvisServerSendData(int serverNum, char *buffer,int bufferLen);
```

The client part of the API takes as input at the start of the simulation the network port number and the IP address of the server. The Client part of the API is integrated into a custom Unreal C++ Actor class which is called every frame (Tick). Each frame/tick during the simulation, the client receives from the server the new positions and rotations of the rigid bodies and the new point positions and/or connectivity of the surfaces The "Procedural Mesh" capability of Unreal is used to generate the polygonal surfaces in Unreal at run-time. The client functions along with their input and output variables and function description are:

```
//Initialize the Visualization client
//port: desired network port number
//ipAddress: IP address ###.###.###.### of the server
//Returns a reference to the Client number or 0 if an error occurred.
int IVRESSvisClientInitialize(int port, char *ipAddress);
```

```
//Receive the positions and rotations of the rigid bodies and the new surfaces point positions and/or connectivities from the server
//Returns the 0 if no error
int IVRESSvisClientRcvData(int clientNum);
```

```
//Returns the number of rigid bodies
//clientNum: Client number
int IVRESSvisClientGetNumberOfRigidBodies(int clientNum);
```

```
//Returns a pointer to string containing the name of rigid body 'bodyNumber' in the visualization engine
//clientNum: Client number
//bodyNumber: Rigid body number: 1, 2, 3, 4 ....
char *IVRESSvisClientGetRigidBodyName(int clientNum, int bodyNumber);
```

```
//Returns the number of surfaces
//clientNum: Client number
int IVRESSvisClientGetNumberOfSurfaces(int clientNum);
```

```
//Returns a pointer to string containing the name of surface 'surfNumber' in the visualization engine
//clientNum: Client number
//bodyNumber: Rigid body number: 1, 2, 3, 4 ....
char *IVRESSvisClientGetSurfacesName(int clientNum, int surfNumber);
```

```
//Returns a pointer to the current position of rigid body 'bodyNumber'. Position: px, py, pz
//clientNum: Client number
//bodyNumber: Rigid body number: 1, 2, 3, 4 ....
double *IVRESSvisClientGetRigidBodyPosition(int clientNum, int bodyNumber);
```

```
//Returns a pointer to the current rotation quaternion of rigid body 'bodyNumber'. Position: qx, qy, qz, qw
//clientNum: Client number
//bodyNumber: Rigid body number: 1, 2, 3, 4 ....
double *IVRESSvisClientGetRigidBodyRotation(int clientNum, int bodyNumber);
```

```
//Returns surface points positions:
//clientNum: Client number
//surfaceNumber: 1, 2, 3, 4 ....
//points: px1 py1 pz1 px2 py2 pz2 .... pxn pyn pzn
//numPoints: total number of points
int IVRESSvisClientGetSurfacePoints(int clientNum, int surfaceNumber, float *points, int *numPoints);
```

```
//Returns surface triangle connectivities:
//clientNum: Client number
//surfaceNumber: 1, 2, 3, 4 ....
//Connectivity: c11 c12 c13 c21 c22 c23 ..... cn1 cn2 cn3
//numPoints: total number of triangles
int IVRESSvisClientGetSurfaceConn(int clientNum, int surfaceNumber, int *triangles, int *numTriangles);
```

In addition, we developed a spreadsheet user interface for users to enter the data needed to interface with the visualization code in the DIS/GroundVehicle software. Wheeled, tracked and legged ground vehicles can be simulated using the DIS/GroundVehicle software. The spreadsheet has the data needed by the server (MBD code). This data is (Figure 5 and Figure 6):

- Network communication port number
- Synchronization time step used to send data to the visualization engine (Unreal).
- Type of rotation measure expected by the visualization engine (Quaternion, Rotation vector, or Rotation matrix).
- Global translation of the MBD model relative to the visualization engine model.
- Global scale of the MBD model relative to the visualization engine model.
- List of rigid body names in the MBD code (DIS) and the corresponding names in the visualization engine (Unreal).
- Relative translation and rotation vector of the geometry of each rigid body between the MBD code and the visualization engine. Note that we don't have to send all the rigid bodies in the MBD model to the visualization engine. We can just send the bodies required to visualize the vehicle. For example the internal driveline components hidden by the vehicle body don't need to be sent.

VisualizationMBD1	Comment:	DEM CoSimulation Object in DIS			
Parameter Description	**Symbol**	**Value1**	**Value2**	**Value3**	**Units**
Enabled	*enabled*	Yes			
Network communication port	*port*	4985			
Synchronization time step	*syncStep*	0.03			
Rotation measure	*rotationMeasure*	Quaternion			
Global translation	*translation*	38.1	-18.99	-0.1	m
Global scale	*scale*	1	-1	1	

Display Bodies

#	IVRESS Rigid body name	Visualization Engine Rigid Body Name	Body center (x y z)	Body rotation (x y z w)
1	VehicleFrame1	BP-FEDA-body	1.6 0 -0.44	
2	Knuckle1FrontR	BP-FEDA-FrontKnuckleR		
3	UpContArm1FrontR	BP-FEDA-FrontUCAR	0 -0.240745 0	0 0 1 0
4	LowContArm1FrontR	BP-FEDA-FrontLCAR	0 -0.322041 0	0 0 1 0
5	Knuckle1FrontL	BP-FEDA-FrontKnuckleL		
6	UpContArm1FrontL	BP-FEDA-FrontUCAL	0 -0.240745 0	0 0 1 0
7	LowContArm1FrontL	BP-FEDA-FrontLCAL	0 -0.322041 0	0 0 1 0
8	Knuckle1Back1R	BP-FEDA-BackKnuckleR		
9	UpContArm1Back1R	BP-FEDA-BackUCAR	0 -0.240745 0	0 0 1 0
10	LowContArm1Back1R	BP-FEDA-BackLCAR	0 -0.322041 0	0 0 1 0
11	Knuckle1Back1L	BP-FEDA-BackKnuckleL		
12	UpContArm1Back1L	BP-FEDA-BackUCAL	0 -0.240745 0	0 0 1 0
13	LowContArm1Back1L	BP-FEDA-BackLCAL	0 -0.322041 0	0 0 1 0
14	Wheel1FrontR	BP-FEDA-wheelFR		
15	Wheel1FrontL	BP-FEDA-wheelFL		
16	Wheel1Back1R	BP-FEDA-wheelBR		
17	Wheel1Back1L	BP-FEDA-wheelBL		

Figure 5 DIS/GroundVehicle spreadsheet user interface showing the FED-Alpha vehicle bodies along with their IVRESS name, Unreal name, translation and rotation. The FED-Alpha IVRESS MBD vehicle model consists of 52 rigid bodies. Of those bodies 17 rigid bodies (vehicle body, wheels, knuckles, and suspension control arms) are sent to Unreal.

VisualizationMBD1	Comment:	DEM CoSimulation Object in DIS			
Parameter Description	**Symbol**	**Value1**	**Value2**	**Value3**	**Units**
Enabled	*enabled*	Yes			
Network communication port	*port*	4985			
Synchronization time step	*syncStep*	0.005			
Rotation measure	*rotationMeasure*	Quaternion			
Global translation	*translation*	38.1	-18.99	-0.1	m
Global scale	*scale*	1	-1	1	

Display Bodies

#	IVRESS Rigid body name	Visualization Engine Rigid Body Name	Body center (x y z)	Body rotation (x y z w)
1	VehicleFrame1	BP-M113-body	1.85 0 -0.4	
2	Arm_Wheel1R	BP-M113-armR1		0 1 0 1.570796327
3	Arm_Wheel2R	BP-M113-armR2		0 1 0 1.570796327
4	Arm_Wheel3R	BP-M113-armR3		0 1 0 1.570796327
5	Arm_Wheel4R	BP-M113-armR4		0 1 0 1.570796327
6	Arm_Wheel5R	BP-M113-armR5		0 1 0 1.570796327
7	Arm_Wheel1L	BP-M113-armL1		0 1 0 1.570796327
8	Arm_Wheel2L	BP-M113-armL2		0 1 0 1.570796327
9	Arm_Wheel3L	BP-M113-armL3		0 1 0 1.570796327
10	Arm_Wheel4L	BP-M113-armL4		0 1 0 1.570796327
11	Arm_Wheel5L	BP-M113-armL5		0 1 0 1.570796327
12	SprocketR	BP-M113-sprocketR	0 0.15 0	0 1 0 1.570796327
13	Wheel1R	BP-M113-wheelR1		0 1 0 1.570796327
14	Wheel2R	BP-M113-wheelR2		0 1 0 1.570796327
15	Wheel3R	BP-M113-wheelR3		0 1 0 1.570796327
16	Wheel4R	BP-M113-wheelR4		0 1 0 1.570796327
17	Wheel5R	BP-M113-wheelR5		0 1 0 1.570796327
18	IdlerR	BP-M113-idlerR	0 0.15 0	0 1 0 1.570796327
19	SprocketL	BP-M113-sprocketL	0 0.15 0	0 1 0 1.570796327
20	Wheel1L	BP-M113-wheelL1		0 1 0 1.570796327
21	Wheel2L	BP-M113-wheelL2		0 1 0 1.570796327
22	Wheel3L	BP-M113-wheelL3		0 1 0 1.570796327
23	Wheel4L	BP-M113-wheelL4		0 1 0 1.570796327
24	Wheel5L	BP-M113-wheelL5		0 1 0 1.570796327
25	IdlerL	BP-M113-idlerL	0 0.15 0	0 1 0 1.570796327
1 26	Rseg1_1	BP-M113-trackR01		0.577348 0.577353 0.577349 2.09439
2 27	Rseg1_2	BP-M113-trackR02		0.577348 0.577353 0.577349 2.09440

Figure 6 M113 DIS/GroundVehicle spreadsheet user interface for the M113 vehicle. The M113 IVRESS MBD vehicle model consists of 154 rigid bodies. Of those bodies 151 rigid bodies (vehicle body, sprockets, wheels, idlers, and track segments) are sent to Unreal.

2.1.2 Integrate Multibody Dynamics Vehicle Models

Six types of vehicles are modeled and demonstrated with IVRESS/DIS – Unreal co-simulation API:

1) **FED-Alpha**: Light-weight multi-purpose personnel carrier wheeled vehicle.
2) **Polaris MRZR**: Small multi-purpose personnel carrier wheeled vehicle.
3) **TracerX**: Small wheeled robotic vehicle.
4) **M113**: Segmented-track personnel carrier vehicle.
5) **Milrem Themis**: Small belt-track robotic vehicle.
6) Legged robots:
    a. **STEPPR2**: 2-legged walking robot.
    b. **WANDERER**: 2-legged walking robot.

The MBD IVRESS models of the FED-Alpha, Polaris MRZR, M113, TracerX and Milrem Themis which were created using the DIS/GroundVehicle user interface are shown in Figure 7, Figure 8, Figure 9, Figure 10, and Figure 11. As an example of a vehicle model in DIS/GroundVehicle, Figure 12 shows the block diagram of the M113 MBD model. Typical DIS/GroundVehicle input spreadsheets for the various M113 sub-systems including wheels, segmented track, and suspension system are shown in Figure 13, Figure 14, and Figure 15 respectively.

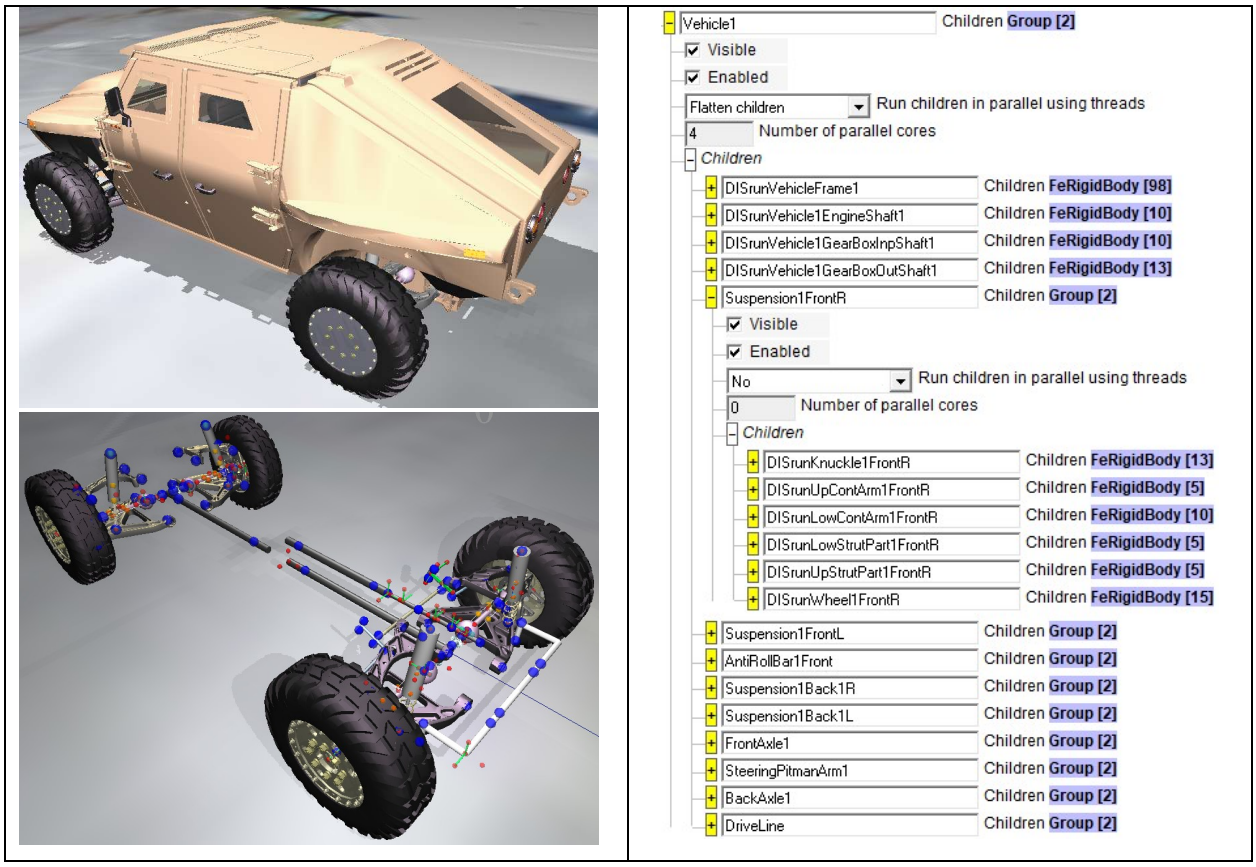


Figure 7 Multibody dynamics model of the FED-Alpha displayed in IVRESS (left) and corresponding IVRESS model tree.

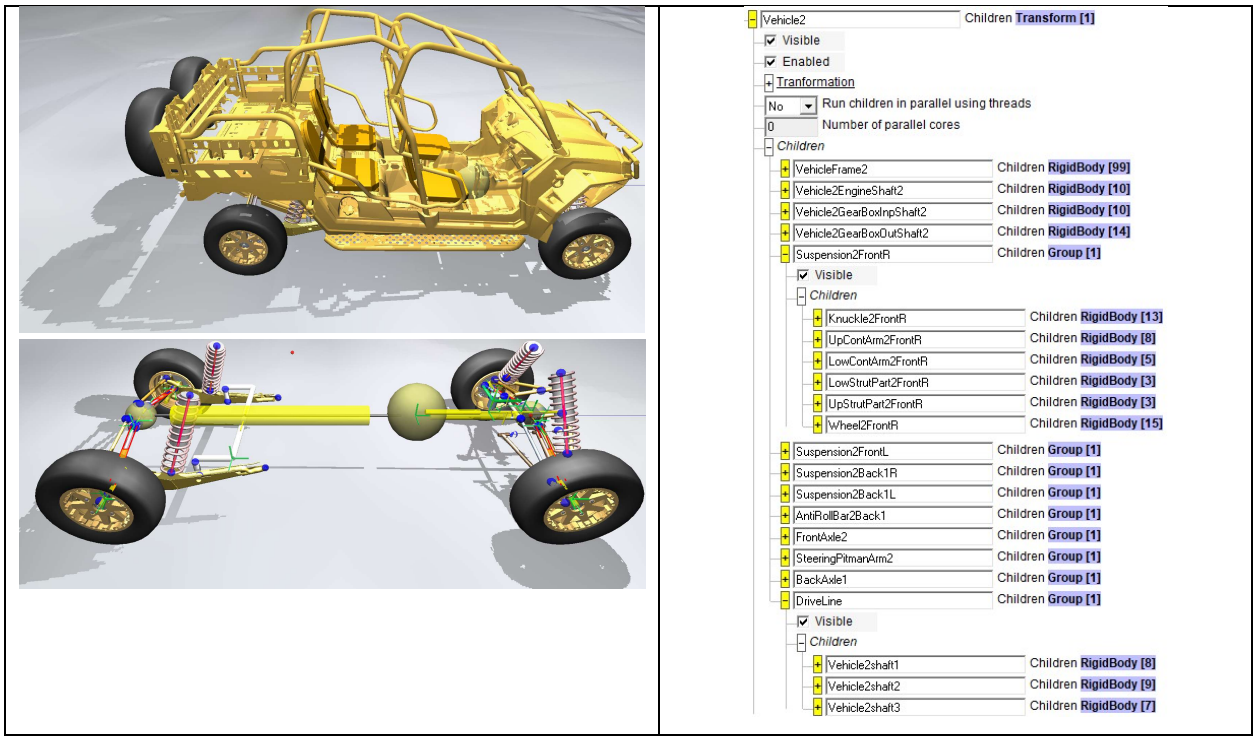


Figure 8 Multibody dynamics model of the Polaris MRZR displayed in IVRESS (left) and corresponding IVRESS model tree.

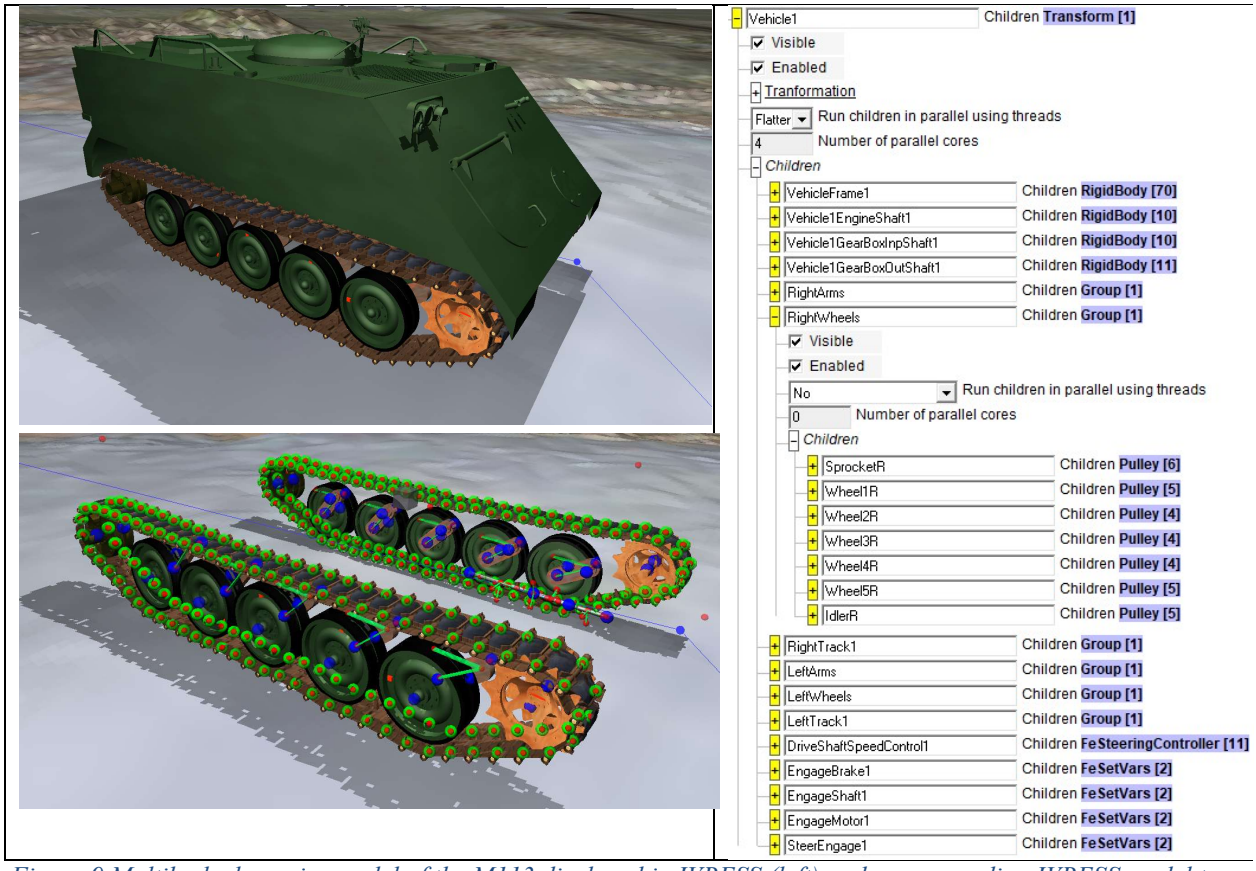


Figure 9 Multibody dynamics model of the M113 displayed in IVRESS (left) and corresponding IVRESS model tree.

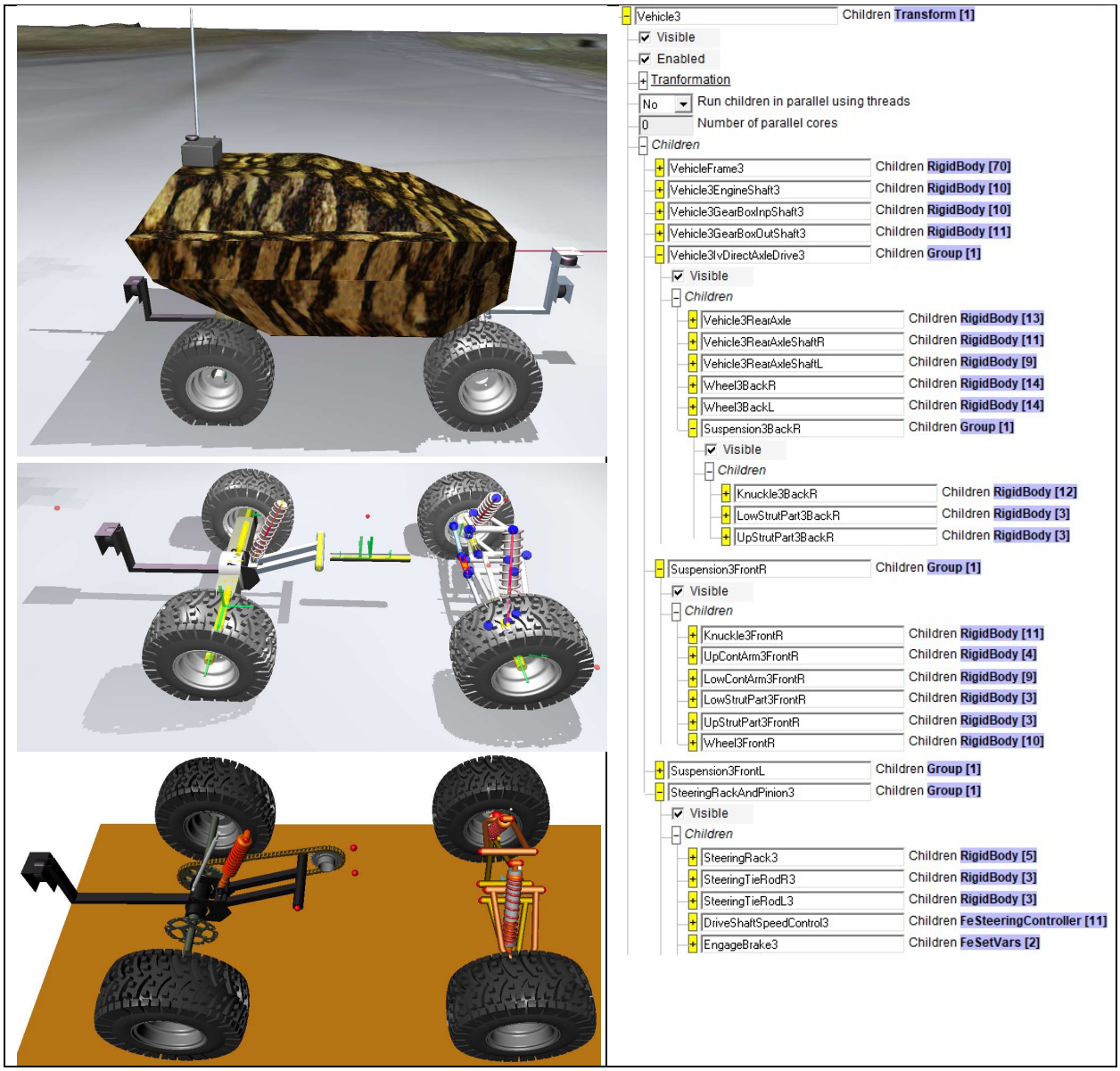


Figure 10 Multibody dynamics model of the TracerX robotic vehicle displayed in IVRESS and corresponding IVRESS model tree.

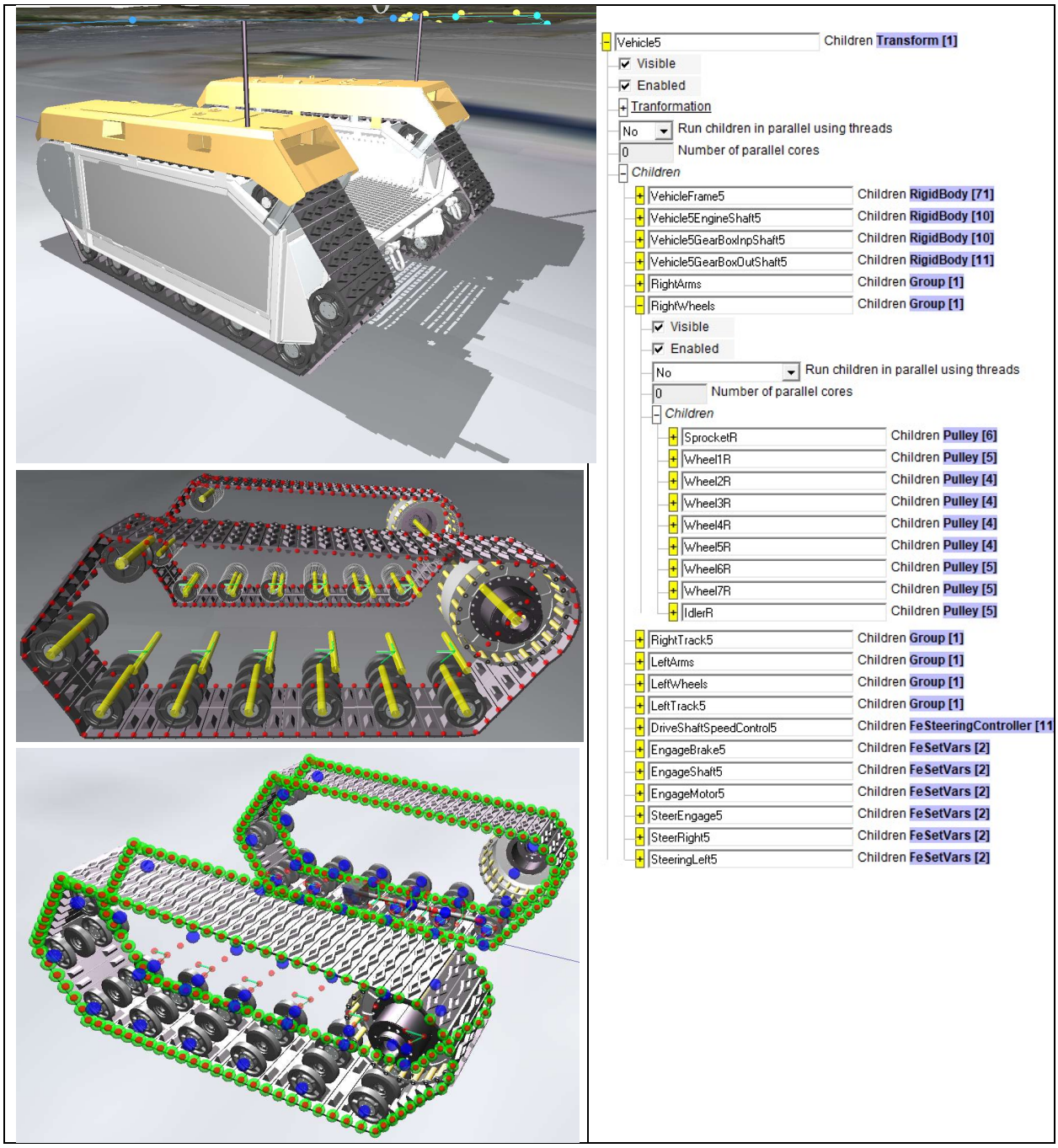


Figure 11 Multibody dynamics model of the Milrem Themis robotic belt-track vehicle displayed in IVRESS (left) and corresponding IVRESS model tree.

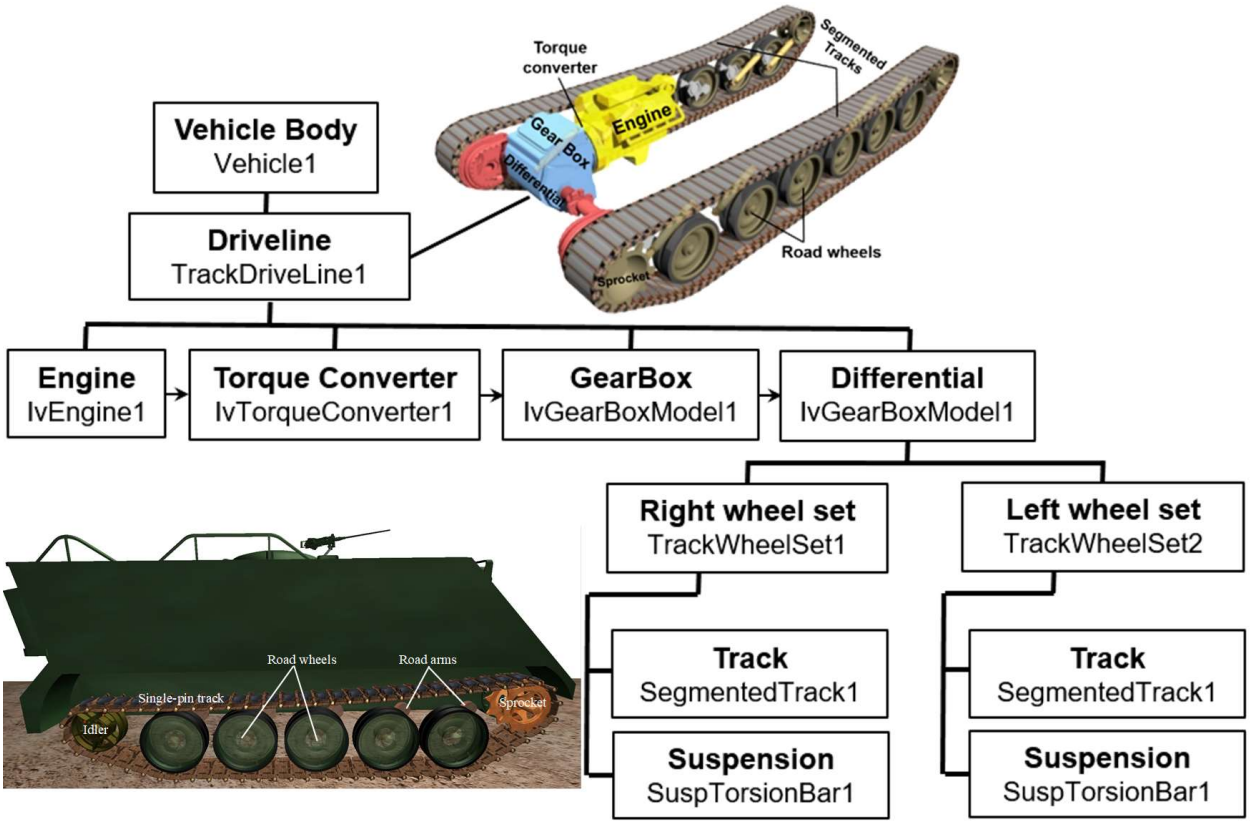


Figure 12 Block diagram showing the configuration of the M113 MBD model.

		Wheel table						
Parameter description	**Symbol**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Wheel name	*wheelName*	Sprocket	Wheel1	Wheel2	Wheel3	Wheel4	Wheel5	Idler
Use in track wrapping calculations	*useWrap*	Path 2	Path 1	No	No	No	Path 1	Path 1
Road arm	*torsionBar*	None	SuspTorsionBar1	SuspTorsionBar1	SuspTorsionBar1	SuspTorsionBar1	SuspTorsionBar1	None
Torsion bar or wheel hinge point X (m)	*torsionBarX*	0	-0.3969	-1.0636	-1.7304	-2.3971	-3.0639	-3.9849
Torsion bar or wheel hinge point Y (m)	*torsionBarY*	0	0	0	0	0	0	0
Torsion bar or wheel hinge point Z (m)	*torsionBarZ*	0	-0.0873	-0.0873	-0.0873	-0.0873	-0.0873	-0.151
Torsion bar arm length (m)	*armLength*	0	0.3175	0.3175	0.3175	0.3175	0.3175	0
Torsion bar nominal angle (deg.)	*armAngle0*	0	43	43	43	43	43	0
Initial rotation angle (deg.)	*wheelAng0*	9	0	0	0	0	0	0
Nominal radius of pulley geometry (m)	*wheelR*	0.22	0.305	0.305	0.305	0.305	0.305	0.219
Pitch radius of pulley (m)	*wheelRp*	0.224	0.335	0.335	0.335	0.335	0.335	0.24
Pulley width (m)	*wheelW*	0.22	0.25	0.25	0.25	0.25	0.25	0.25
Pulley Mass (m)	*wheelMass*	100	80	80	80	80	80	70
Pulley Rotational mom. of inertia (kg.m^2)	*wheelJ*	3.24	4.232	4.232	4.232	4.232	4.232	2.268
Pulley Ixx (kg.m^2)	*wheelInertia*	3.24	4.232	4.232	4.232	4.232	4.232	2.268
Driver sprocket	*sprocket*	Yes	No	No	No	No	No	No
Geometric icon file name	*icon*	../CAD/SprocketL.wrl	../CAD/WheelL.wrl	../CAD/WheelL.wrl	../CAD/WheelL.wrl	../CAD/WheelL.wrl	../CAD/WheelL.wrl	../CAD/IdlerL.wrl
Contact surface file name	*contactSurf*	../CAD/SprocketSurface1.wrl	../CAD/WheelSurface.wrl	../CAD/WheelSurface.wrl	../CAD/WheelSurface.wrl	../CAD/WheelSurface.wrl	../CAD/WheelSurface.wrl	../CAD/IdlerSurface.wrl
Contact search tolerance (m)	*tolerance*	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Figure 13 Input table for the M113 wheels in the DIS/VehicleInterface software.

SegmentedTrack1	Comment				
Parameter Description	**Symbol**	**Value1**	**Value2**	**Value3**	**Units**
Track type	*trackType*	Single Pin			
Initial track strain	*strain*	0.028			
Track width	*widthSeg*	0.24			m
Contact tolerance	*tolerance*	0.015			m
Segment 1	_				
Track segment 1 length	*lengthSeg1*	0.1524			m
Track segment 1 joint stiffness	*stiffSeg1*	25590000			N/m
Track segment 1 joint damping	*dampSeg1*	210000			N.s/m
Track segment 1 mass	*massSeg1*	8.5714			Kg
Track segment 1 (Ixx, Iyy, Izz)	*inertiaSeg1*	0.156214	0.15621	0.0318	$Kg.m^2$
Track segment 1 center of mass (w.r.t. hinge point)	*cgSeg1*	0	0	0	m
Track segment 1 icon file name	***fileSeg1***	**../CAD/Track_link.wrl**			
Track segment 1 to wheel contact surface A file name	***contSurfA1***	**../CAD/TrackToWheelSurf.wrl**			
Contact surface normal stiffness per unit area	*contSurfNorSA1*	5000000000			N/m^3
Contact surface normal stiffness per unit area	*contSurfNorDA1*	2000000			$N.s/m^3$
Contact surface friction coefficient	*contSurfMuA1*	0.5			
Contact surface friction spring per unit area	*contSurfFricSA1*	1000000000			N/m^3
Contact surface friction damping per unit area	*contSurfFricDA1*	2000000			$N.s/m^3$
Contact surface wheel list	*contSurfListA1*	Wheel1 Wheel2 Wheel3 Wheel4 Wheel5 Idler			
Track segment 1 to wheel contact surface B file name	***contSurfB1***	**../CAD/TrackToSprocketSurf.wrl**			
Contact surface normal stiffness per unit area	*contSurfNorSB1*	500000000			N/m^3
Contact surface normal damping per unit area	*contSurfNorDB1*	100000			$N.s/m^3$
Contact surface friction coefficient	*contSurfMuB1*	0.5			
Contact surface friction spring per unit area	*contSurfFricSB1*	500000000			N/m^2
Contact surface friction damping per unit area	*contSurfFricDB1*	200000			$N.s/m^2$
Contact surface wheel list	*contSurfListB1*	Sprocket			
Track segment 1 to pavement contact surface file name	***contRoad1***	**../CAD/TrackToRoadSurf.wrl**			
Contact surface normal stiffness per unit area	*contRoadNorS1*	300000000			N/m^3
Contact surface normal stiffness per unit area	*contRoadNorD1*	200000			$N.s/m^3$
Contact surface friction coefficient	*contRoadMu1*	0.8			
Contact surface friction spring per unit area	*contRoadFricS1*	500000000			N/m^2
Contact surface friction damping per unit area	*contRoadFricD1*	200000			$N.s/m^2$
Contact surface search tolerance	*contRoadTol1*	0.1			m
Track segment 1 to soil contact surface file name	*contSurfSoil1*	../CAD/TrackToSoilSurf.wrl			

Figure 14 Input sheet for the M113 segmented track in the DIS/VehicleInterface software.

SuspTorsionBar1	Comment				
Parameter Description	**Symbol**	**Value1**	**Value2**	**Value3**	**Units**
Arm icon	*fileArm*	../CAD/Arm.wrl			
Mass/Inertia Properties	_				
Arm mass	*massArm*	40			kg
Arm (Ixx, Iyy, Izz)	*inertiaArm*	0.1	1.6	1.6	$kg.m^2$
Arm center of mass (w.r.t. hinge point)	*cgArm*	-0.15	0	0.03	m
Rotational Stiffness/Damping	_				
Rotational stiffness	*rotStiff*	9671.53			N.m/rad
Rotational damping	*rotDamp*	100			N.m/(rad/s)
Stop rotational stiffness	*stopStiff*	500000			N.m/rad
Stop rotational damping	*stopDamp*	2000			N.m/(rad/s)
Angles/Torque	_				
Torque at nominal angle	*torque0*	3000			N.m
Nominal angle	*angle0*	15			degree
Stop angle 1	*stopAng1*	-4.589999274			degree
Stop angle 2	*stopAng2*	50.00002143			degree

Figure 15 Input sheet for the M113 torsional spring suspension system in the DIS/VehicleInterface software.

Figure 16 shows the IVRESS model of the STEPPR2 2-legged walking robot. The STEPPR2 model has 24 rigid bodies, 18 revolute joints (2 for the upper body and 16 for each leg), 2 spherical joints, 16 rotational actuators (2 for the upper body and 7 for each leg), and 2 contact surfaces one for each foot.

Figure 17 shows the IVRESS model of the WANDERER 2-legged walking robot. The WANDERER model has 26 rigid bodies, 25 revolute joints (3 for the upper body, 6 for each leg, and 5 for each arm), 25 rotational actuators (3 for the upper body, 6 for each leg, and 5 for each arm), and 2 contact surfaces one for each foot.
We are currently working on simulating the walking gait for those two robots. The robot models will then be demonstrated on the AVT-341 sample terrain (Task 2.5).

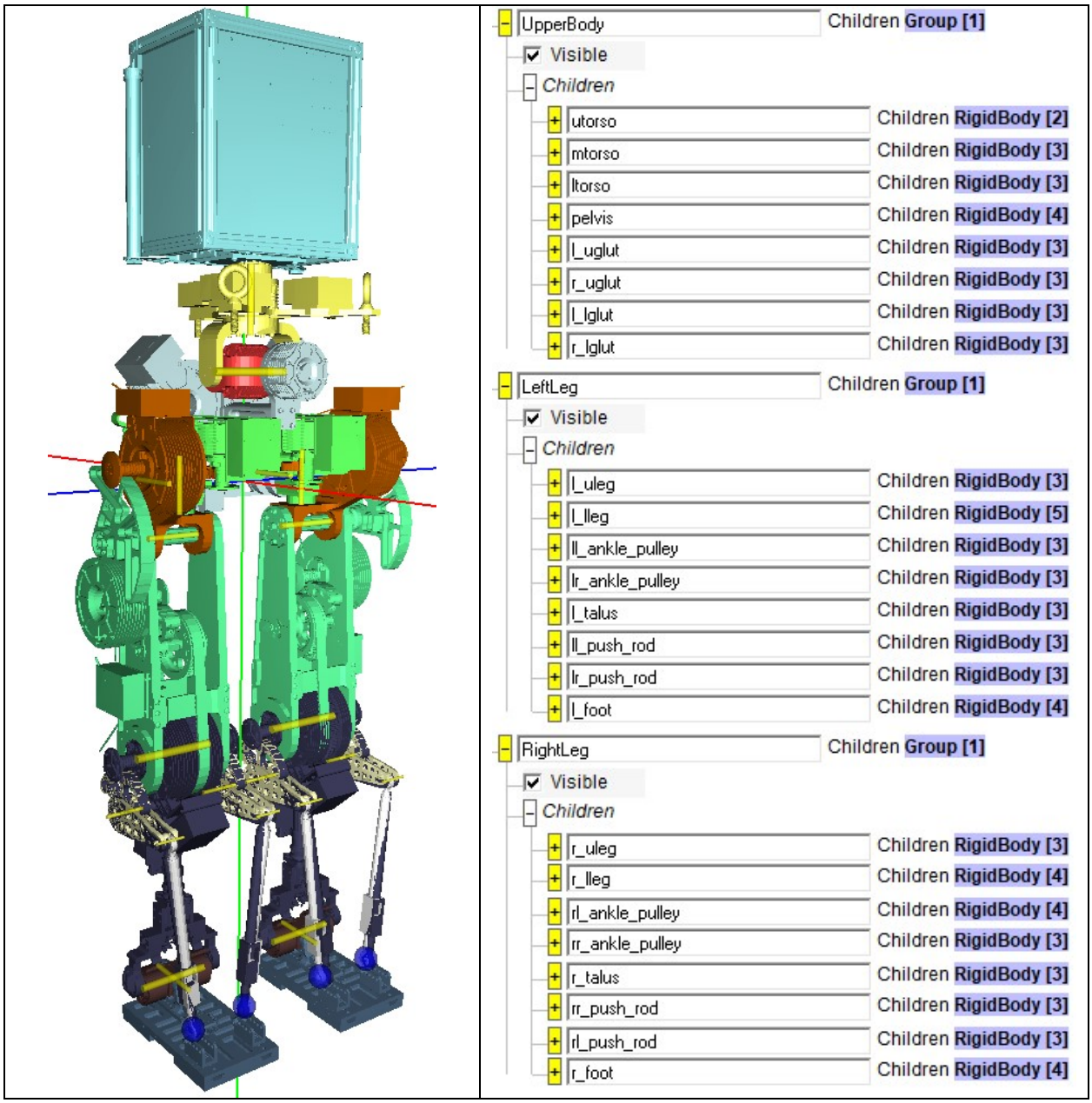


Figure 16 Multibody dynamics model of the STEPPR2 2-legged robot displayed in IVRESS (left) and corresponding IVRESS model tree.

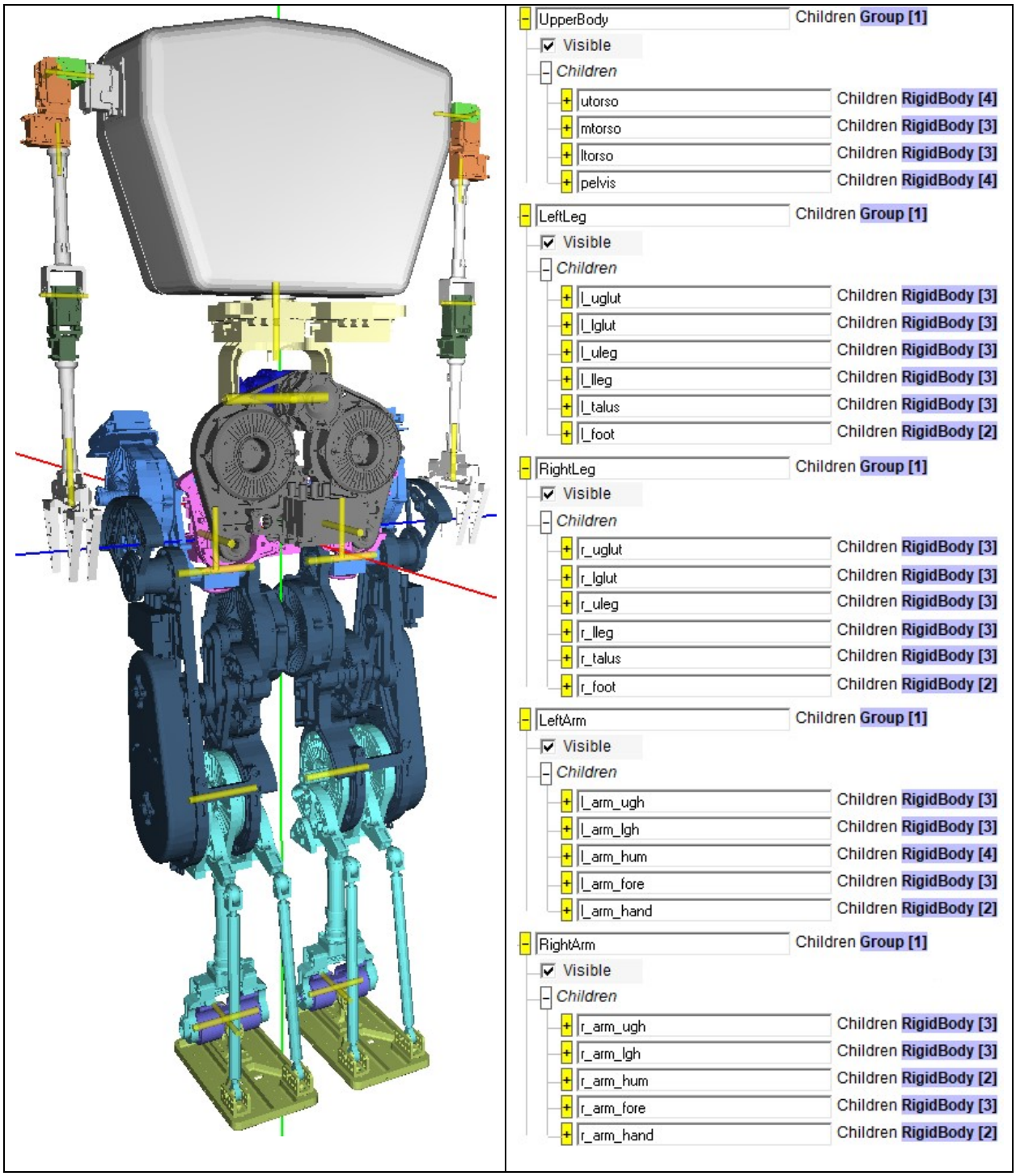


Figure 17 Multibody dynamics model of the WANDERER 2-legged robot displayed in IVRESS (left) and corresponding IVRESS model tree.

2.1.3 AVT-341 Typical Terrain Model and Simulation Scenario

The NATO AVT-341 research task group on "mobility assessment methods and tools for autonomous military ground systems," provided the GIS data of a typical off-road terrain. Multiple typical autonomous and manned vehicles simulation scenarios were also developed as part of AVT-341. The GIS terrain is the Keweenaw research center (KRC) test site at Michigan Technological University [32]. Figure 18 shows the NATO AVT-341 KRC terrain map imported into IVRESS/DIS. The size of the terrain is about 1.2 ✕ 1.9 km. The terrain is imported into IVRESS as a set of triangles, then it is converted in IVRESS to a Cartesian elevation grid with a resolution of 0.1 m (Figure 19). The resolution of the Cartesian grid is 11600 ✕ 18800. The FED-Alpha vehicle is simulated on a desired path defined using way points and a corresponding speed at each way point (Figure 18). The path and speed correspond to a demonstration scenario defined by NATO AVT-341. Figure 20 and Figure 21 show snapshots of the FED-Alpha and M113 MBD vehicle simulations in IVRESS/DIS on the AVT-341 scenario path.

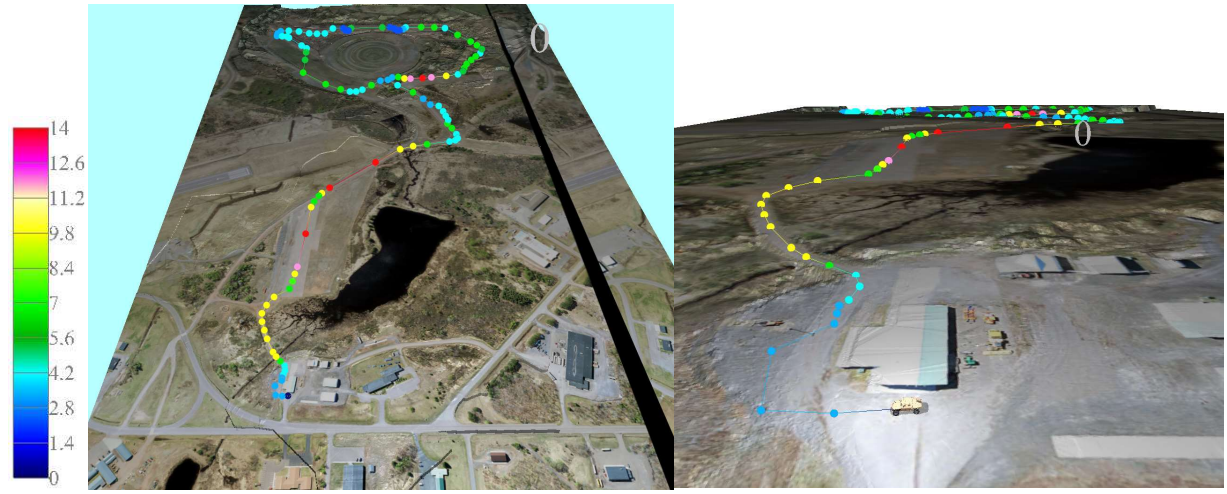


Figure 18 NATO AVT-341 KRC terrain map imported into IVRESS/DIS. The path color indicates the desired vehicle speed in m/s.

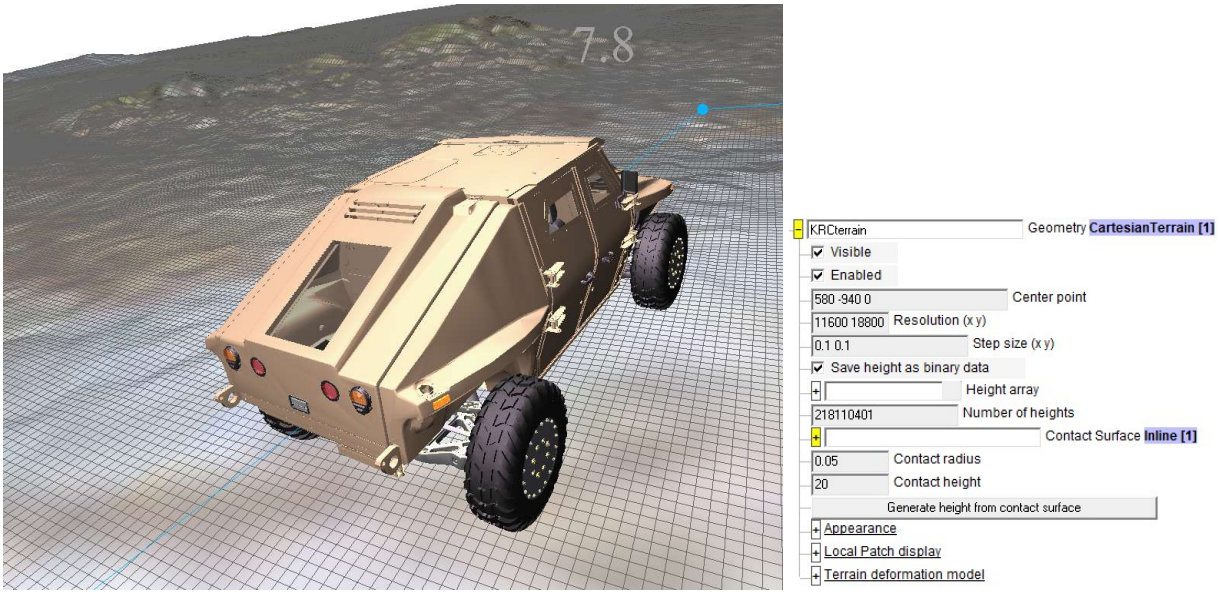


*Figure 19 NATO AVT-341 KRC terrain map imported into IVRESS/DIS as a Cartesian elevation grid. The IVRESS properties for the **CartesianTerrain** object are shown on the left. The resolution of the grid is 11600 ✕18800 with a cell size of 0.1 ✕0.1 m.*

Figure 20 Snapshots of the FED-Alpha MBD vehicle simulation in IVRESS/DIS on the AVT-341 scenario path. The total simulation time is about 600 s.

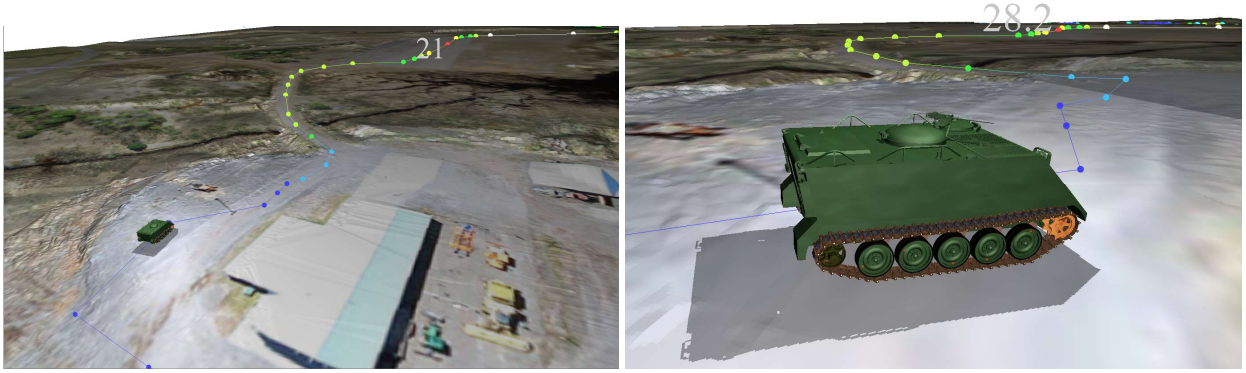


Figure 21 Snapshots of the M113 MBD vehicle simulation in IVRESS/DIS on the AVT-341 scenario path.

The same KRC terrain is created / imported in Unreal (see next Section). Then, the NATO AVT-341 FED-Alpha vehicle simulation scenario shown in Figure 18 and Figure 20 is visualized in Unreal using the co-simulation API presented in this section. The DIS solver runs in real-time on one computer. The Unreal engine runs on another computer. The DIS solver sends the following data each synchronization time step (0.03 s) to the Unreal engine:

- Position of the center of each rigid body (3*double).
- Quaternion of each rigid body (4*double).

Figure 22 shows a Humvee MBD vehicle model along with the FED-Alpha model displayed in Unreal. For the FED-Alpha vehicle the 17 rigid bodies shown in Figure 5 are sent each frame to Unreal. The Camera is attached to the frame of the FED-Alpha vehicle in Unreal. Snapshots of the resulting FED-Alpha DIS – Unreal co-simulation are shown in Figure 23. Note that this framework supports an arbitrary number of vehicles with each vehicle model consisting of an arbitrary number of rigid bodies. For the M113 vehicle 151 rigid bodies (Figure 6) representing the vehicle body, 2 sprockets, 2 idlers, 5✕2 wheels, 5✕2 road arms, and 63✕2 track segments are sent each frame to Unreal. The Camera is attached to the main body of the M113 vehicle in Unreal. Snapshots of the resulting M113 DIS – Unreal co-simulation are shown in Figure 24.

Figure 22 Models of the Fed-Alpha (right) and HUMVEE (left) vehicles in Unreal.

Figure 23 Snapshots from the FED-Alpha multibody dynamics vehicle AVT-341 scenario simulation in DIS displayed using the Unreal engine. The positions and quaternions of the 17 vehicle rigid bodies are sent using the Networking API to the Unreal engine in each simulation frame.

Figure 24 Snapshots from the M113 multibody dynamics vehicle AVT-341 scenario simulation in DIS displayed using the Unreal engine. The positions and quaternions of the 151 vehicle rigid bodies are sent using the Networking API to the Unreal engine in each simulation frame.

Task 2.2: Integrate the Unreal Engine

A sample virtual environment was developed in Unreal engine to test the AVSP. The requirements of the sample environment are:

1. It should be based on a real location to demonstrate the capability of simulating actual and relevant scenarios.
2. It should include a variety of topographic features such as slopes (hills and ditches) and surface roughness.
3. It should include a variety of terrain features such as various soil and vegetation types.
4. It should include man-made structures.
5. An accurate height data set of the area should be readily available.
6. Availability of data on the different soil types in the area is highly desirable.

The vehicle testing site of Michigan Technological University's Keweenaw Research Center (KRC) [32] was chosen as the sample virtual environment because it satisfies all of the above requirements. Lidar scans of a 900 acre Area of Operation (AO) at KRC were used to generate terrain height data of the AO as a Triangularized Irregular Network (TIN) mesh that was exported as an xml file (Figure 25). The data was generated by KRC in support of the NATO AVT-341 research task group: "Mobility Assessment Methods and Tools for Autonomous Military Ground Systems" [33]. Since ASA is involved with the AVT-341, we have access to that data and we will be using it to test the AVSP. The xml TIN data of the topography of KRC's AO was read using the Carlson P3D software [34] and converted to a dae format solid model (Figure 26). The MeshLab software [35] was then used to convert the solid model to a surface mesh in stl format (Figure 27). The SMS 13.0 software [36] was then used to convert the surface mesh into a node scatter, which was then converted also using the SMS 13.0 software into a raster 2D 32-bit tif file where the height data is encoded into the color of the terrain elements (Figure 27). The tif file was finally converted into a 2D 16-bit greyscale png file using the GDAL command line tool with the gdal_translate command. GDAL is distributed with the QGIS software package [37]. The png format encodes a terrain's height data in its greyscale where the dark areas represent low elevations and the light areas represent high elevations (Figure 27).

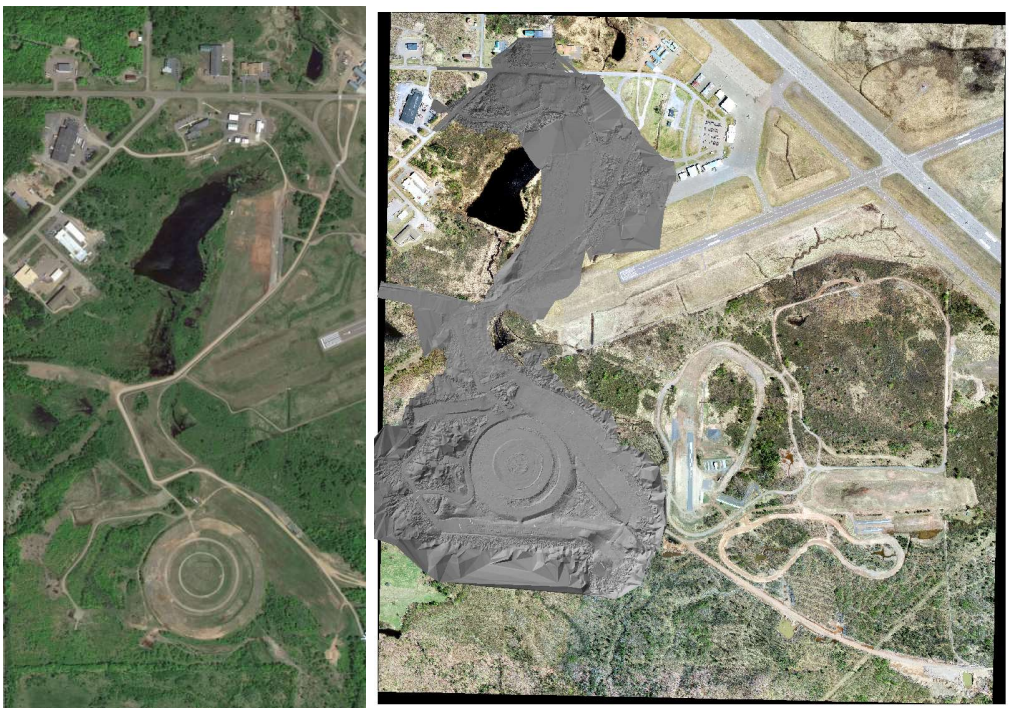


Figure 25 Aerial view of KRC's Area of Operation (AO) (left). The terrain height data model of the AO superimposed on KRC's aerial view (right).

The png file was then imported into the Unreal engine using its terrain import function (Figure 27). The x and y dimensions of the terrain were scaled by a factor of 30.48 to convert their dimensions from feet to centimeters which is the length unit used in Unreal. A cylinder with a diameter of 340 meters and a height of 38.34 meters was then generated in Unreal to check the scale of the terrain. The 340 meters diameter was chosen because it is the diameter of the circular feature that is found at the bottom middle of the AO. The 38.34 height dimension was chosen to be equal to the elevation difference between the highest point in the terrain and the lowest point. The cylinder was

superimposed on top of the terrain (Figure **28**) and the correctness of the x and y scales of the terrain was verified. The z scale of the terrain was then adjusted by a factor of 7.49 to match the height of the cylinder.

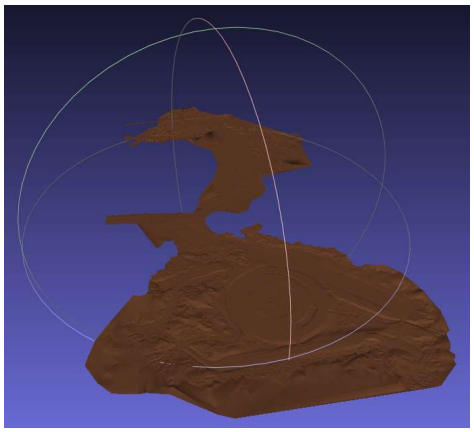
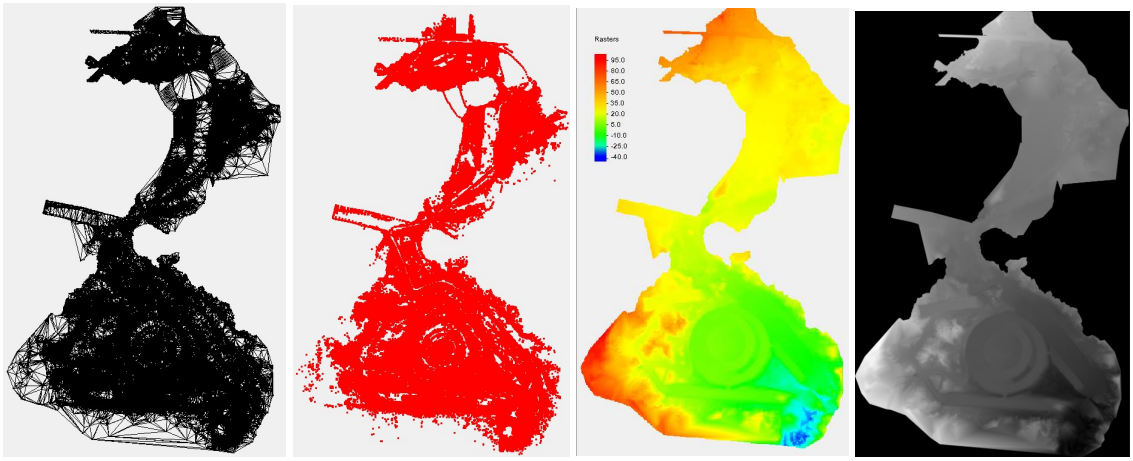


Figure 26 Isometric view of a solid model of KRC's Area of Operation.



(a) (b) (c) (d)

Figure 27 KRC Area of Operation shown as: (a) stl surface mesh, (b) node scatter, (c) 32-bit color tif raster, and (d) 16-bit greyscale png raster.

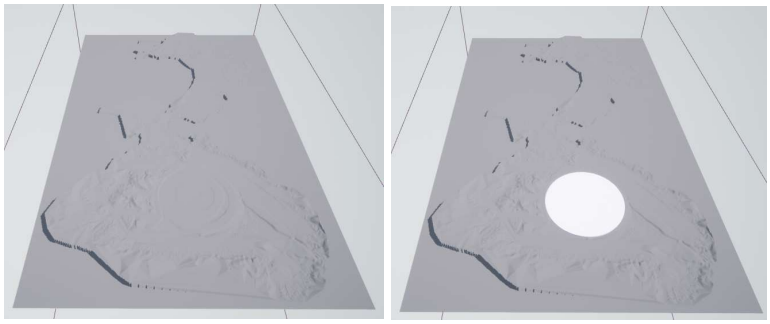


*Figure **28** The terrain model of KRC's Area of Operation (AO) after it was imported into Unreal (left). A 340 m cylinder superimposed on the AO's circular feature to check the terrain's scale (right).*

The solid models of six buildings and a long concrete pad located in a staging area in KRC were generated (Figure 29 and Figure 30). The staging area serves as the start and end points of an autonomous vehicle movement scenarios that is part of the NATO AVT-341 project. Models of the FED-Alpha and HUMVEE vehicles were also sent from IVRESS to the Unreal engine (Figure 22). Various objects such as cars, trucks, construction vehicles, and a shipping container were placed in the Unreal environment as props to try to mimic the look of the KRC site (Figure 31).

Figure 29 Picture of the Cold-Test Laboratory building at the KRC test site (left). Visualization using Unreal engine of a VR model of the Cold-Test Laboratory building (right).

Figure 30 Top view of the KRC staging area from Google Earth (left) and from a virtual environment in Unreal Engine (right). The Cold-Test Laboratory building is in the top middle.

Figure 31 Google Earth aerial view of part of the KRC test site's vehicle course (left). Visualization using Unreal Engine of the FED-Alpha vehicle on an approximate virtual copy of the terrain on the left (right).

The vegetation (Figure 32), and asphalt and dirt roads (Figure 33) were placed in their corresponding locations within the Unreal virtual environment. Furthermore, a course of concrete barriers (Figure 34) and fences (Figure 35) that are part of a NATO AVT-341 autonomous vehicle scenario were also placed in the virtual environment. The NATO AVT-341 autonomous vehicle scenarios also include animals crossing the vehicle's path (Figure 36), other vehicles crossing the vehicle's path (Figure 37), and search and rescue operations (Figure 38). We are currently setting up these and other test scenarios that will be used to evaluate the performance and capabilities of the AVSP. The area of the KRC test site encompassing the main building and its surroundings was modeled in Unreal (Figures 27-31). The landscape material set was also expanded by adding new textures for asphalt, dirt, mud, gravel, and snow. A water blueprint was used to model a nearby lake (Figure 31).

Figure 32 Visualization using Unreal Engine of the FED-Alpha vehicle on a dirt road with dense vegetation.

Figure 33 Visualization using Unreal Engine of the FED-Alpha vehicle driving on an asphalt road.

Figure 34 Aerial view of the location of the concrete barrier course from the NATO AVT-341 scenario (left). Visualization using Unreal Engine of the FED-Alpha vehicle going through the course (right).

Figure 35 Visualization using Unreal Engine of the FED-Alpha vehicle driving next to a chain-link fence.

Figure 36 Visualization showing two deer animals as part of a NATO AVT-341 scenario where an animal crosses the path of the autonomous vehicle.

Figure 37 Visualization of a scenario where a pickup truck crosses the autonomous vehicle's path.

Figure 38 Visualization of a search and rescue scenario involving an overturned SUV.

Figure 39 Aerial view (left) and VR visualization (right) of the KRC main building and staging area.

Figure 40 Image (top) and VR visualization (bottom) of the front view of the KRC main building.

Figure 41 Image (left) and VR visualization (right) of the left side view of the KRC main building.

Figure 42 Image (left) and VR visualization (right) of the right side view of the KRC main building.

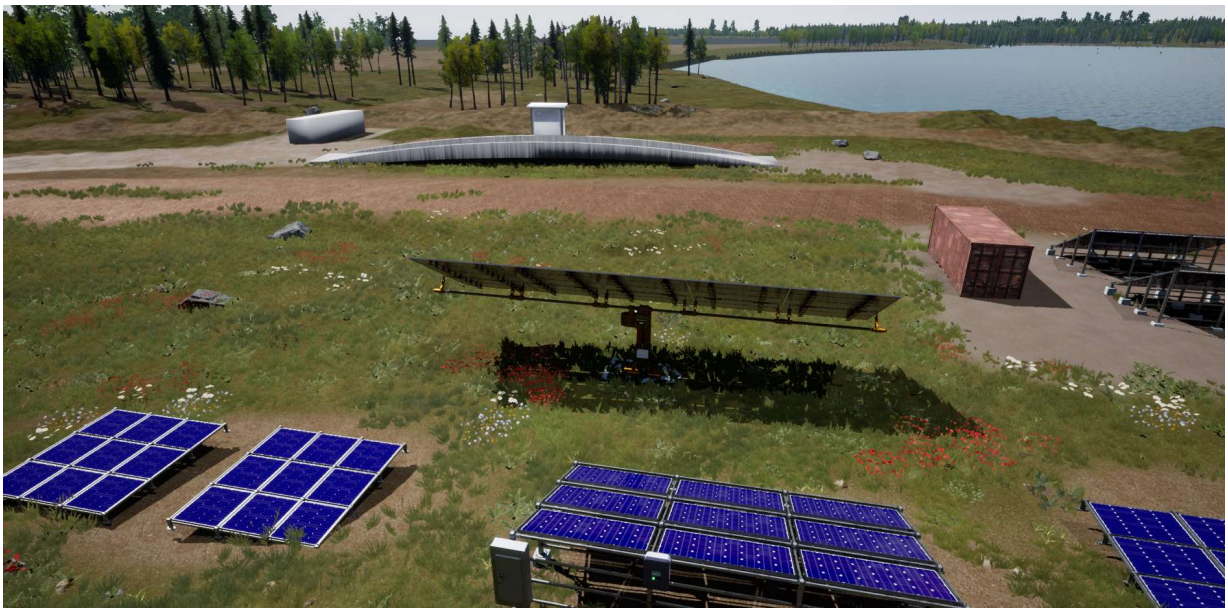


Figure 43 View from the top of the main KRC building looking behind the building at the solar panel array, the vehicle scale, and the lake.

The capability to display objects moving along a spline was added to the environment (Figure 44). This allows the simulation of stationary, walking and running pedestrians (Figure 45), moving vehicles (Figure 46), and animals (Figure 47) that can cross the path of the autonomous vehicle.

Figure 44 Top view of KRC staging area showing 4 walking and 2 stationary pedestrian splines.

Figure 45 KRC staging area showing 3 walking and 2 stationary pedestrians (left). Close up of a walking pedestrian (right).

Figure 46 Visualization of a dump truck moving along a spline path.

Figure 47 Visualization of a stag deer running across the path of the Fed-Alpha vehicle.

Several environmental conditions were added to the simulation. These include fog (Figure 48), rain (Figure 49), hail (Figure 50), and snow (Figure 51). The snow and hail differ mainly in the way the move; while the hail particles fall straight down at high speed, the snow particles float down more slowly while moving side to side. Furthermore, a day and night system was incorporated into the simulation. The system takes as input the longitude, latitude, and elevation of the simulation's location, along with the date. The day and night system then calculates the positions of the sun and the moon based on the location's information, the date, and the time of day (Figure 52, Figure 53, and Figure 54). The day and night system can update the time of day and the corresponding sun and moon locations dynamically during the simulation. A night sky was added, and the day sky was improved with animated cloud movement.

The KRC environment was replicated 2 more times to simulate a total of three different seasons: spring/summer, fall, and winter. Table 2 shows the elements of the environment that change for the different seasons. Figure 55 and Figure 56 show visual comparisons of different areas of the KRC VR environment for the 3 simulated seasons.

Table 2 Elements of the environment that change between seasons.

Environment component	**Spring/Summer**	**Fall**	**Winter**
Trees: non-evergreen	Green leaves	Yellow or red leaves	No leaves, snow covered
Trees: evergreen	Green leaves	Green leaves	Green leaves, snow covered
Rocks	Regular	Regular	Snow covered
Grass	Green with flowers	Yellow no flowers	Snow covered
Other ground textures	Regular	Regular	Snow covered
Asphalt roads	Regular	Regular	Snow texture

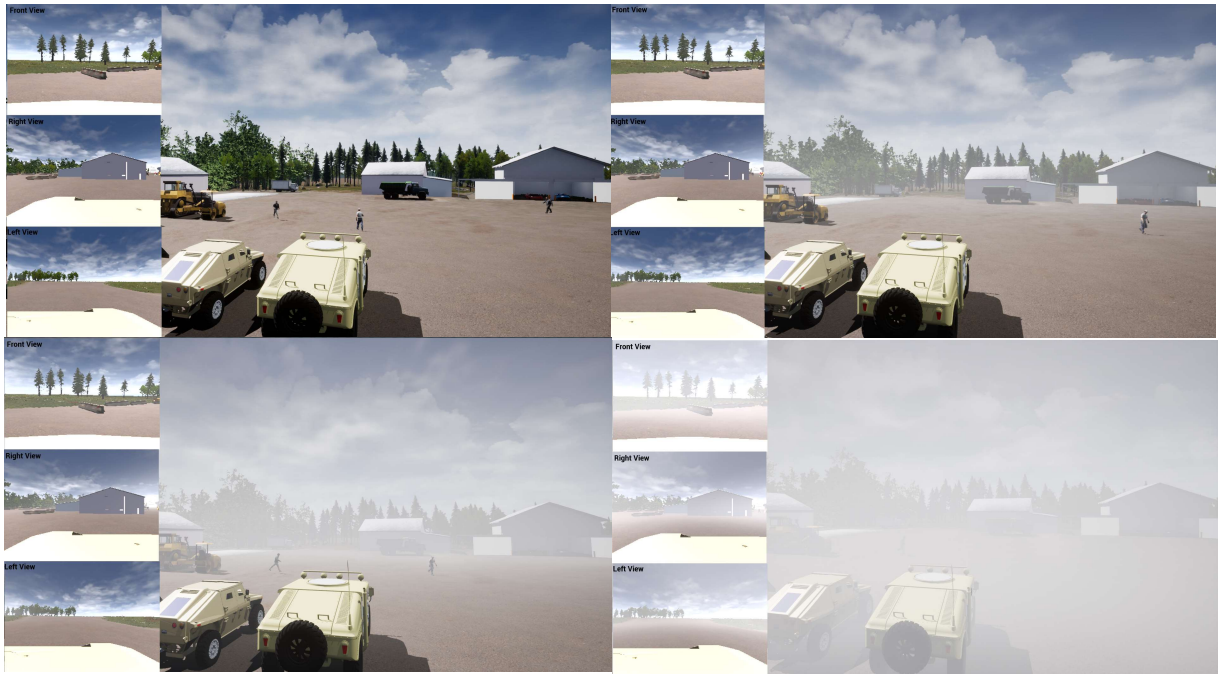


Figure 48 Visualization of environmental fog showing no fog (top left), light fog (top right), medium fog (bottom left), and heavy fog (bottom right).

Figure 49 Visualization of rain.

Figure 50 Visualization of hail.

Figure 51 Visualization of snow.

Figure 52 Visualization of the sun's location over the KRC area on April 26, 2021 at 4 PM (left) and 6 PM (right).

Figure 53 Visualization of the sunset over the KRC area on April 26, 2021 at 7 PM (left) and the sunrise at 5:30 AM (right).

Figure 54 Visualization of night time showing the moon's location at 12 AM (left), and the KRC main building parking lot at night (right).

Figure 55 Visualization of an asphalt road (left) and a dirt road (right) during spring (top), fall (middle), and winter (bottom).

Figure 56 Visualization of the KRC staging area (left) and a storage and vehicle parking area (right) during spring (top), fall (middle), and winter (bottom).

Task 2.3: Integrate Sensor Models

2.3.1 Visual Light Cameras

We added the capability to display (and send through the network) from the Unreal engine the camera frames from any number of cameras such as front, left side and right side cameras on typical autonomous vehicle sensor pods (e.g. Figure 57).

Figure 57 Sensor pod with 3 cameras (front, left and right facing cameras) and a Lidar sensor the roof of the FED-Alpha vehicle.

Figure 58 Front, left and right camera views from the sensor pod on the roof of the vehicle displayed on the left side of the screen.

2.3.2 Lidar

After testing the Lidar model of the Microsoft AirSim [25] Unreal plugin (Figure 59), we decided to develop our own Lidar model as an Unreal blueprint. This gives us more control over the Lidar model's parameters, eliminates some of the overhead computational cost that AirSim requires, and makes the installation of the final software simpler. The Lidar blueprint uses Unreal's *LineTraceByChannel* function to send out rays from a specified location. The blueprint controls the number of rays that are sent out per time tick, the duration of the time tick, and the vertical and horizontal range of the emitter (Figure 60 and Figure 61). When the line trace collides with a Game Object in the Unreal environment, it returns the location of the hit. The normalized distance to the collision is used to interpolate between two colors. The hit points are then drawn in the unreal environment and colored using the distance from the emitter to the points (Figure 62). We are working on implementing a Radar model based on this Lidar model.

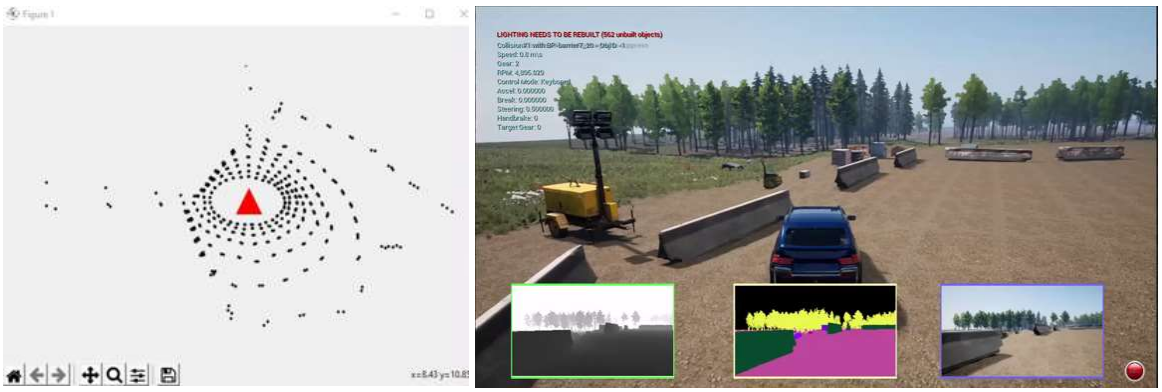


Figure 59 Demo vehicle moving over the KRC terrain while outputting Lidar data in real-time (left) using AirSim.

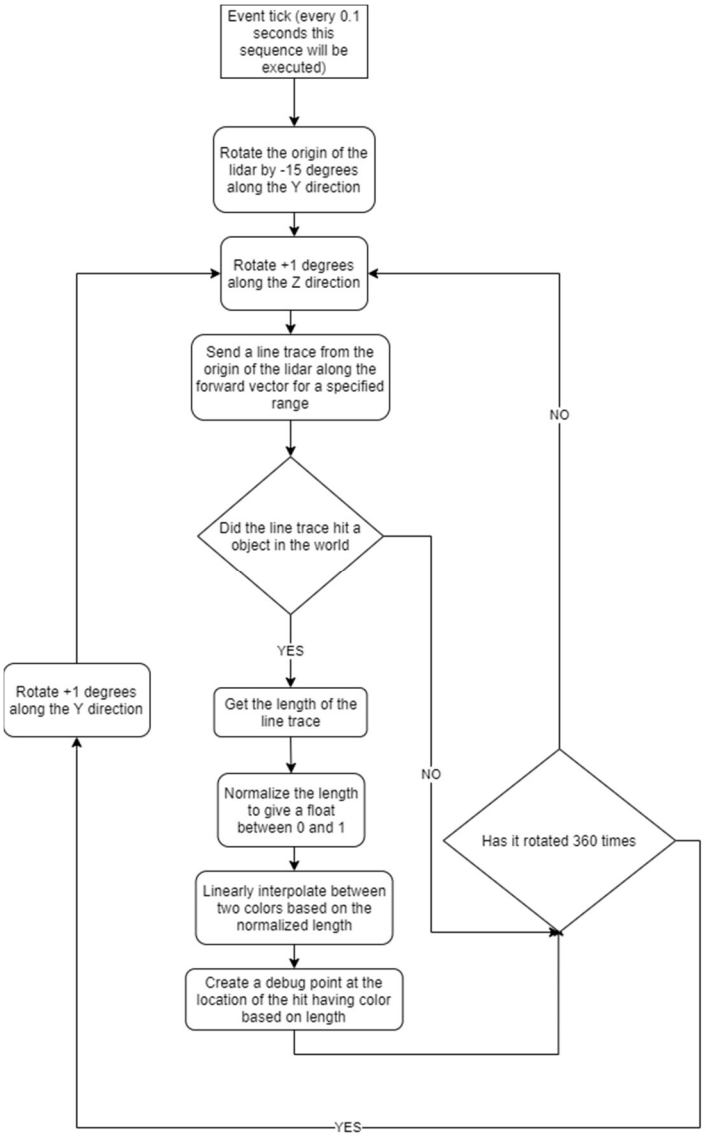


Figure 60 Flow chart of the Lidar algorithm (blueprint) that was developed in Unreal.

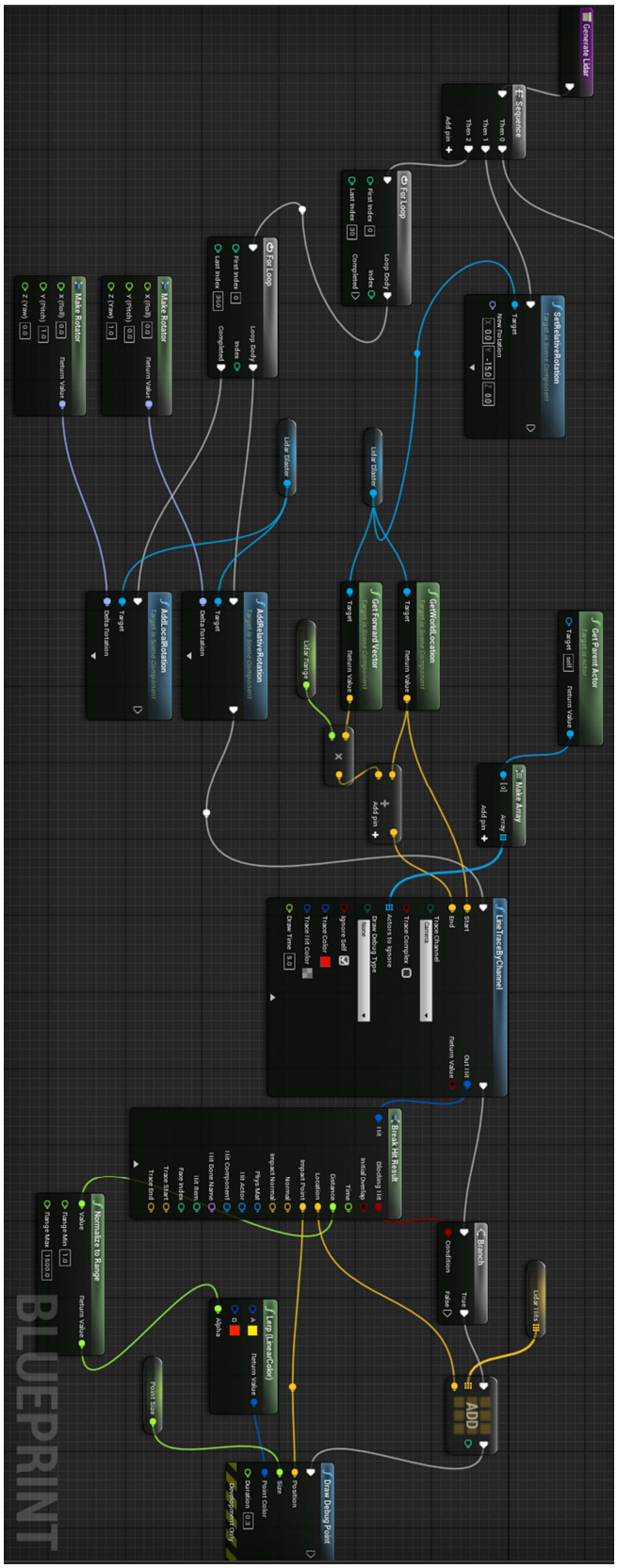


Figure 61 Unreal Lidar Blueprint script.

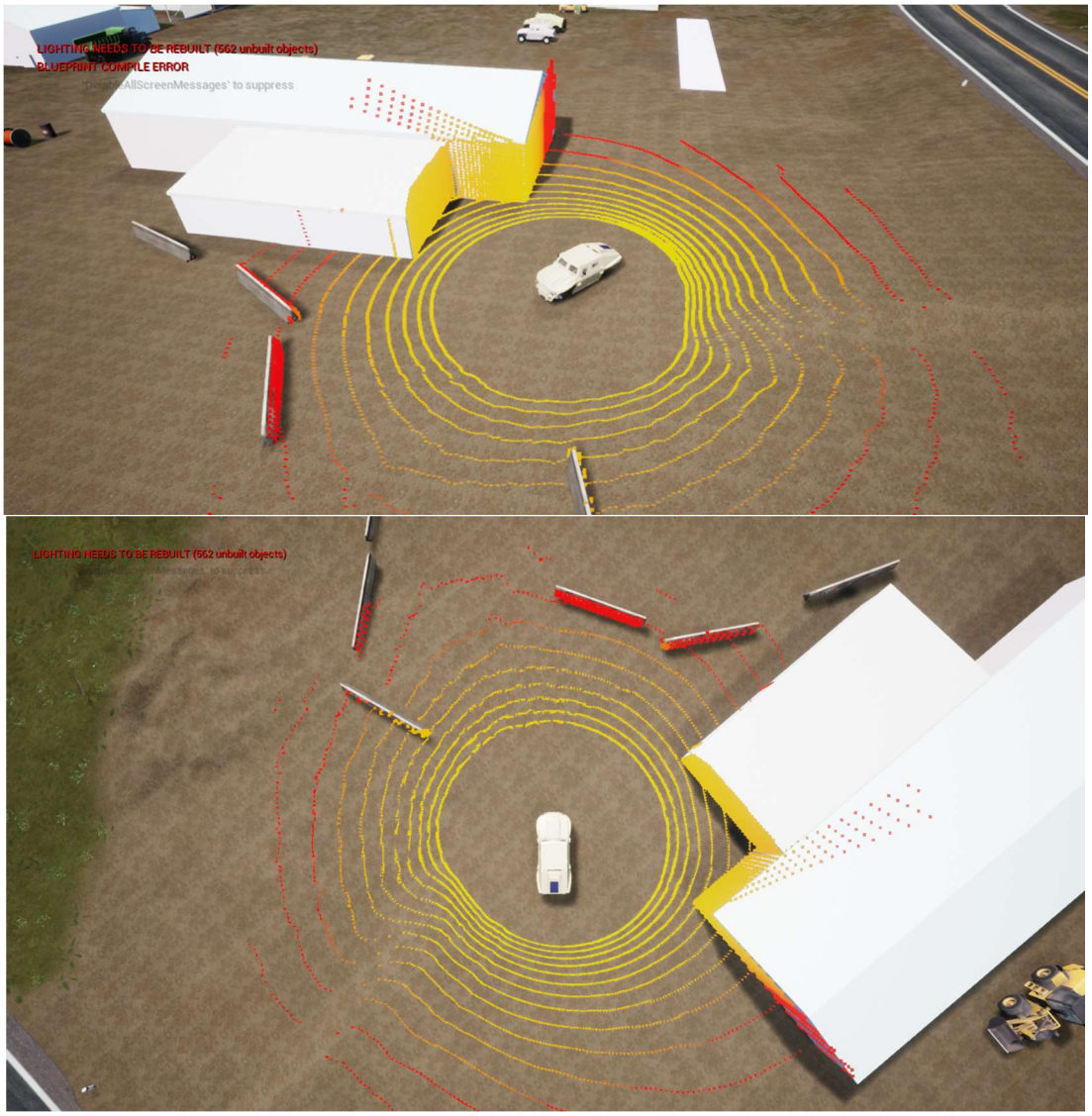


Figure 62 Lidar point cloud around the FED-Alpha vehicle within the Unreal environment colored by distance from the vehicle.

Figure 63 shows snapshots from an IVRESS/DIS simulation of the FED-Alpha vehicle during a traverse on the KRC virtual terrain. The vehicle is visualized using Unreal with the Lidar sensor model point cloud. The Lidar is simulated using the custom Unreal Blueprint script shown in Figure 61. Figure 64 and Figure 65 show the same FED-Alpha traverse simulation from different points of view. Figure 64 and Figure 65 show the entire Lidar point cloud reflecting off the terrain topography and the objects in the virtual environments such as trees, fences, vehicles, structures, etc.

Figure 63 Snapshots from an VRESS/DIS simulation of the FED-Alpha vehicle of a traverse on the KRC virtual terrain displayed using Unreal with the Lidar sensor model point cloud. The Lidar point cloud can be displayed as shown in the camera views or hidden depending on the user's preference.

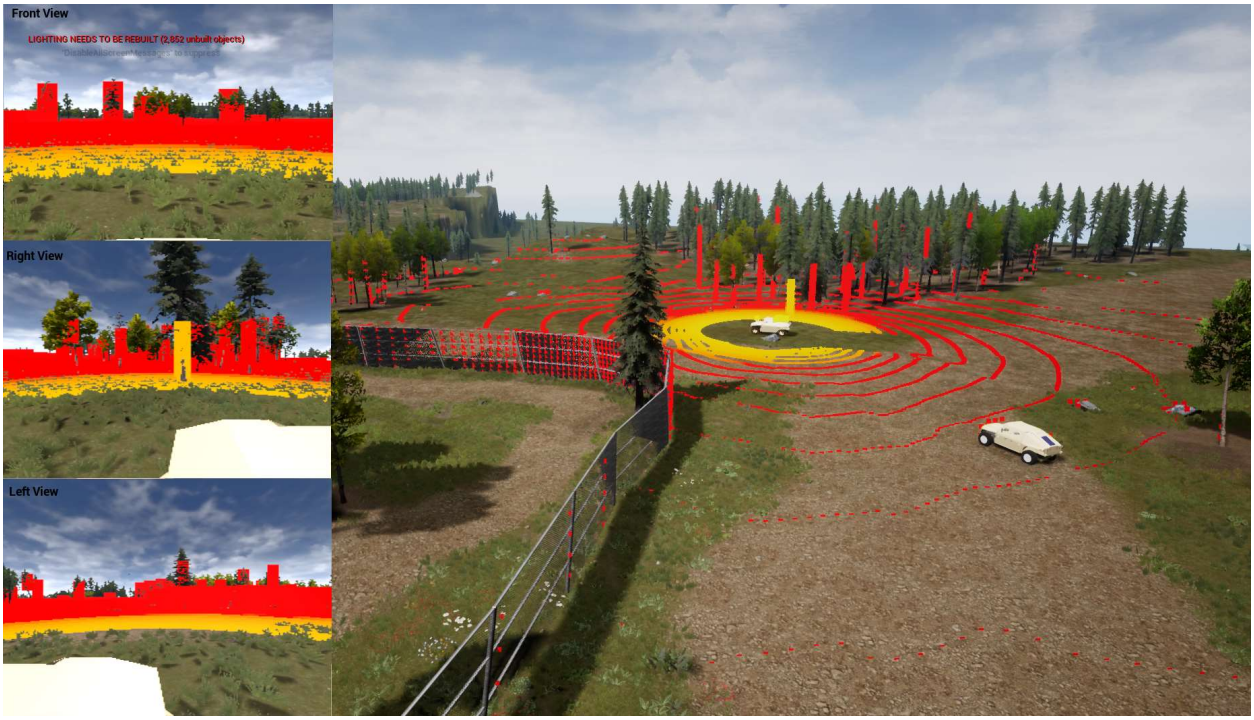


Figure 64 Snapshot from an IVRESS/DIS simulation of the FED-Alpha vehicle during a traverse on the KRC virtual terrain showing the Lidar point cloud reflecting off trees, a fence, and another following FED-Alpha vehicle. The Lidar range is 100 m.

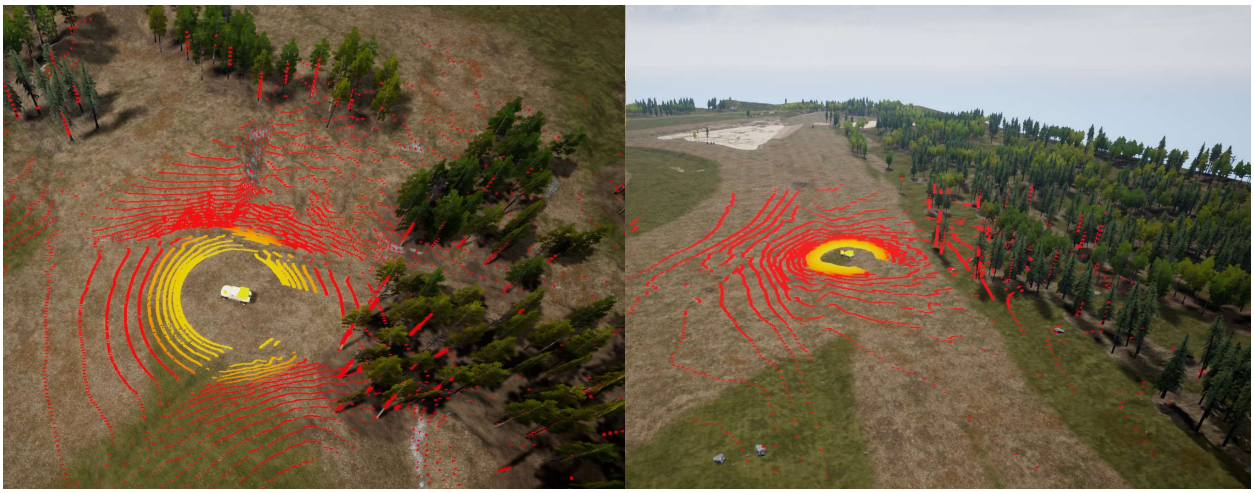


Figure 65 Top view of the FED-Alpha vehicle simulated using IVRESS/DIS during a traverse on the KRC virtual terrain visualized using Unreal with the Lidar sensor model point cloud.

In addition, we developed the capability to send the Lidar point cloud data through the network to other programs/computers. This is done through the network C++ API which is developed as an Unreal Class (see Task 2.1). Figure 66 shows snapshots of the FED-Alpha KRC traverse simulation showing the Lidar point cloud which is sent from Unreal to IVRESS every simulation frame using the co-simulation network API. The Lidar point cloud is sent as a ROS message in the format shown in Figure 67.

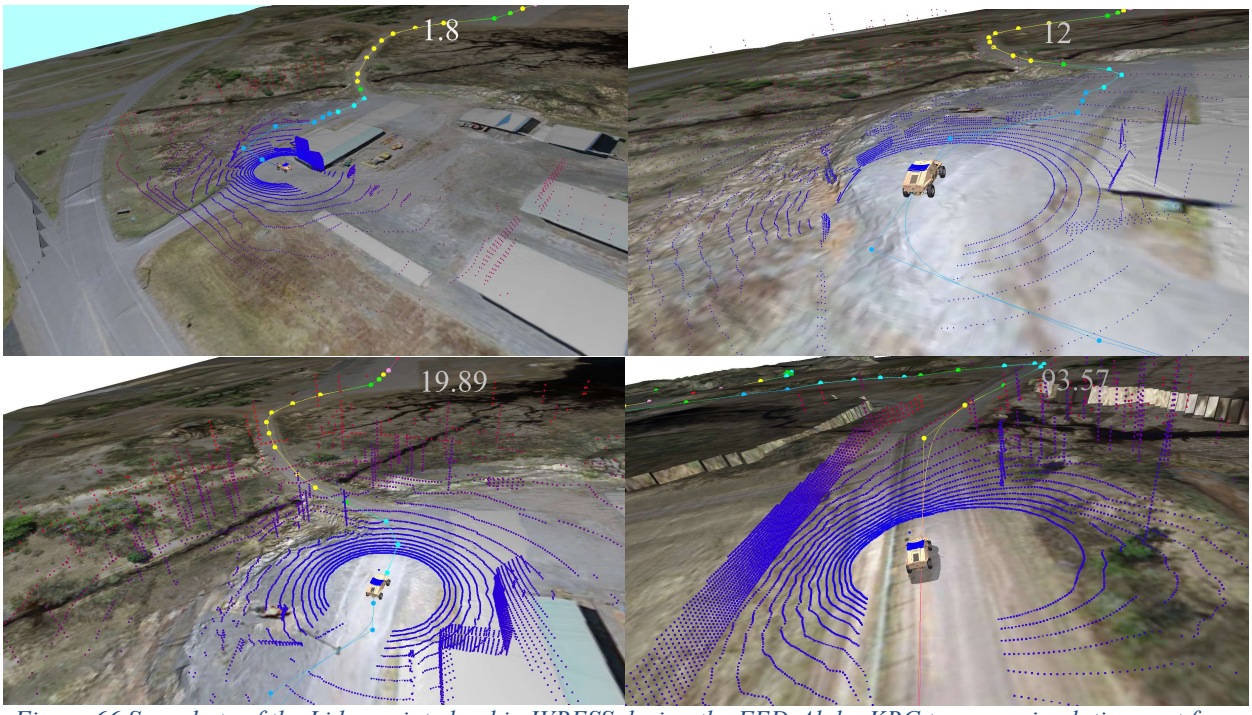


Figure 66 Snapshots of the Lidar point cloud in IVRESS during the FED-Alpha KRC traverse simulation sent from Unreal to IVRESS using the co-simulation network API.

File: sensor_msgs/PointCloud2.msg

Raw Message Definition

```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool    is_bigendian # Is this data bigendian?
uint32  point_step   # Length of a point in bytes
uint32  row_step     # Length of a row in bytes
uint8[] data         # Actual point data, size is (row_step*height)

bool is_dense        # True if there are no invalid points
```

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

```
header:
  seq: 0
  stamp:
    secs: 1527193450
    nsecs:  13024000
  frame_id: "velodyne"
height: 1
width: 1
fields:
  -
    name: "x"
    offset: 0
    datatype: 7
    count: 1
  -
    name: "y"
    offset: 4
    datatype: 7
    count: 1
  -
    name: "z"
    offset: 8
    datatype: 7
    count: 1
  -
    name: "intensity"
    offset: 16
    datatype: 7
    count: 1
is_bigendian: False
point_step: 32
row_step: 32
data: [218, 49, 151, 64, 105, 199, 245,
27, 0, 0, 182, 0, 0, 0, 0, 0, 0, 0]
is_dense: True
```

Figure 67 Lidar point cloud message format.

2.3.3 Radar

The Radar model uses the point cloud generated from the Lidar function created in the unreal engine. The Lidar blueprint uses Unreal's 'LineTraceByChannel' function to send out rays from a specified location. The blueprint controls the number of rays that are sent out per time tick, the duration of the time tick, and the vertical and horizontal range of the emitter. When the line trace collides with an Object in the Unreal environment, it returns the location of the hit. The hit points are then drawn in the unreal environment and colored using the distance from the emitter to the points. The point cloud data for radar is extracted and sampled at 10 hz. Then the clustering algorithm "DBSCAN" [11, 38] is used to group the points together and extract the centroid of that cluster. These centroids are considered core points, and each cluster is considered to be a detected object. The distance and angle of each point is plotted and shown by drawing a line from the vehicle to the center of each core point.

The HFOV (Horizontal Field of view) and VFOV (vertical field of view) of the radar sensor were set to 30° and 4.4° respectively to match typical values for a Radar sensor. When it was tested in the virtual KRC environment, the radar was able to detect the barriers along the field of view (Figure 68). Figure 68 shows the extracted point cloud plotted using small light points and the objects plotted with large darker points with the cross showing the centroid of each cluster. The clusters represent the barriers and the other vehicles in front of the FED-Alpha vehicle. Figure 69 and Figure 70 show the clusters for trees, vehicles and some stray clusters from the ground in front of the FED-Alpha. Figure 71 shows a comparison of the Radar output simulated using the DBSCAN algorithm versus an actual typical Radar sensor output.

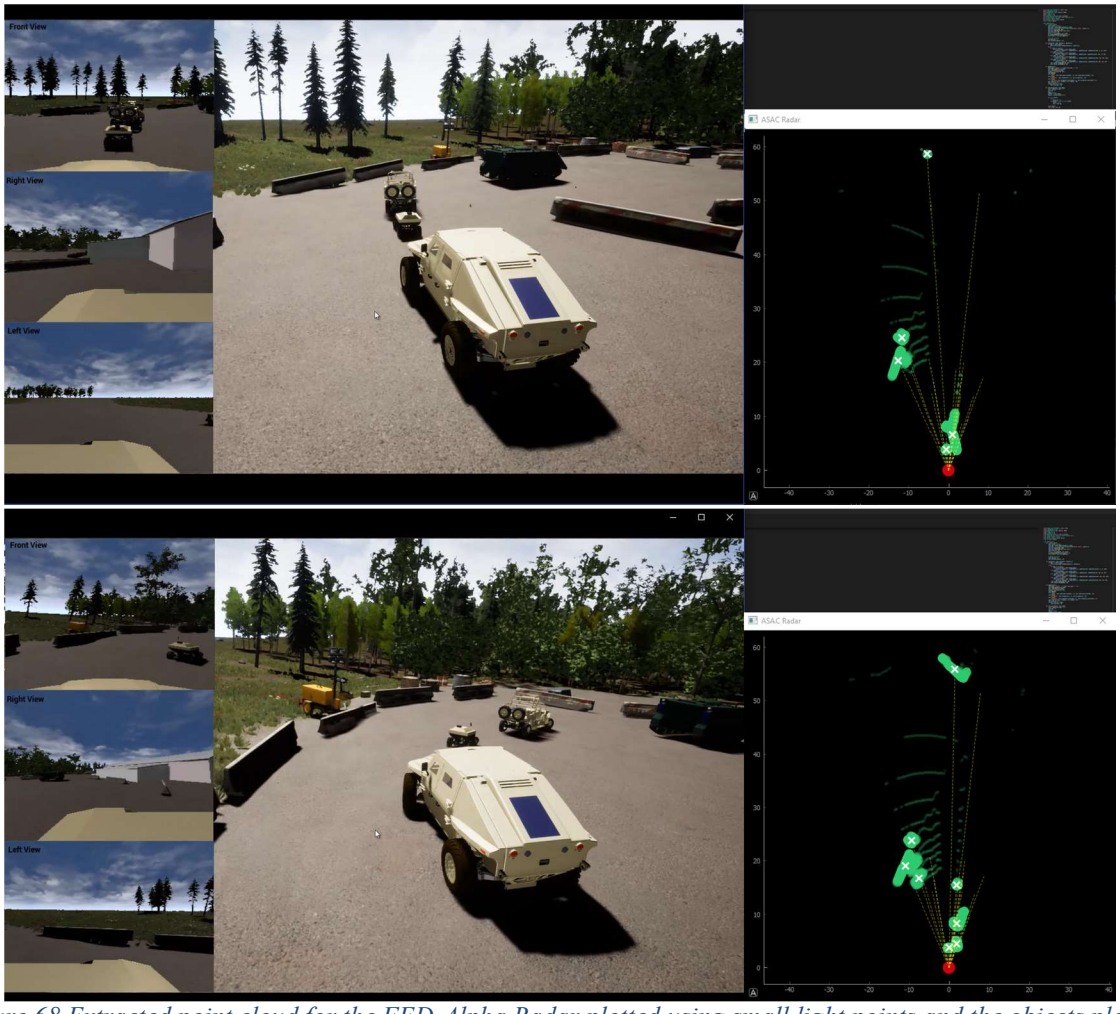


Figure 68 Extracted point cloud for the FED-Alpha Radar plotted using small light points and the objects plotted with large darker points with the cross showing the centroid of each cluster. The Radar sensor is placed on the front bumper of the FED-Alpha vehicle and is shown as a red dot.

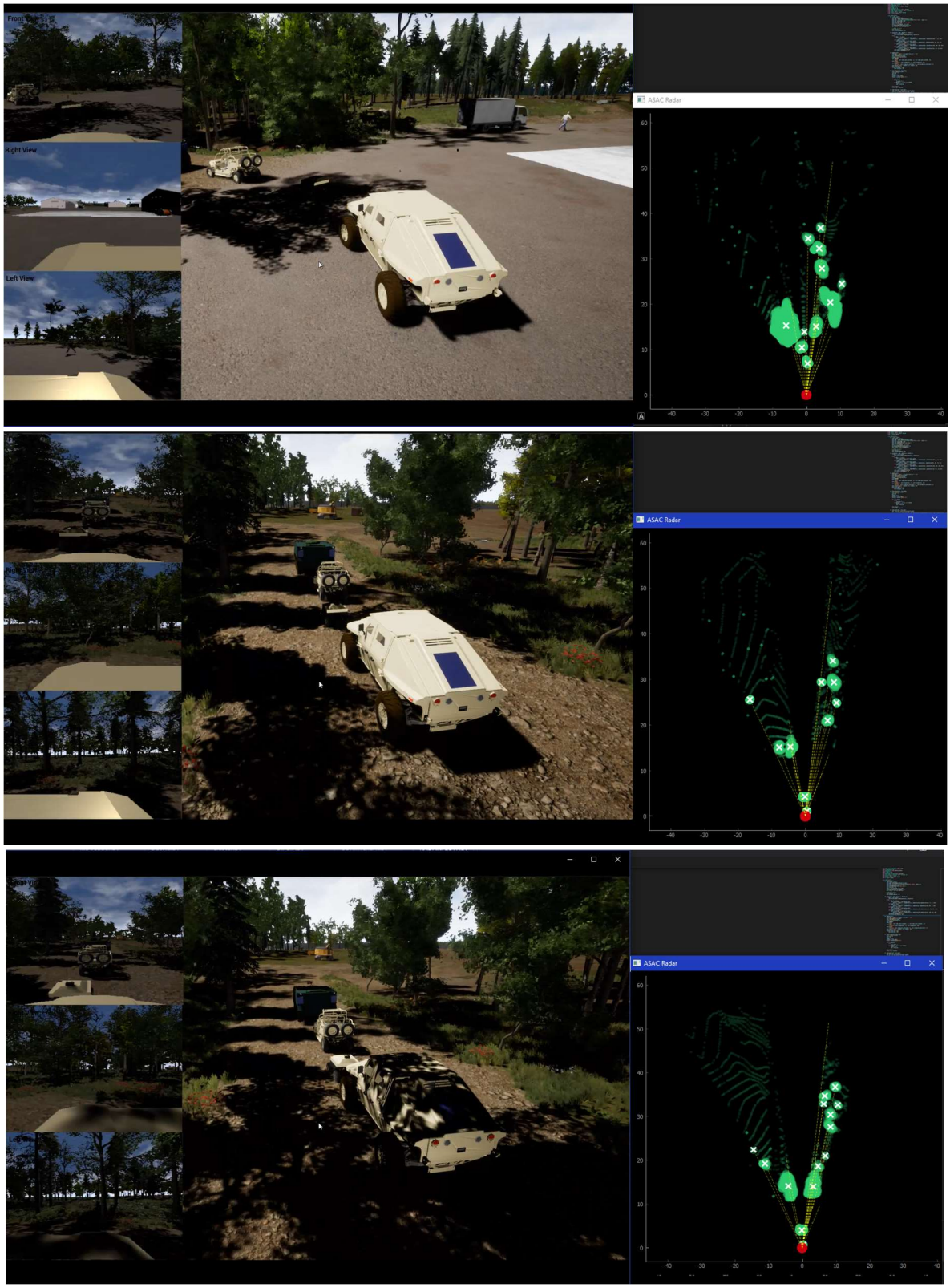


Figure 69 Extracted point cloud for the Radar showing the clusters for the trees and other vehicle.

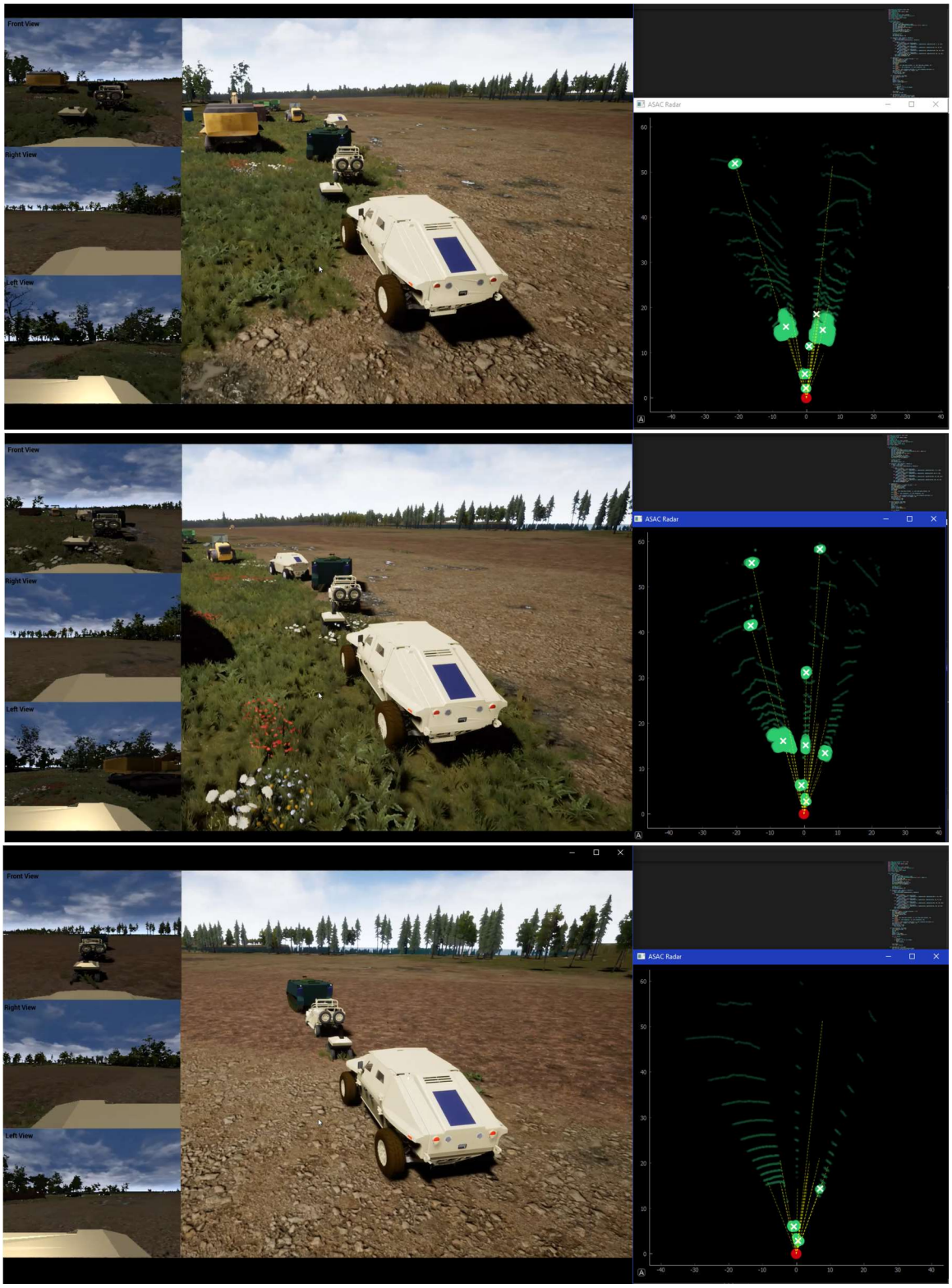


Figure 70 Extracted point cloud for the Radar showing the clusters for the vehicles in front of the FED-Alpha along with some stray rays.

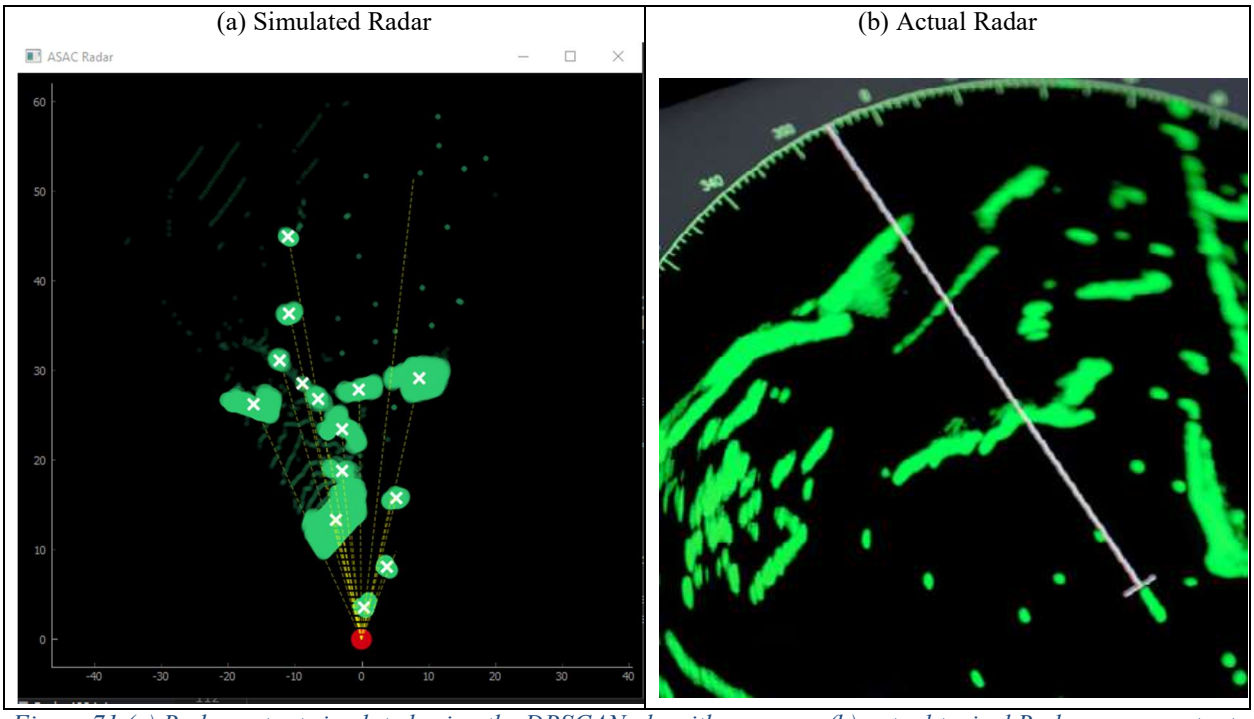


Figure 71 (a) Radar output simulated using the DBSCAN algorithm; versus (b) actual typical Radar sensor output.

Task 2.4: Integrate Simple Terramechanics

We implemented a simple terramechanics (ST) height-field model using the IVRESS/DIS Cartesian elevation grid terrain model (**CarterisanTerrain** object) shown in Figure 19. The potential *i-j* contact grid cell is found using:

$$i = (p_x - o_x)/d_x \qquad j = (p_y - o_y)/d_y \tag{2.4.1}$$

where *i*, *j* are the grid cell indices, p_x, p_y are the coordinates of the contact point on the tire, o_x, o_y are the coordinates of the corner point of the Cartesian grid, and d_x, d_y are the x, y sizes of a grid cell. A bilinear interpolation is then performed to find the height of the cell point under the contact point. The difference between the height of the cell contact point and the z-coordinate of the contact point is the penetration distance. If this distance is negative then no contact is detected otherwise contact is detected. Thus, contact search cost is reduced to evaluating Equation (2.4.1).

Each grid point on the Cartesian elevation grid point has a known area *A* and 4 vertical deformation (sinkage) variables:

1. z_p: permanent plastic deformation. z_p is calculated as follows:

   The normal stress σ_n is calculated using:

$$\sigma_n = \frac{F_n}{A} \tag{2.4.2}$$

   where F_n is the normal force applied by the vehicle running gear on the terrain grid point. z_p is calculated using:

$$\rho_0 = \frac{m}{A\,h_0} \quad \rightarrow \quad m = A\,h_0\,\rho_0$$
$$\rho = \frac{m}{A\,(h_0 - z_p)} \quad \rightarrow \quad (h_0 - z_p) = \frac{m}{A\rho} \quad \rightarrow \quad z_p = h_0 - \frac{m}{A\rho} \tag{2.4.3}$$

   where h_0 is the depth to the bedrock (or to hard compacted soil). ρ_0 is the initial bulk density of the soil. The bulk density ρ corresponding to that normal stress is found from the bulk density versus normal stress curve from a hydrostatic compression physical test.

2. z_e: elastic deformation. z_e is calculated using:

$$z_e = (h_0 - z)E\,\sigma_n \tag{2.4.4}$$

where z is the total vertical deformation of the soil, E is the soil's elastic constant and σ_n is the applied normal pressure.

3. z_f: flow deformation. z_f is calculated using:

$$z_f = b_1\sigma_n + b_2 \int \sigma_n(t)\, v_r(t)\, dt \qquad (2.4.5)$$

where *t is time,* and v_r *is the relative velocity between the running gear and the soil.* b_1 and b_2 are factors which determine the bearing and shearing strengths of the soil under the running gear due to pressure and slip, respectively. Note that b_1 and b_2 can both be a function of the total depth z.

4. z_n: soil height gain due to flow from neighbors. z_n is calculated using:

$$z_n = \sum_i f(z_{f_i}, d_i) \qquad (2.4.6)$$

where f is a distance weighting function, , d_i is the distance to neighboring point i, and z_{f_i} is the flow deformation at neighboring point *i.*

The total soil vertical deformation (z) is given by:

$$z = z_p + z_e + z_f - z_n \qquad (2.4.7)$$

The maximum tangential traction stress τ that the soil can support is calculated using:

$$\tau = (c + \sigma_n \tan\varphi\ )\left(1 - e^{(-s/k_s)}\right) \qquad (2.4.8)$$

where s is the slip, c is the cohesion strength of the soil, φ *is the soil friction angle.* c and φ are directly obtained from an unconfined soil shear test. The value of k_s is experimentally tuned. s is calculated using:

$$s = \frac{v_{theoretica\ } - v_{actual}}{v_{theoretical}} \qquad (2.4.9)$$

where v_{actual} is the actual longitudinal vehicle speed and $v_{theoretica\ }$ is the theoretical vehicle speed if slip is zero. For a wheeled vehicle:

$$v_{theoretical} = r\ \omega \qquad (2.4.10)$$

where r is the tire's outer radius and ω is the tire's angular velocity. For a tracked vehicle:

$$v_{theoretical} = \frac{p\ N_{sprocket}}{2\pi}\ \omega \qquad (2.4.11)$$

where p is the track pitch length, $N_{sprocket}$ is the number of drive sprocket teeth, and ω is the drive sprocket angular velocity.

The maximum tangential traction force that the tire/track segment can generate is given by:

$$F_{max.traction} = \int_A\ \tau\ dA \qquad (2.4.12)$$

where A is the tire/track contact area. The soil resistance force on a tire/track segment due to bulldozing and rolling resistance is calculated using:

$$F_{soil\ resistance} = \left(c + \sigma_n \tan\varphi\ \right) w\ z_{rel} + f A\ \sigma_n \qquad (2.4.13)$$

where w is the width of the tire/track segment, z_{rel} is the relative vertical sinkage of the tire/track segment in the surrounding soil, and f is the soil-tire rolling resistance coefficient. Note that the contact patch area is a function of z_{rel}. If the total resistance force due to the soil, slope, and drawbar-pull is larger than the traction force when the slip is 100% then the vehicle will get stuck. Otherwise, the vehicle slip is found by setting the total traction force equal to the total resistance force, then calculating the required slip (s) using Equation (2.4.8).

Figure 72 and Figure 73 show the tire rut on a Cartesian elevation grid ST terrain patch with the soil transport model (z_n) using equation (2.4.6) disabled and enabled, respectively. Figure 74 shows a snapshot of a terrain area modeled using the **CarterisanTerrain** object with the simple terramechanics model for displaying the tire ruts on the terrain. Figure 75 shows a snapshot of the entire KRC terrain **CarterisanTerrain** object with the FED-Alpha vehicle going over a prescribed test track on the terrain with the simple terramechanics used to model the tire ruts on the terrain.

We are working on sending the **CarterisanTerrain** terrain grid points which have moved at the current time step (display frame) from DIS to Unreal using the co-simulation API presented in Task 2.1. This will allow the terrain in Unreal to show the ruts that are formed by all moving vehicles on the terrain.

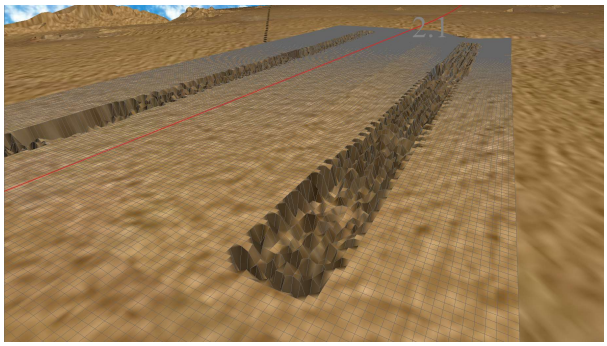


Figure 72 Tire rut on the Cartesian elevation grid ST terrain patch with the soil transport model disabled.

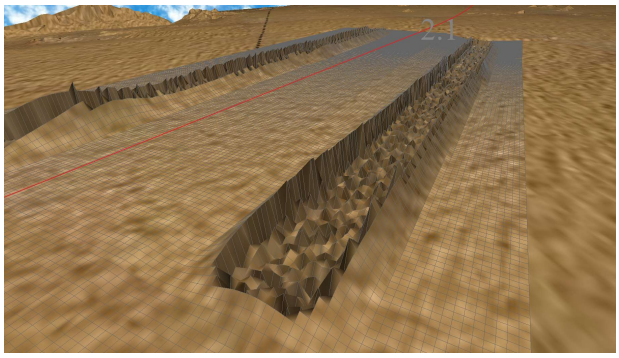


Figure 73 Tire rut on a Cartesian elevation grid ST terrain patch with the soil transport model enabled.

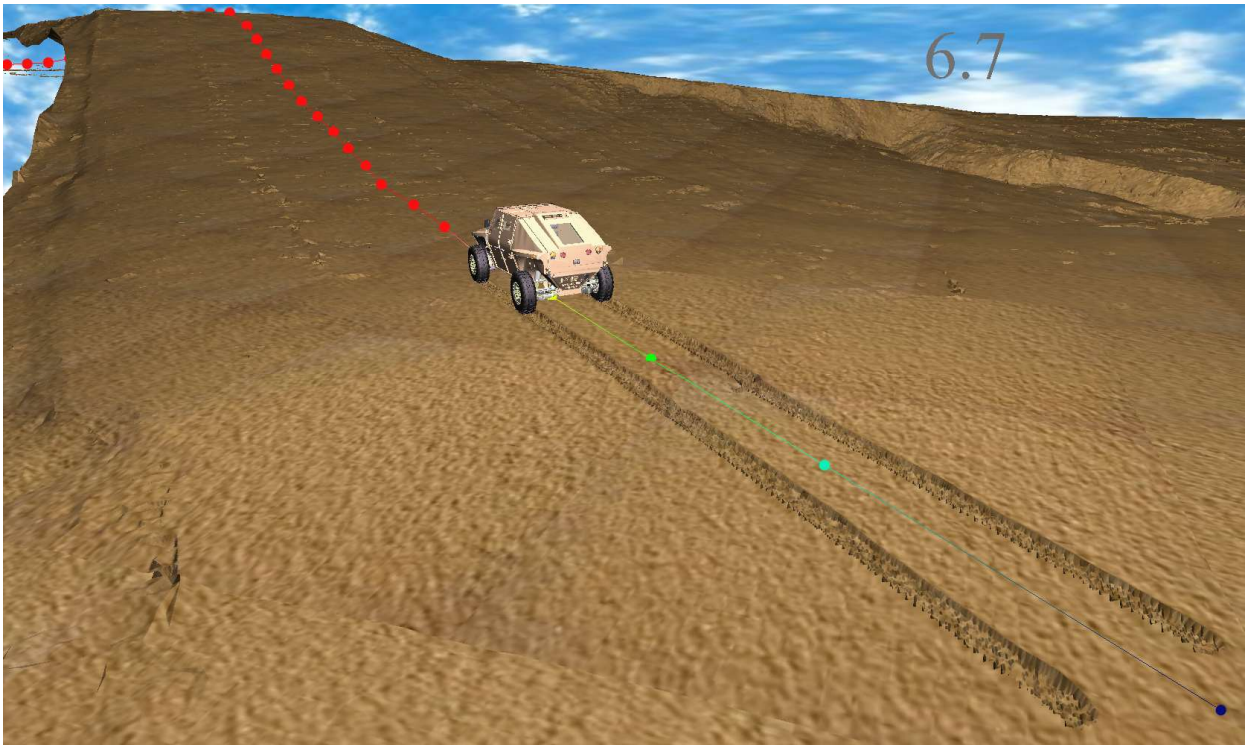


Figure 74 MBD FED-Alpha vehicle model along with a simple terramechanics terrain height-field model for displaying tire ruts using IVRESS/DIS.

Figure 75 IVRESS/DIS simulation of the FED-Alpha vehicle going over a simple terramechanics terrain height-field terrain modeled using the Cartesian elevation grid for the entire KRC terrain map. The grid cells that the tires contacted are displayed in red (upper-left). The ruts (sinkage) due to the vehicle tires on terrain are shown on the right and bottom figures. The grid point sinkage is calculated using a simple terramechanics model.

Task 2.5: Simulate four vehicles using parallel computing

2.5.1 AVSP demonstration using a four vehicle simulation with the NATO AVT-341 Simulation Scenario

A simulation scenario comprising four autonomous vehicles performing an AVT-341 operation scenario (which includes departure, reconnaissance, force protection and withdrawal) was developed in order to demonstrate the modeling capabilities of the AVSP including:

- Ability to include high-fidelity MBD models of multiple vehicles in the same simulation.
- Ability to simulate different types/classes of vehicles including: wheeled vehicles, tracked vehicles, large vehicles, small vehicles, and legged vehicles in the same simulation.

- Ability to represent the following environmental conditions: terrain topography, vegetation, natural and man-made static obstacles, dynamic agents (people, animals, and vehicles), weather conditions, and time of day / date conditions.
- Vehicle sensors including: visible light cameras and Lidar.
- Autonomous driving using way points with prescribed speed along with a leader-follower pursuit controller.

The following four vehicles were modeled in the scenario: FED-Alpha, M113, Polaris MRZR, and TracerX. IVRESS runs on one computer and Unreal runs on another computer. The IVRESS computer sends the vehicles' rigid bodies positions and orientations to the Unreal computer using the co-simulation API presented in Section 2.1.1.

A leader follower pursuit controller is used to maintain a separation distance from the follower vehicle to the leader vehicle in front. Each vehicle knows its own position. In addition, the pursuit controller receives from the leader vehicle in front its position and velocity. The pursuit controller then adjusts the vehicle speed of the follower vehicle in order to maintain the desired distance along the way point path to the leader vehicle. The desired distance can be varied either by the autonomy stack or by a human operator. The way point controller also allows setting a normal offset to the way point path for each vehicle. The distance to the leader vehicle along with the normal offset to the way point path can be used to create vehicle formations such as line, column, and diamond formations. The Lidar and camera sensors are simulated for two vehicles: the FED-Alpha and TracerX. Figure 76 shows a snapshot of the four-vehicle simulation on the KRC virtual terrain. The lead vehicle is the M113. It is followed by the Polaris MRZR which is followed by the TracerX then the FED-Alpha. Figure 77 shows four sequential snapshots from the start of the simulation scenario in daytime (the day is set to April 26 at 1 PM on the KRC site latitude and longitude). Figure 78 shows 4 sequential snapshots at nighttime (the day is set to April 26 at 10 PM on the KRC site latitude and longitude). Figure 79 and Figure 80 show the four-vehicle simulation along with the point cloud for the TracerX Lidar at daytime and nighttime. Figure 81 shows the four-vehicle simulation with dynamic agents (deer) crossing the path of the vehicles. Figure 82 shows the four-vehicle simulation with dynamic agents (deer) crossing the path of the vehicles. Figure 83 shows the four-vehicle simulation over the RMS lanes in the KRC test site. Figure 84 shows the four-vehicle simulation over a side slope. Figure 85 shows snapshots from the four autonomous vehicles simulation displayed in IVRESS showing the waypoints path along with the desired speed along the path.

Figure 76 Four autonomous vehicles simulation. The lead vehicle is the M113 and is followed by the Polaris MRZR, TracerX, and FED-Alpha.

Figure 77 Snapshots from the 4 autonomous vehicles simulation during daytime.

Figure 78 Snapshots from the 4 autonomous vehicles simulation at nighttime.

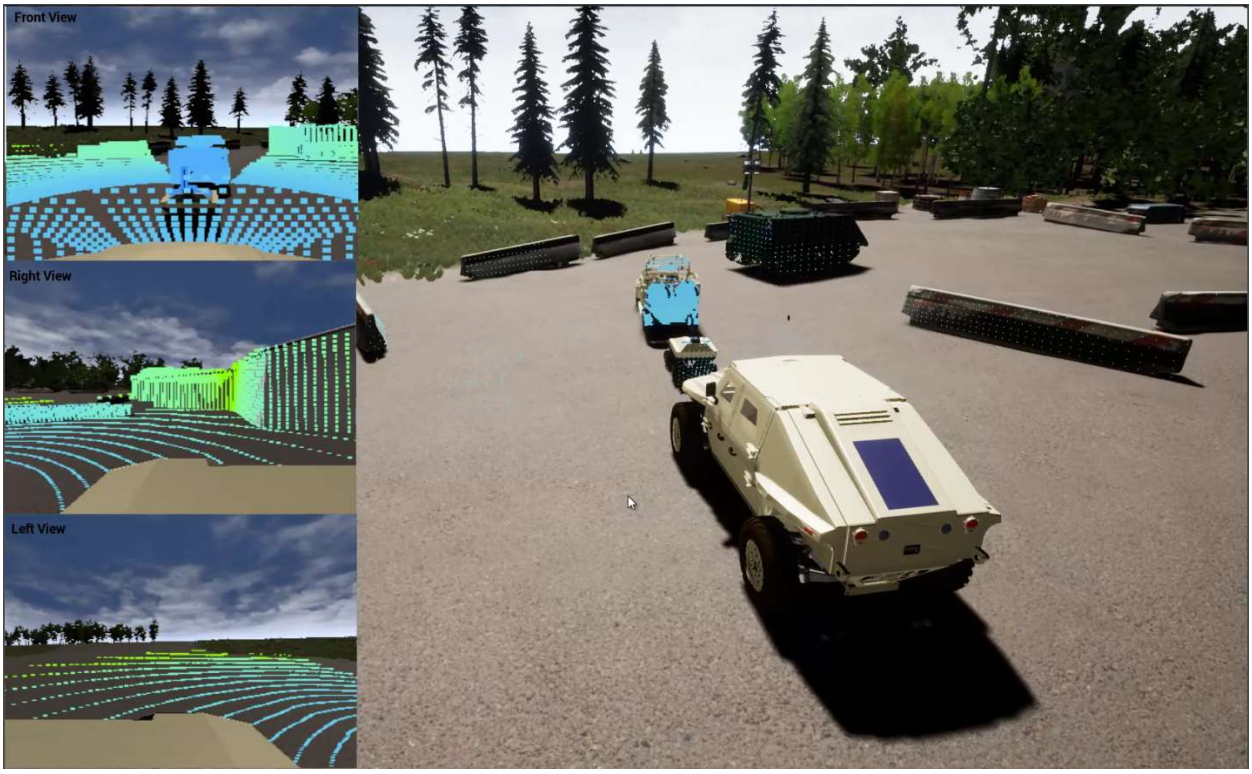


Figure 79 Snapshot from the 4 autonomous vehicles daytime simulation showing the Lidar simulation of the TracerX vehicle.

Figure 80 Snapshot from the 4 autonomous vehicles nighttime simulation showing the Lidar simulation of the TracerX vehicle.

Figure 81 Snapshots from the four-vehicle simulation with dynamic agents (deer) crossing the path of the vehicles.

Figure 82 Snapshots from the four-vehicle simulation over an off-road vegetation covered terrain.

Figure 83 Snapshot from the four-vehicle simulation over RMS lanes.

Figure 84 Snapshot from the four-vehicle simulation over a side slope.

Figure 85 Snapshots from the four autonomous vehicles simulation displayed in IVRESS going over the KRC terrain. In the bottom figure, the vehicles are going over the RMS lanes at the KRC site.

2.5.2 Demonstrate integration with a typical autonomy stack

We started integrating a simple autonomy stack in order to control the autonomous vehicles and close the simulation loop. The chosen stack was developed by Mississippi State University and provided by the NATO AVT-341 autonomy sub-group (https://github.com/CGoodin/avt_341). This stack is developed to work with any simulator which can be interfaced with ROS (Robot Operating System). Figure 86 shows the modules and data flow of the avt_341 autonomy stack. Four modules make up the avt_341 autonomy stack:

1) Perception: detects the general terrain slope then generates a spatial occupancy grid based on the slope.

2) Long-range Planner: Generates a road center line based a user-defined set of waypoints in global coordinates.
3) Local Planner: Uses the road center line along with the spatial occupancy grid to modify the vehicle path based on the static and dynamic obstacles detected in order to avoid collision with those obstacles.
4) Control: A PID-controller is used for speed control. A pure-pursuit PID algorithm with *kp* dependent on the vehicle speed and wheelbase is used for steering control.

The input to the avt_341 autonomy stack consists of:

1) Waypoints: The global path given in the form of waypoints to be followed by the vehicle, supplied in a ".yaml" file format.
2) Terrain map is provided as an occupancy grid in the form of a "nav_msgs/OccupancyGrid" message.
3) Odometry: position and orientation of the autonomous vehicle in the form of a "nav_msgs/Odometry" message.
4) Lidar data: point cloud of the Lidar data from the Lidar emission point on the vehicle in the form of a "sensor_msgs/PointCloud2" message.

The main output of the avt_341 autonomy stack is the "cmd_vel" message which contains the desired velocity vector data which is fed back to the simulator. The velocity vector can be transformed into the commanded throttle, braking and steering to the autonomous vehicle.

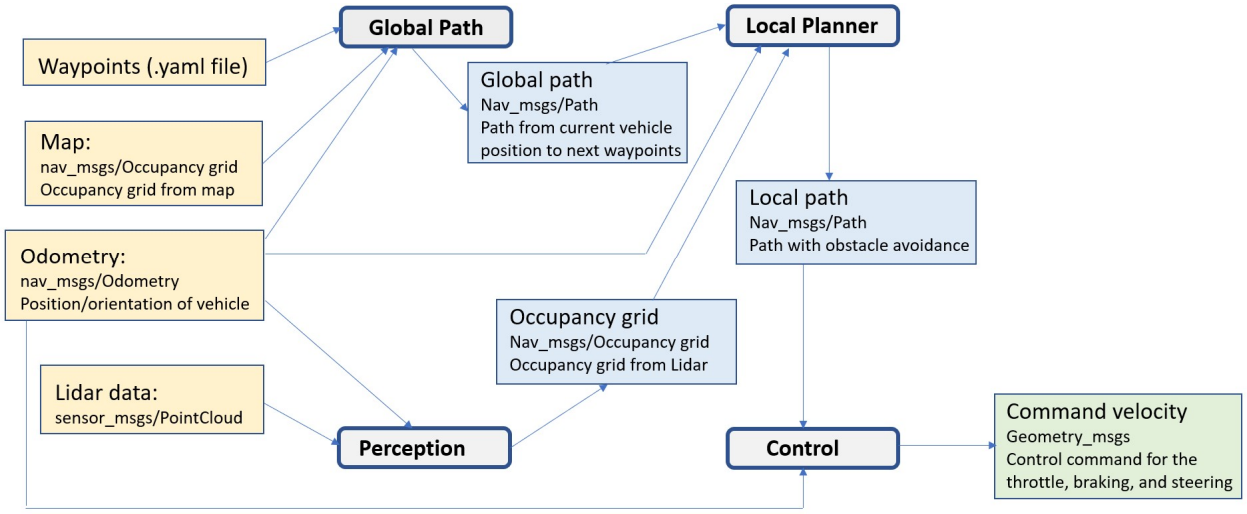


Figure 86 AVT-341 autonomy stack block diagram and data flow.

Figure 87 shows a block diagram of the data flow between the avt_341 autonomy stack, the Unreal Engine and IVRESS. The autonomy stack receives the following data from IVRESS:

- avt_341/nav_msgs/Odometry: Position/orientation data of the vehicle.
- avt_341/sensor_msgs/PointCloud2points: Pointcloud2 data from the Lidar (Figure 67).

The stack receives the vehicle position and Lidar data from IVRESS. If it detects an obstacle in the path of the vehicle, it uses a pure pursuit controller which gives necessary steering and velocity commands to avoid the obstacle. Meanwhile the local planner follows the center line of the path and tries to keep the vehicle on it. Thus the vehicle comes back to the path immediately after avoiding the obstacle.

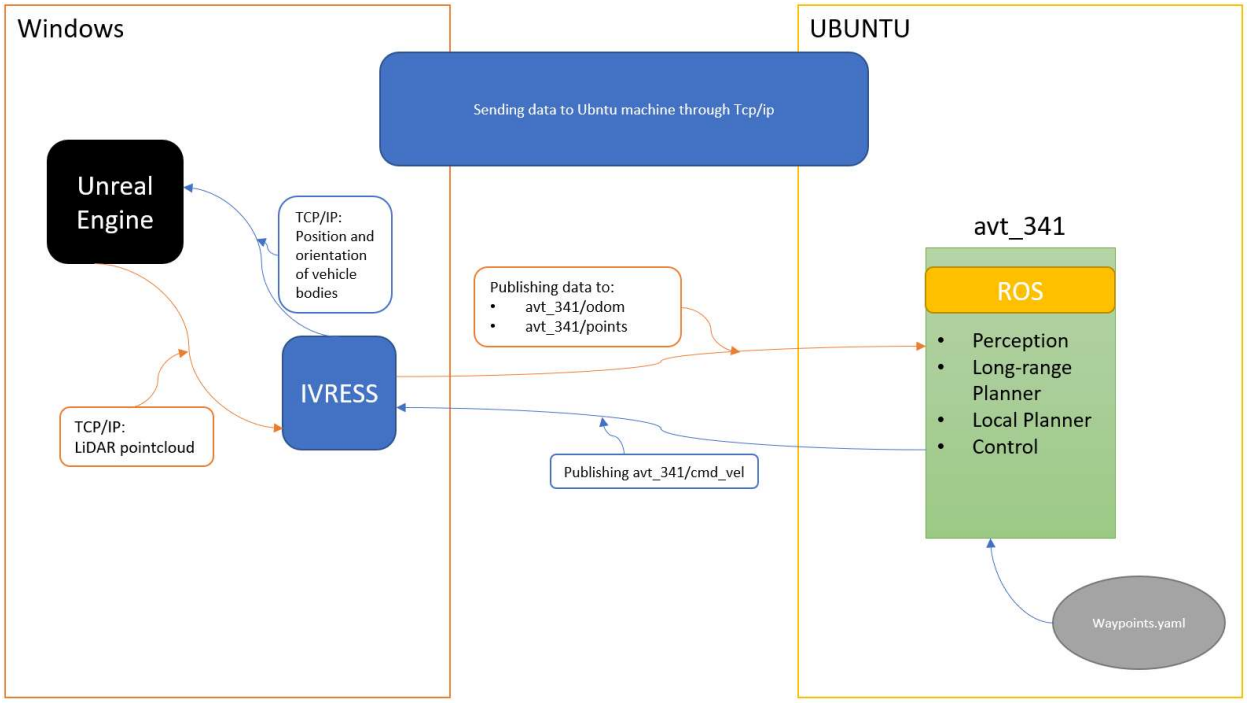


Figure 87 Block diagram showing the data flow between the avt_341 autonomy stack, the Unreal Engine (used to display the environment) and IVRESS (used to simulate the vehicle).

We are still in the process of developing the ROS message interface to send the Odometry and Lidar point cloud to the avt_341 autonomy stack from IVRESS and receive the commanded velocity back from the stack. However, we tested the avt_341 autonomy stack with Microsoft AirSIM in order to demonstrate the integration of an autonomy stack with an autonomous vehicle simulator. AirSIM runs inside Unreal and has APIs for ROS communication. A block diagram showing the data flow between AirSIM and avt_341 autonomy stack is shown in Figure 88. In order to integrate the avt_341 autonomy stack with AirSIM, we had to remap the vel_cmd which contain the desired vehicle velocity vector published by the stack to /airsim_node/drone_1/car_cmd which contains the throttle, braking, and steering commands to the vehicle. The throttle and brake values are real numbers from 0 to 1 and the steering value is a real number between -1 and 1. Figure 89 shows a snapshot of an AirSIM vehicle controlled by the avt_341 autonomy stack avoiding a red sphere obstacle which is in the waypoint path of the vehicle.

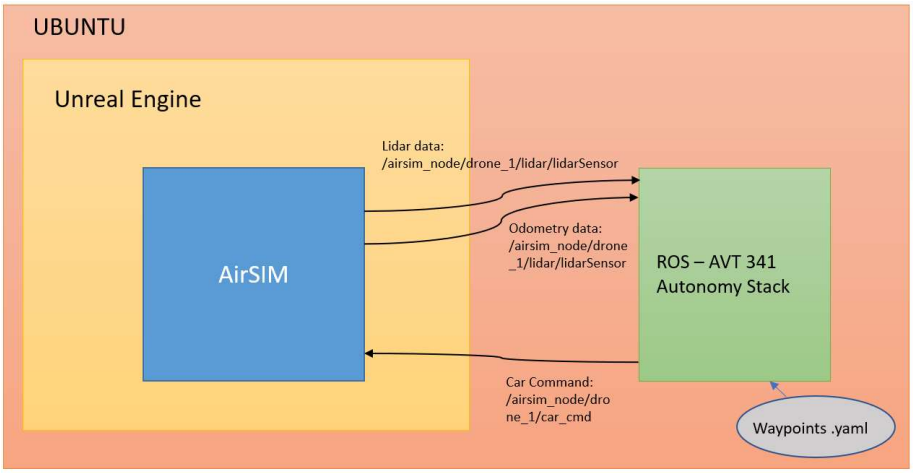


Figure 88 Block diagram showing the data flow between the avt_341 autonomy stack, the Unreal Engine and AirSIM.

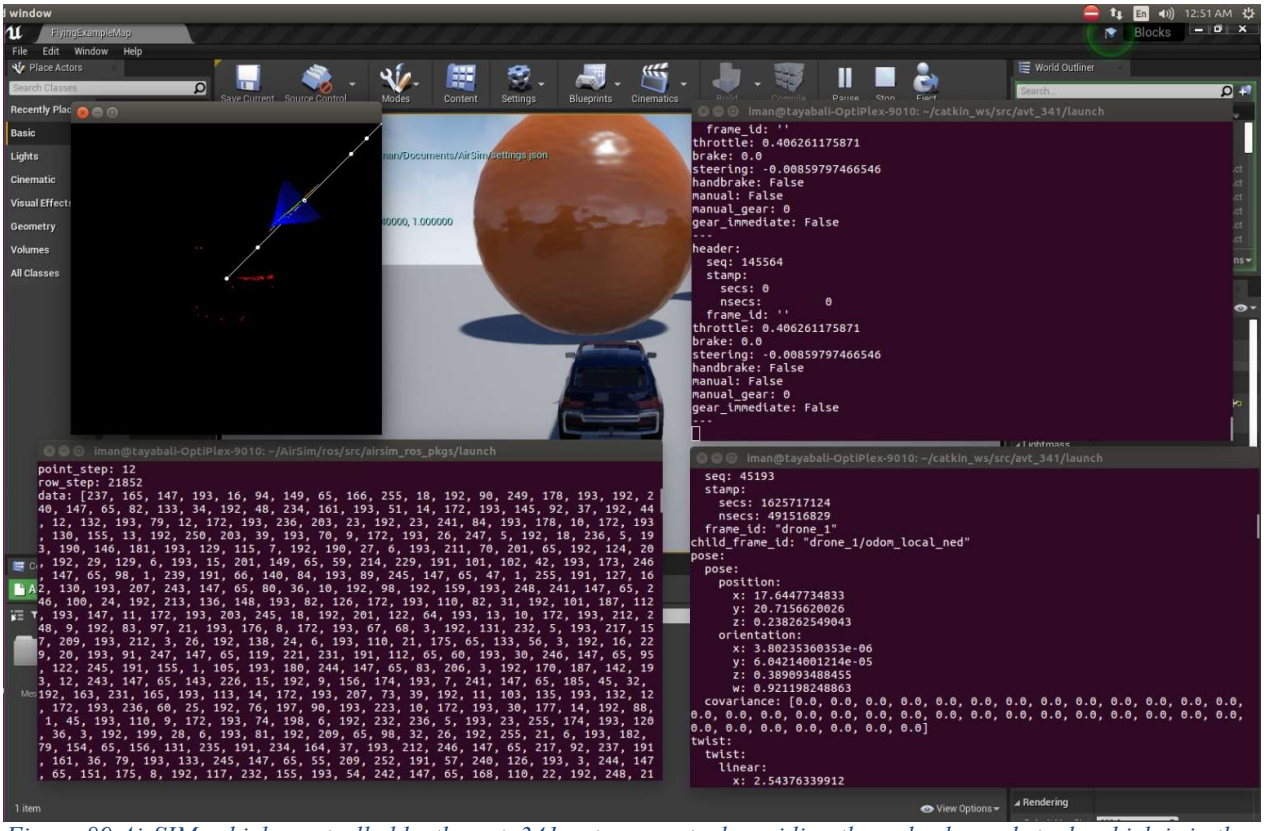


Figure 89 AirSIM vehicle controlled by the avt_341 autonomy stack avoiding the red sphere obstacle which is in the waypoint path of the vehicle. Left top window shows the global planner which shows that the vehicle following the waypoints given. Left bottom window shows the published Lidar data which is fed to the autonomy stack. The right bottom window shows the location of the vehicle sent to the autonomy stack. The right top window shows the car_cmd given to the vehicle in order to avoid the obstacle but still follow the path.

Task 3: Write Phase I Final Report

This task is completed.

3. Project Results

The project results are:

1. Developed a co-simulation Networking interface between Ascience's IVRESS MBD engine and the Unreal visualization engine (2.1.1). The co-simulation interface allows simulating any number of MBD vehicle models using IVRESS and visualizing the simulation in real-time using the Unreal engine. Also, note that this co-simulation interface can also be integrated into any other MBD software and used to co-simulate with the Unreal engine.
2. Created MBD models for wheeled vehicles (FED-Alpha, TracerX, and Polaris MRZR) and tracked vehicles (M113 and Milrem Themis) using the IVRESS engine. The vehicle models were integrated into the Unreal virtual environment (Section 2.1.2). In addition, we created MBD models for the STEPPR2 and the WANDERER walking robots using the IVRESS engine.
3. Developed a realistic virtual environment using Unreal based on the scenarios specified by NATO AVT-341 for the demonstration of autonomous vehicles' mobility simulation tools (Sections 2.1.3 and 2.2). The scenario uses a GIS terrain map of KRC's test site. The virtual environment includes: dynamic agents (vehicles, people, and animals), vegetation (trees, grass and shrubs), natural obstacles (rocks), and man-made obstacles (barriers and structures). The following environmental effects were included in the virtual environment: fog, rain, hail and snow. A dynamic day and night system was incorporated into the virtual environment. In addition, fall and winter versions of the KRC environment were created in order to demonstrate the effects of change of season on the performance of autonomous vehicles.

4. Integrated Lidar, Radar and visual light camera sensors models into the AVSP framework using the Unreal engine (Section 2.3).
5. Incorporated a simple terramechanics model using a Cartesian elevation grid into the AVSP framework (Section 2.4). Note that IVRESS has a complex terramechanics capability where the soil is modeled using the discrete element method (DEM) (Figure 90). Note that IVRESS has a terramechanics co-simulation capability which allows simulating the soil using simple or complex terramechanics in IVRESS and simulating the vehicle using any multibody dynamics code. This capability has been demonstrated using Chrono (https://projectchrono.org/) as the MBD code and IVRESS/DEM as the complex terramechanics code (Figure 91). In addition, this co-simulation capability is also currently being implemented for ADAMS/MBD and IVRESS for complex or simple terramechanics along with a finite element tire model in IVRESS (Figure 92).
6. Incorporated a waypoint controller in conjunction with a leader-follower pursuit controller to control the speed and steering of follower autonomous vehicles (Section 2.5.1).
7. Demonstrated the capabilities of the integrated AVSP system by simulating four autonomous vehicles of various sizes and types performing the AVT-341 traverse scenario in the KRC virtual test site using parallel computing with the vehicles MBD model running on one computer and the virtual environment visualization in Unreal running on another computer (Section 2.5.1).

Figure 90 Snapshot from an IVRESS complex terramechanics DEM simulation of the FED-Alpha vehicle on wet fine grained soil on the KRC terrain.

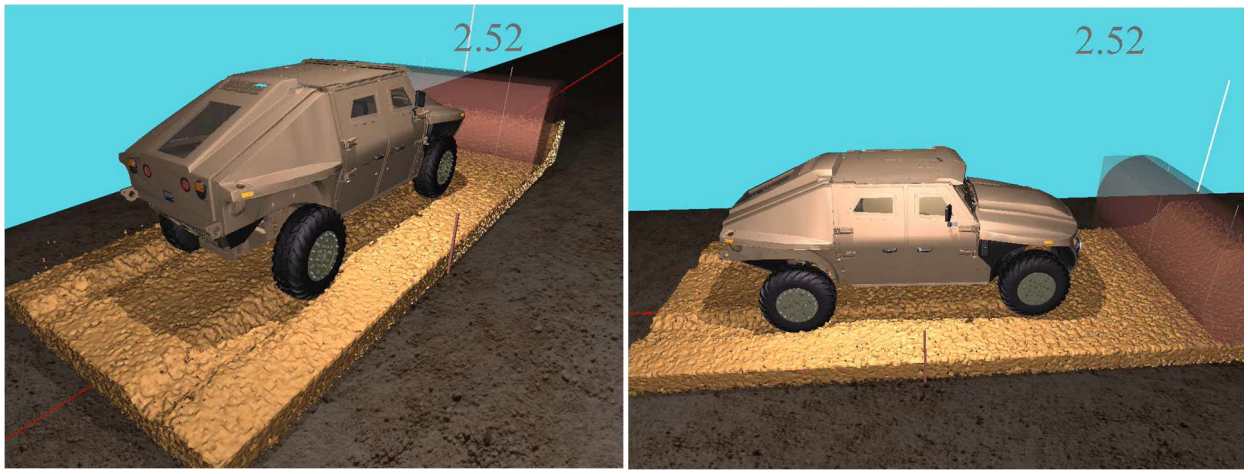


Figure 91 Snapshots from the maximum acceleration/speed Chrono/MBD – IVRESS/DEM co-simulation of the FED-Alpha on dry fine grained soil.

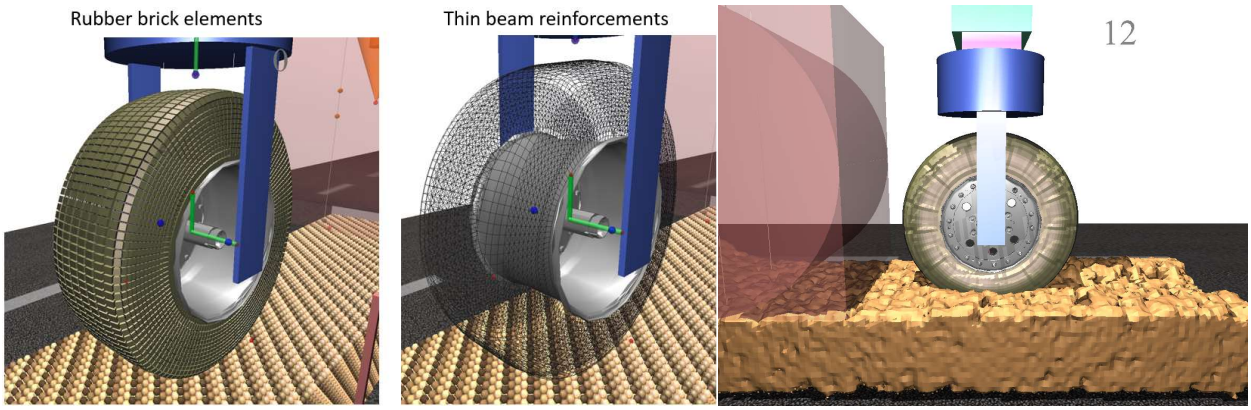


Figure 92 IVRESS finite element tire model on complex terramechanics DEM soil patch.

4. Significant Changes

None

5. Problem Areas

None

6. Project Schedule

Table 3 Phase I project schedule (total duration 7 months). The Δ symbol denotes the completion of a task.

#	Task Month	1	2	3	4	5	6	7	% effort	% ASA	% IUPUI
1	Perform a literature review				Δ				5	50	50
2.1	Develop AVSP prototype							Δ	20	100	0
2.2	Integrate the Unreal engine			Δ					20	55	45
2.3	Integrate sensor models							Δ	20	20	80
2.4	Integrate simple terramechanics					Δ			20	100	0
2.5	Simulate 4 vehicles using parallel computing							Δ	10	70	30
3	Write Phase I report							Δ	5	100	0
									100	69.5	30.5

7. Expenditures

Actual Costs and Progress payments

Total Project Cost	Cumulative Cost	Cumulative Payments	% Spent	% Invoiced
$166,496.14	$166,496.14	$166,496.14	100%	100%

Breakdown of Actual Costs

Cost Item		To Date	
	Rate/hour	Hours	Cost
Direct Labor Costs			
Tamer Wasfy	$90.00	427.8323	$38,504.91
Hatem Wasfy	$60.00	720	$43,200.00
Omar Elmaraghi	$45.00	32	$1,440.00
SubTotal Direct Labor (DL)			**$83,144.91**
Labor Overhead (DL * 7.65%)			$6,360.59
Total Direct Labor (TDL)			**$89,505.49**
Direct Material Costs			
			$0.00
SubTotal Direct Material (DM)			**$0.00**
Material Overhead rate (0%)			$0.00
Total Direct Material (TDM)			**$0.00**
Total Other Direct Costs (TODC)			**$0.00**
G&A (rate 30% x Base (TDL+TDM+TODC)			**$26,851.65**
Sub-Contrract to IUPUI			**$50,139.00**
Total Cost			**$166,496.14**

8. References

1. Unreal Engine. www.unrealengine.com/en-US/
2. Unity. www.unity.com/
3. Ohnishi, R., Hoshino, Y. Position Estimation Using Stereo Camera Images and Physics Engine Simulation for Robot Control. IEEE 2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS). 2018.
4. MathWorks. Stereo Vision for Depth Estimation. www.mathworks.com/discovery/stereo-vision.html
5. ANSYS. Why Autonomous Vehicles Need Thermal Cameras. www.ansys.com/blog/why-autonomous-vehicles-need-thermal-cameras-flir-ces
6. Vasstein, K., Brekke, E.F., Mester, R., Eide, E. Autoferry Gemini: A Real-Time Simulation Platform for Electromagnetic Radiation Sensors on Autonomous Ships. The 3rd International Conference on Maritime Autonomous Surface Ship (ICMASS 2020), 2020.
7. Christie, C.L., Gouthas, E., Williams, O.M., Swierkowski, L. Dynamic Thermal Signature Prediction for Real-Time Scene Generation. Proc. SPIE 8707, Technologies for Synthetic Environments: Hardware-in-the-loop XVIII, July 2013.
8. Schwalbe, E. Geometric modelling and calibration of fisheye lens camera systems. International Archives of the Photogrammetry. Remote Sensing and Spatial Information Sciences, 36, 2012.
9. Toepfer, C., Ehlgen, T. A Unifying Omnidirectional Camera Model and its Applications**.** IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007.
10. Goodin, C., Durst, P.J., Gates, B., Cummins, C., Priddy, J. "High Fidelity Sensor Simulations for he Virtual Autonomous Navigation Environment." Simulation, Modeling, and Programming for Autonomous Robots, pp. 75-86, 2010.
11. Mathworks. Radar Toolbox. https://www.mathworks.com/products/radar.html#radar-data-synthesis
12. ROS. http://wiki.ros.org/ROS/Introduction
13. ROS-M. https://rosmilitary.org
14. Serrano, D. Introduction to JAUS for Unmanned Systems Interoperability. NATO Science and Technology Organization. STO-En-Sci-271.
15. National Highway Traffic Safety Administration (NHTSA). Vehicle-to-Vehicle Communication. https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communication
16. El-Said, M., Mansour, S., Bhuse, V. DSRC Based Sensor-Pooling Protocol for Connected Vehicles in Future Smart Cities. Procedia Computer Science, 140, pp.70-78, 2018.
17. rFpro. http://www.rfpro.com
18. cognata. https://www.cognata.com
19. CARLA. https://carla.org
20. Foretify. https://www.foretellix.com/technology
21. MathWorks.Automated Driving Toolbox. https://www.mathworks.com/products/automated-driving.html
22. ANVEL. https://quantumsignalai.com/anvel
23. Project Chrono - An Open-Source Physics Engine. www.projectchrono.org
24. Simcenter Prescan. https://tass.plm.automation.siemens.com/prescan
25. AirSim. https://github.com/Microsoft/AirSim
26. Ansys VRXPERIENCE. www.ansys.com/products/av-simulation/ansys-vrxperience-driving-simulator
27. NVIDIA DRIVE Constellation. https://developer.nvidia.com/drive/drive-constellation
28. Gazebo. http://gazebosim.org/
29. Goodin, C., Carrillo, J.T., McInnis, D.P., Cummins, C.L., Durst, P.J., Gates, B.Q., Newell, B.S. Unmanned Ground Vehicle Simulation with the Virtual Autonomous Navigation Environment. 2017 International Conference on Military Technologies (ICMT) , Brno, Czech Republic, May 31 – June 2, 2017.
30. Berkeley DeepDrive. https://deepdrive.berkeley.edu/#completed-projects
31. Wong, J.Y., Terramechanics and Off-Road Vehicle Engineering, 2nd Edition. 2010, Oxford, England: Elsevier.
32. Keweenaw Research Center. https://www.mtu.edu/krc
33. NATO AVT-341. “Mobility Assessment Methods and Tools for Autonomous Military Ground Systems” https://www.sto.nato.int/Lists/test1/activitydetails.aspx?ID=16784
34. Carlson Equipment & Software. https://www.carlsones.com/
35. MeshLab. https://www.meshlab.net/
36. AQUAVEO. SMS 13.0 – Surface Water Modeling System. https://www.aquaveo.com/software/sms-surface-water-modeling-system-introduction

37. QGIS – A Free and Open Source Geographic Information System. https://qgis.org/en/site/
38. Ester, Martin, et al. (eds), “A density-based algorithm for discovering clusters in large spatial databases with noise.” Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. CiteSeerX 10.1.1.121.9220. ISBN 1-57735-004-9.