2020 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
MODELING SIMULATION & SOFTWARE TECHNICAL SESSION
AUGUST 11-13, 2020 - NOVI, MICHIGAN

EXPLORING THE REQUIREMENTS AND CAPABILITIES OF OFF-ROAD SIMULATION IN MAVS AND GAZEBO

[1]Marc N. Moore, [1]Payton A. Ray, [1]Christopher Goodin, [1]Christopher R. Hudson, [1]Matthew Doude, [1]Daniel W. Carruth, [2]Mark R. Ewing, Jr., [2]Brent W. Towne

[1]Center for Advanced Vehicular Systems, Mississippi State University, Starkville, MS

[2]U.S. Army Corps of Engineers Engineer Research and Development Center, Vicksburg, MS

ABSTRACT

Simulation is critical to the development of effective unmanned ground vehicles (UGVs). Simulation provides the ability to test virtual hardware and software systems in conditions that may be difficult to recreate physically. An important benefit of simulation is that it grants researchers access to simulated hardware, such as sensors and vehicles, that might not be available otherwise. To successfully simulate both hardware and software systems, it is essential to acknowledge the needs and requirements of the simulation platform. In this paper, we investigate two simulation environments being used at Mississippi State University to model and simulate UGVs: the Mississippi State University Autonomous Vehicle Simulator (MAVS) and Gazebo.

Within this paper we investigate the specific modeling needs for the Clearpath Robotics Warthog UGV in both simulation environments. We found that Gazebo has more options for vehicle and robot customization. However, Gazebo requires more up-front and explicit information to simulate even basic vehicles. MAVS, in contrast, is a platform that uses pre-defined vehicle and tire models that reduce the informational requirements and better supports rapid prototyping of four-wheeled ground vehicles. The narrower scope of MAVS limits its ability to model complex robots, but it excels at vehicle-terrain interaction and sensor simulation. It is fundamental to understand what level of granularity each system offers regarding simulation creation (i.e., how customizable the vehicle, physics, and environment is) to utilize each simulation environment effectively.

Citation: Moore, M.N., Ray, P.A., Hudson, C.R., Goodin, C., Doude, M., Carruth, D.W., Ewing, M.R., & Towne, B.W. (2020). "Exploring the Requirements and Capabilities of Off-Road Simulation in MAVS and Gazebo", In *Proceedings of the Ground Vehicle Systems Engineering and Technology* Symposium (GVSETS), NDIA, Novi, MI, Aug. 13-15, 2020.

1. INTRODUCTION

Simulation is an increasingly important step in the process of creating, testing, and training advanced robotic and vehicular systems. Simulation platforms are currently being used to support the development of autonomous unmanned ground vehicles (UGVs), often intended for military operations. Exploring the capabilities of UGVs in simulation is beneficial in addressing specific challenges that accompany these vehicles, as well as how they navigate and interact with terrain. Some of the challenging scenarios presented when working with UGVs include navigation in unconstrained off-road terrains, navigation in environments with limited (or without) GPS information, human error while operating, and high cost of hardware. Simulation provides a straightforward way to approach these tasks and supports the production of robust, reliable systems.

The two simulation platforms that will be discussed in this paper are the Mississippi State University Autonomous Vehicle Simulator (MAVS) [1] and Gazebo [2]. The goals of this paper are to note the differences between these simulation environments and to highlight their individual strengths. To illustrate the differences in creating and running simulations in both platforms, we describe a model of the Clearpath Warthog and document the necessary steps for both simulators.

One of the benefits of Gazebo is its long development history. Gazebo has been used by many researchers and offers a large library of models, environments, and documentation. Gazebo's 3D model editing tool enables rapid prototyping and real-time feedback. In contrast, MAVS excels at accurate, physics-based sensor simulations. MAVS is capable of rendering photo-realistic outdoor scenes with authentic sensor interaction. Both Gazebo and MAVS provide integrations with the Robot Operating System (ROS). ROS is a widely used framework for developing software for autonomous robots and ground vehicles. By integrating ROS, both MAVS and Gazebo provide interfaces for integrating popular perception, planning, and control algorithms for testing.

2. VEHICLE MODELING & SIMULATION

2.1 Advantages

Safe Testing Conditions. When developing any vehicle, autonomous or not, passenger and pedestrian safety is crucial. For self-driving or autonomous cars, simulation provides a way to thoroughly test things like reaction time in a safe, simulated environment. Self-driving technologies can be evaluated over time by simulating miles driven, or by introducing rare edge cases. In this case, simulating dangerous conditions before real-world implementation is a necessity.

Access to Hardware. Often, research in robotics or ground vehicle systems is limited (or prolonged) by costs associated with hardware and design. Monetary costs are not the only relevant concern – the constraint of time is also a factor. When physically constructing models from the ground up, extra money and time can be spent re-engineering prototypes. Introducing modeling and simulation into the research development cycle manages these issues. Virtual modeling allows design changes to be quick and requires less human involvement. Reverting to previous model instances becomes much more convenient. Simulating a model in action lets the creator instantly observe if modifications function as expected.

Diverse Simulation Scenarios. Setting up testing scenes for vehicles requires an on-site team of humans responsible for building and monitoring the scene, operating the vehicle(s), and running tests. This occupies a significant amount of time which could otherwise be used to create more diverse scenarios in simulated environments. Simulation offers control over scenario parameters that can be difficult or impossible to recreate in real-world testing (i.e., rainfall, dust, movement of vehicles,

etc.) When these tasks become relatively abstracted via simulation, testing can be centered around exploring new and interesting research ideas.

Automatic Generation of Labeled Data. Researchers have used Gazebo to "synthesize automatically labeled 3D point clouds of natural environments" [3]. Specifically, the 3D laser rangefinder on the ground mobile robot "Andabata" [4] was emulated within Gazebo. The paper addressed the relevance of scene classification, which enables ground robots to autonomously navigate natural environments. Throughout the literature datasets containing 3D scans of terrain elements (i.e., ground, vegetation) can be found – most of which have already been manually or interactively annotated in software. The authors noted that few labeled datasets exist for ground robots in natural environments. Using simulation, the researchers generated realistic 3D point clouds by assigning arbitrary reflectivity values to environment elements.

Research has also been done to automate the process of collecting and labelling training data for convolutional neural networks (CNNs) using MAVS [5]. It was found that the simulated data generated could be useful for training CNNs to segment image or camera data in outdoor environments with limited error. Prior to running simulations, all objects within a scene were semantically labeled. In MAVS, they generated random terrain surfaces, three different ecosystem types, three different sensor types, and automatically labeled the training and LiDAR data. If humans were to manually label the data described above, it could take countless hours.

Validation of Results. When simulated models accurately represent real-world models, simulation can provide an effective tool for research, development, and testing of UGVs. Confidence in the results of simulation depends on verification and validation of the models that comprise the simulation framework. This requires verifying that the specifications meet the needs of the model or system user and validating that the model or system's output matches what is intended. It is important to recognize that the transition from simulation to real-world (sim to real transfer) is rarely one-to-one. There are limitations in sim to real transfer in areas like Reinforcement Learning [6] and in tasks like 3D human pose estimation [7].

3. MODELING PLATFORMS

3.1 Gazebo

Gazebo is a popular open-source 3D robotics simulation platform. From 2004-2011, Gazebo was a contributor to the Player Project [8], founded to encourage research efforts in robotics and sensor systems through the use of free software– specifically simulation platforms. As a result, client/server robot control interface Player, 2D robot simulator Stage, and 3D robot simulator Gazebo were all developed. By 2011, Gazebo had transitioned into an independent project. Gazebo was created for the purpose of rapidly testing algorithms, designing complex robots, simulating, training systems within realistic scenarios, and more.

To accurately and flexibly render simulation components and their physical features, Gazebo uses two different XML file formats: Unified Robotic Description Format (URDF) and Simulation Description Format (SDF). Files specifying a given robot or vehicle's visual, structural, and kinematic properties (i.e., links, joints, sensors, etc.) are generally written in URDF. While URDFs are useful and standardized in some robotics applications (e.g., NVIDIA Isaac Sim [9], MathWorks Simscape Multibody [10]), URDFs only define a robot in isolation and cannot specify the pose of that robot within a world. Originally designed to address the shortcomings of URDFs within Gazebo [11], SDF was introduced. Typically, files written in SDF are used for controlling and visualizing movement at the robot/vehicle level, or describing elements like

terrain, agents, or static and dynamic objects at the world level.

Four diverse, high-performance physics engines are available through Gazebo's physics API: Open Dynamics Engine (ODE) [12], Bullet [13], Simbody from Stanford University [14], and Dynamic Animation and Robotics Toolkit (DART) from Georgia Tech [15]. Twenty-five sensor classes [16] are defined for Gazebo, including common sensors for autonomous robotic and vehicular systems like LiDAR and GPS. Gazebo is capable of emulating realistic 3D scenes using Object-Oriented Graphics Rendering Engine (OGRE) [17]. User interaction with Gazebo can be achieved through its native 3D graphic editor or by modifying a file's code directly.

ROS is commonly used as a direct communication, planning, and control interface with Gazebo. ROS is a widely used framework for developing software and implementing real-time control of simulated robots and vehicles. Communication with ROS is best accomplished by including additional simulation-specific tags within URDF files as well as installing Gazebo plug-ins and dependencies. This enables seamless control via the *ros_control* [18] packages. The *ros_control* set of packages includes controller interfaces, controller managers, transmissions and interfaces for hardware. Gazebo simulations are ultimately launched through compatible *roslaunch* [19] files.

3.2 MAVS

MAVS is a collection of software tools and libraries used for realistic on-road and off-road vehicle simulations. MAVS has been in development for three years and was started to address the shortcomings of existing simulators to interactively simulate autonomous navigation in complex off-road terrain. MAVS leverages Intel's Embree platform [20]: a collection of ray tracing kernels optimized for CPUs. Embree allows MAVS to support in-depth physics models for LiDAR and camera systems to produce accurate physics-based sensor data.

In [1], the authors showcased the detail given to simulated sensors by accurately modeling the behavior of LiDAR using ray tracing and validating their results against controlled field tests, analytical models, and laboratory results. This work demonstrated MAVS' ability to simulate LiDAR in complex environments in real-time.

MAVS additionally provides resources for automatic terrain generation for off-road autonomy, enabling rapid testing in a large set of unique environments. Since real-world autonomous systems must contend with adverse weather conditions, MAVS provides realistic simulated weather environments. Environment details such as fog, snow, clouds, wind, and time of day are all adjustable through its user interface. It uses ReactPhysics3D to model vehicle physics; however, it is compatible with other vehicle dynamics models including Chrono [21].

Another notable feature of MAVS is its robust vehicle-terrain interaction (VTI) model. MAVS represents vehicles using a multibody dynamics model which allows multiple independent forces to be calculated for each component of the vehicle. This enables realistic simulations of vehicle behavior on different surfaces. Currently, MAVS has implemented equations to model tire interactions on six different surfaces: wet and dry pavement, fine- and coarse-grained soil, snow, and ice.

MAVS is written in C++ and has an optional Python wrapper for ease of use. Geometric and physical descriptions for vehicles and terrain are specified in JSON files. The input JSON file contains features such as wheel offsets and chassis dimension that are used by the RP3D engine. The MAVS coordinate system follows an East-North-Up scheme. The positive x direction is east, the positive y direction is north, the positive z direction points upward, and the default length unit is in meters. Currently, the software is available at no cost to non-commercial users.

3.3 Simulation Use Cases

A. Gazebo

Gazebo has been used as a simulation tool for a variety of scientific robotic applications. Okayama University of Science researchers [22] show that autonomous navigation algorithms performed similarly in both real-world and simulated environments. The experiment was conducted using two mobile robots: Pioneer 3-DX [23], a small two-wheel two-motor differential robot, and PeopleBot [24], a differential robot designed for service/human-interface tasks. Their results demonstrate that simulation-based code developed using ROS and Gazebo can be deployed to real-world scenarios without modification.

Researchers from Innopolis University and Kazan Federal University [25] modeled the Russian crawler-type UGV "Engineer" using ROS and Gazebo. Their research described the complexity of modeling, animating, and simulating UGVs and approximating track-terrain interaction. Despite the challenges, the model succeeded at mirroring the movement and physics of the real Engineer robot. It also supported both crawler or "caterpillar" locomotion and upper manipulator control.

In an effective display of Gazebo's range of simulation environments, a plug-in was created to model unmanned underwater vehicles and structures [26]. Submersible turbines and sensors were modeled that react to the environment using hydrodynamic and hydrostatic force simulations. Gazebo's ability to model vehicles with multiple degrees of freedom has been extended to simulate UAVs as well [27].

B. MAVS

MAVS is a relatively new option for high-performance ground vehicle simulation. In one of the earliest papers about the platform [1], the authors addressed the problem of realistically simulating LiDAR and its interaction with vegetation. The use of LiDAR in on-road autonomous vehicles is well established, but unique challenges present themselves when applying these techniques to unstructured, natural environments. One issue that remains unaddressed due to the lack of adequate simulations is the failure of LiDAR to accurately distinguish between obstacles like trees or concrete, and objects that can be easily traversed like grass or low vegetation. In this work the authors presented a statistical method for modeling LiDAR returns from grass; a common scenario in real-world, off-road autonomous vehicle development that was underserved in current simulation platforms. The authors point to the three requirements for accurate, physics-based LiDAR simulation: Beam divergence and beam shape, modeling the light-scattering properties of vegetation, and on-board signal processing.

4. VEHICLE IMPLEMENTATION

4.1 Vehicle Data

A. Gazebo

The foundation for every Gazebo simulation is the world file. As mentioned in previous sections, files which describe elements at the world-level are written in SDF. World files are indicated with a *.world* extension and contain all elements involved in a simulation. These elements are items such as robots, sensors, objects, agents, etc. as well as global parameters like the sky, ambient light, and physics properties. Some world elements are marked as static, meaning they only possess collision geometry (or geometry relative to the interaction of one object with another). Static encompasses all objects which are not meant to move within the simulated world environment. Properly labeling these entities as so ensures that non-moving objects do not have unnecessary performance defects on the simulation. Correspondingly, there are world elements marked as dynamic. This is specified by either setting the *<static>* element to false in an SDF file or omitting the element entirely. Dynamic objects possess inertia in additional to collision geometry, distinguishing them from static objects.

Model files are another key component required to successfully run Gazebo simulations. Like world files, they are written in the SDF XML file format. The purpose of model files is to both simplify world files and ease model reuse in general. Gazebo model files can be provided from the online model database [28], shipped as example models with Gazebo (in previous versions), shared amongst the online community of users, or created using Gazebo's model editor or other 3D modeling software (i.e., Blender [29]).

To better understand how models are constructed in Gazebo, Table 1 provides an overview of the components of an *SDF Model Object*, which refers to the <*model*> tag in an SDF file. Each model has a collection of *Links*, *Joints*, *Visuals*, *Collision* objects, *Inertial* and *Sensor* properties, and *Plugins* for controlling the model itself [30].

Table 1: Components of SDF Models.

Variable	Description
Links	Physical link for one body in model (i.e., wheel) with collision, visual, and inertial properties
Collision	Collision properties of a link; Contains the geometry for collision checking, usually a simple shape or triangle mesh
Visual	0 or more visual properties of a link; Specifies the shape (i.e., cylinder) of an object
Inertial	Inertial/dynamic properties of a link (i.e., mass)
Sensor	0 or more sensors that collect data from world for plug-in use
Light	0 or more light sources attached to a link
Joints	Connects two links that have kinematic and dynamic properties
Plugins	Third-party libraries which control models

The ordering of variables listed above in Table 1 is the suggested order in which features should be added to an SDF model file – from least to most complex.

B. MAVS

Every MAVS simulation is defined by the core MAVS class. This class has methods and member variables for setting every simulation parameter such as the vehicle, physics engine, sensors, etc. The context of simulation environments is defined in the Environment member of the MAVS class. Features like light and weather are managed by the Environment. Environmental features like Rain, turbidity, albedo, fog density, cloud-cover fraction, snowfall rate, and wind speed can be set by calling the member functions of the environment class.

A scene is a component of the MAVS environment and must be created and added to the environment. Whereas Gazebo combines physical properties and the ambient properties in the world file, MAVS separates these into two classes, the scene and environment. The scene is defined as a series of meshes which describe the polygonal objects within the scene as well as the terrain and terrain features. Random scenes can be created automatically, or a scene can be generated using a description file. Scene variables can be described via the text file and can also be modified using Python commands. Features such as potholes and terrain roughness can also be defined for the scene.

To implement a vehicle model using RP3D, MAVS requires the vehicle specifications be defined using via a JSON file. The JSON must contain five components that are defined using

forty-five variables. The first structure, *Chassis*, is defined via the *Sprung Mass*, the *Center of Gravity Offset*, and the *Dimensions* per Table 2.

Table 2: Chassis Parameters.

Variable	Description
Center of Gravity Offset	The offset in meters from the lower plane of the chassis
Dimensions	The length, width, and height of the chassis in meters
Sprung Mass	The portion of the vehicle mass supported by the suspension

The suspension, or *Axles*, are defined by nine variables for each axle.

Table 3: Suspension Parameters.

Variable	Description
Longitudinal Offset	Offset from center of gravity, positive and negative for front and rear axle, respectively
Track Width	Distance between center of each tire
Spring Constant	Required for the linear spring-damper model used by the suspension
Damping Constant	
Spring Length	
Steered/Powered	Boolean
Unsprung Mass	The portion of the vehicle mass not supported by the suspension
Max Steer Angle	The angle in both directions a tire can turn

The tire specifications are a subset of the *Axles* structure and require five variables. Both tires on the axle are given the same values, *Spring Constant, Damping Constant, Radius, Width,* and *High Slip Crossover Angle*.

Table 4: Tire Parameters.

Variable	Description
Spring Constant	Tires are also modeled as an independent spring-damper system, requiring these constants
Damping Constant	
Radius	Tire radius in meters
High Slip Crossover Angle	Used by the Crolla Model [31] to calculate net lateral traction
Width	Tire width in meters

The final list of values, *Initial Pose*, contains two variables, *Position* and *Orientation*. *Position* is a three-element list containing the desired origin in Cartesian coordinates of the simulated vehicle. *Orientation* is a four-element quaternion describing the rotation of the vehicle in 3D space.

5. VEHICLE-TERRAIN INTERACTION

As mentioned previously, a motivation for the development of MAVS was the lack of simulators that could adequately simulate both sensors and terrain for off-road autonomous vehicles. To address limitations in representations of terrain, a robust vehicle-terrain interaction model was developed within MAVS. However, we are unaware of any publications that describe a similar model for Gazebo. Although there are multiple options for physics engines, ODE, the default physics engine for Gazebo, does not seem to pay special attention to modeling tire and surface interactions. The friction and contact model used by ODE is based on an efficient implementation of the Dantzig LCP solver [32] but it is unclear how that is used to implement VTI.

To better simulate vehicle-terrain interaction, particularly off-road terrain, MAVS implements multiple equations that accurately model tires and a variety of surfaces. The MAVS VTI model is

iterative and calculates all the relevant forces in six phases for each discrete timestep. The multibody dynamics (MBD) model introduced in the previous sections describes vehicle behavior such as orientation, position, and velocity for each time step. These variables are used by the VTI model to calculate the torque and forces to be applied to the vehicle through the hub of the wheel which is connected to the chassis by a slider joint and spring-damper system.

5.1 Wheel Velocities

The first step in the VTI calculation is calculating the wheel velocities based on the global tire frame. The MBD model describes tire velocity using the global world coordinates and an orientation matrix. Using the tire velocity from the global coordinates, ($\vec{v}_t$), and the Look-To vector, $\overrightarrow{LT}(t)$, the longitudinal velocity at timestep t, $v_{\parallel}(t)$, is calculated with

$$v_{\parallel}(t) = \overrightarrow{LT}(t) \cdot \vec{v}_t(t)$$

Where the longitudinal velocity is a portion of the total tire velocity that is directed forward. Likewise, the lateral velocity is found using the Look-side vector. This is the velocity perpendicular to the wheel hub if the tires were straight, calculating the side-to-side velocity at time t with

$$v_{\perp}(t) = \overrightarrow{LS}(t) \cdot \vec{v}_t(t)$$

And finally, the vertical velocity of the tires is found using the Look-Up vector with

$$v_{\uparrow}(t) = \overrightarrow{LU}(t) \cdot \vec{v}_t(t)$$

The tire velocity based on the tire frame can therefore be defined as the sum of these products, producing the new reference frame to calculate VTI with

$$\vec{v}_t(t) = \overrightarrow{LT}(t)v_{\parallel}(t) + \overrightarrow{LS}(t)v_{\perp}(t) + \overrightarrow{LU}(t)\, v_{\uparrow}(t)$$

5.2 Normal Forces

The second step in calculating the VTI model is to derive the normal forces and tire deflection. This will be used in the longitudinal and lateral force calculation. Using the coordinate of the tire at the current time step, $\vec{p}_t = [p_x, p_y, p_z]$, the terrain height at the same point, $Z(x, y)$, and the tire diameter, d, the tire deflection, $\delta(t)$, is calculated as

$$\delta(t) = Z(p_x, p_y) + \frac{d}{2} - p_z$$

Using the tire spring coefficient, k¸ the damping coefficient, c, and the vertical tire velocity from the previous step, the normal force $N(t)$, is calculated with

$$N(T) = k\delta(t) - c\, v_{\uparrow}(t)$$

5.3 Slip and Slip Angle

To calculate the effective radius, tire deflection must be taking into consideration. By subtracting tire deflection from the undeflected radius, the effective radius is given by

$$r_{eff} = \frac{d}{2} - \delta(t)$$

To calculate the tire slip, $s(t)$, using the longitudinal and angular velocity the following piece-wise equation is implemented:

$$s(t) = \begin{cases} \dfrac{r\omega_t}{v_{\parallel}} - 1 & v_{\parallel} < r\omega_t \text{ and } v_{\parallel} \neq 0 \\ 1 - \dfrac{v_{\parallel}}{r\omega_t} & v_{\parallel} > r\omega_t \text{ and } r\omega_t \neq 0 \\ 0 & otherwise \end{cases}$$

To calculate the slip angle at time step t, $\alpha(t)$, the steering angle, $\theta(t)$, is used in the following equation.

$$\alpha = \tan^{-1}\frac{v_{\perp}(t)}{|v_{\parallel}(t)|} - \theta(t)$$

The first step in the VTI calculation is calculating the individual wheel velocities based on the global tire frame. The MBD model describes tire velocity using the global world coordinates and an orientation matrix. To calculate the forces acting on the tires, these need to be converted to velocities relative to the tire frame rather than the global frame. Using the tire velocity from the global coordinates, ($\vec{v}_t$), and the Look-To vector, $\overrightarrow{LT}(t)$, the longitudinal velocity at timestep t, $v_{\parallel}(t)$, is calculated with

5.4 VTI Forces

Using the previous steps, the force exerted on the tire by the terrain is modeled with the following VTI equation. The inputs include the normal force, the tire slip, tire slip angle, and tire deflection.

$$F^{vti} = \left[F_{\parallel}^{vti}, F_{\perp}^{vti}\right] = f(N, \delta, s, \alpha)$$

5.5 Wheel Angular Velocity

The wheel dynamics and the VTI are treated independently in MAVS. After the VTI forces are updated, the wheel angular velocity is calculated at each time step according to the equation

$$\omega_t(t + \delta t) = \omega_t(t) + \frac{dt}{I_t}\left(Q(t) - \frac{d}{2}F_{\parallel} - \beta\omega_t(t)\right)$$

Where β is the viscous friction coefficient of the tire, $Q(t)$ is the applied torque from the driveline, I_t is the moment of inertia of the tire, and $F_{\parallel}$ is the longitudinal net traction calculated from the VTI. The angular velocity calculated at this time step will be used in the following time step to calculate the wheel slip.

5.6 Global Frame Forces

The 3D force applied to the wheel hub, in global coordinates, is given by

$$F_{global} = F_{\parallel}^{vti}\overrightarrow{LT} + F_{\perp}^{vti}\overrightarrow{LS} + N\overrightarrow{LU}$$

This force is applied when the tire and VTI model completes. The vehicle and driveline models then perform their update steps and re-initiate the tire model with the updated tire position, velocity, angular velocity, and applies torque.

6. WARTHOG IMPLEMENTATION

The vehicle selected for this implementation, the Clearpath Robotics Warthog [33], is described as "ROS Ready" by the manufacturer and has simulation and modeling files provided and maintained on GitHub [34]. Figure 1 shows a Warthog in the field. The files provided include a URDF file that describes the behavior of the model as well as the physical specifications like size and weight. Additionally, an object file is provided that specifies the dimensions used by the visual representation. As this was provided by the manufacturer, the details are assumed to be accurate and reliable.

Figure 1: Clearpath promotional image of Warthog with sensor attachments.

These simulation files are intended to be used with the ROS and Gazebo platforms and did not include certain details required by MAVS. MAVS is designed with common ground vehicles as its focus and thus lacks the ability to simulate some mechanisms more often used in robotics. The rigid multibody dynamic model used by MAVS prevents it from simulating the geometric passive articulation that allows the left and right halves of

the Warthog to pivot independently. Since the Warthog does not have a suspension like an on-road, production vehicle, certain assumptions were made to model it in MAVS.

The weight of the Warthog provided in the datasheet was used for the Sprung Mass in the vehicle description JSON for MAVS. Normally, this would only be the weight of the vehicle components supported by the suspension and would not include the weight of the wheels or the entire weight of the suspension itself. The maximum engine torque was also absent from modeling files and was estimated using information provided in datasheets.

To get details about the axle and tire placement, the object file was opened in a 3D viewer provided by MAVS. By viewing the model provided by Clearpath in the viewer, the tire width, radius, axle offsets, and center of gravity were measured. Additionally, the visual model of the Warthog was offset from the physical model and needed to be centered. Once the model corrections were made the vehicle was simulated within MAVS. Figure 2 shows the completed vehicle in a forest environment.

Figure 2: MAVS Warthog model in example off-road forest environment.

For Gazebo, the process was much simpler and straightforward. Using a fresh installation of Ubuntu 16.04, the process of downloading, installing, and running the Warthog simulation took only a few minutes. No modification to the setup or simulation files were needed. Figure 3 below shows the Warthog model launched in Gazebo inside of an example urban environment created by Clearpath [34].

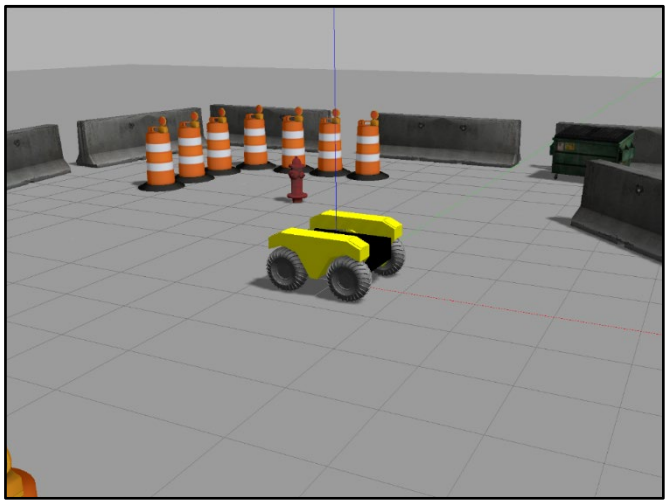

Figure 3: Gazebo Warthog model in example urban environment.

7. CONCLUSIONS

In this paper, we explored both the differences and capabilities of two 3D simulation environments, MAVS and Gazebo, by describing the minimum informational requirements needed for simulation and vehicle setup. To further illustrate this, we implemented Clearpath Robotics' Warthog UGV in each platform and made necessary adaptations to the model to faithfully match the virtual vehicle to the physical vehicle. Research use cases for MAVS and Gazebo were discussed to highlight the potential for robot and ground vehicle creation, testing, and simulation customization relative to each environment. Additionally, the fidelity of both physics models were discussed—paying close attention to the vehicle-terrain interactions. This knowledge was shared to facilitate and encourage the use of modeling and simulation of UGVs in MAVS and Gazebo.

Future work will be done to test and validate the accuracy of our simulated results by comparing them to real-world implementations using the Warthog vehicle.

8. REFERENCES

[1]C. Goodin, M. Doude, C. Hudson and D. Carruth, "Enabling Off-Road Autonomous Navigation-Simulation of LIDAR in Dense Vegetation", *Electronics*, vol. 7, no. 9, p. 154, 2018. Available: 10.3390/electronics7090154 [Accessed 18 May 2020].

[2]Open Source Robotics Foundation. (2020) Gazebo [Online]. Available: http://gazebosim.org/.

[3]M. Sanchez, J. Martinez, J. Morales, A. Robles and M. Moran, "Automatic Generation of Labeled 3D Point Clouds of Natural Environments with Gazebo", *2019 IEEE International Conference on Mechatronics (ICM)*, 2019. Available: 10.1109/icmech.2019.8722866 [Accessed 18 May 2020].

[4]J. L. Martínez, M. Morán, J. Morales, A. J. Reina and M. Zafra, "Field navigation using fuzzy elevation maps built with local 3D laser scans", *Applied Sciences*, vol. 8, no. 397, pp. 1-18, 201.

[5]C. Goodin, S. Sharma, M. Doude, D. Carruth, L. Dabbiru and C. Hudson, "Training of Neural Networks with Automated Labeling of Simulated Sensor Data", *SAE Technical Paper Series*, 2019. Available: 10.4271/2019-01-0120 [Accessed 18 May 2020].

[6]Kaspar, Manuel & Muñoz Osorio, Juan & Bock, Juergen. (2020). Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization.

[7]Doersch, C., & Zisserman, A. (2019). Sim2real transfer learning for 3D pose estimation: motion to the rescue. *ArXiv, abs/1907.02499.*

[8] Player/Stage project. (2020) The Player Project [Online]. Available: http://playerstage.sourceforge.net/.

[9]NVIDIA Corporation (2020). NVIDIA Isaac Sim [Online]. Available: https://developer.nvidia.com/isaac-sim.

[10]The MathWorks, Inc. (2020). Simscape Multibody [Online]. Available: https://www.mathworks.com/products/simmechanics.html.

[11]Open Source Robotics Foundation. (2020) Tutorial: Using a URDF in Gazebo [Online]. Available: http://gazebosim.org/tutorials/?tut=ros_urdf.

[12]Russell L. Smith. (2020) Open Dynamics Engine [Online]. Available: http://ode.org/wiki/index.php?title=Main_Page.

[13]Real-Time Physics Simulation. (2020) BULLET Physics Library [Online]. Available: https://pybullet.org/wordpress/.

[14](2020) Simbody: Multibody Physics APL [Online]. Available: http://simtk.org/projects/simbody/.

[15]J. Lee et al., "DART: Dynamic Animation and Robotics Toolkit", *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018. Available: 10.21105/joss.00500 [Accessed 18 May 2020].

[16]Gazebo API. (2020) Sensors [Online]. Available: https://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/group__gazebo__sensors.html.

[17]OGRE3D. (2020) Object-Oriented Graphics Rendering Engine [Online]. Available: https://www.ogre3d.org/.

[18]ROS.org. (2020) ros_control Package Summary [Online]. Available: http://wiki.ros.org/ros_control.

[19]ROS.org. (2020) roslaunch Package Summary [Online]. Available: http://wiki.ros.org/roslaunch.

[20]Intel Corporation. (2009-2020) Intel Embree – High Performance Ray Tracing Kernels [Online]. Available: https://www.embree.org/.

[21]Tasora, A., Serban, R., Mazhar, H., Pazouki, A., Melanz, D., Fleischmann, J., ... & Negrut, D. (2015, May). Chrono: An open source multi-physics dynamics engine. In *International Conference on High Performance Computing in Science and Engineering* (pp. 19-49). Springer, Cham.

[22]K. Takaya, T. Asai, V. Kroumov and F. Smarandache, "Simulation environment for

mobile robots testing using ROS and Gazebo," 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, 2016, pp. 96-101.

[23]Adept Technology, Inc. (2011) Pioneer 3-DX [Online]. Available: https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf.

[24]Adept Technology, Inc. (2011) PeopleBot [Online]. Available: http://www.generationrobots.com/media/PeopleBot-PPLB-RevA.pdf

[25]M. Sokolov, R. Lavrenov, A. Gabdullin, I. Afanasyev and E. Magid, "3D modelling and simulation of a crawler robot in ROS/Gazebo", *Proceedings of the 4th International Conference on Control, Mechatronics and Automation - ICCMA '16*, 2016. Available: 10.1145/3029610.3029641 [Accessed 18 May 2020].

[26]M. Manhaes, S. Scherer, M. Voss, L. Douat and T. Rauschenbach, "UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation", *OCEANS 2016 MTS/IEEE Monterey*, 2016. Available: 10.1109/oceans.2016.7761080 [Accessed 4 June 2020].

[27]M. Zhang et al., "A high fidelity simulator for a quadrotor UAV using ROS and Gazebo", *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, 2015. Available: 10.1109/iecon.2015.7392534 [Accessed 5 June 2020].

[28]Github, Inc. (2020) gazebo_models Model Database [Online]. Available: https://github.com/osrf/gazebo_models.

[29]The Blender Foundation. (2020) Blender [Online]. Available: https://www.blender.org/.

[30]Open Source Robotics Foundation. (2014) Gazebo Components [Online]. Available: http://gazebosim.org/tutorials?tut=components&cat=get_started.

[31]M. Parker, S. Shoop, B. Coutermarsh, K. Wesson and J. Stanley, "Verification and validation of a winter driving simulator", *Journal of Terramechanics*, vol. 46, no. 4, pp. 127-139, 2009. Available: 10.1016/j.jterra.2009.05.002 [Accessed 18 May 2020].

[32]D. Baraff, "Fast contact force computation for nonpenetrating rigid bodies", *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, 1994. Available: 10.1145/192161.192168 [Accessed 3 June 2020].

[33]Clearpath Robotics, Inc. (2020) Warthog [Online]. Available: https://clearpathrobotics.com/warthog-unmanned-ground-vehicle-robot/.

[34]Github, Inc. (2020) Common packages for Warthog [Online]. Available: https://github.com/warthog-cpr/warthog.