2017 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
AUTONOMOUS GROUND SYSTEMS (AGS) TECHNICAL SESSION
AUGUST 8-10, 2017 - NOVI, MICHIGAN

LEVERAGING HIGH-FIDELITY SIMULATION TO EVALUATE AUTONOMY ALGORITHM SAFETY

Ryan Penning
James English
Daniel Melanz
Brett Limone
Energid Technologies
Cambridge, MA

Paul Muench, PhD
David Bednarz, PhD
US Army TARDEC
Warren, MI

ABSTRACT

The age of large autonomous ground vehicles has arrived. Wherever vehicles are used, autonomy is desired and, in most cases, being studied and developed. The last barrier is to prove to decision makers (and the general public) that these autonomous systems are safe. This paper describes a rigorous safety testing environment for large autonomous vehicles. Our approach to this borrows elements from game theory, where multiple competing players each attempt to maximize their payout. With this construct, we can model an environment that as an agent that seeks poor performance in an effort to find the rare corner cases that can lead to automation failure.

INTRODUCTION

The age of large autonomous ground vehicles has arrived. More autonomy is desired wherever vehicles are used in repetitive, dirty, and dangerous tasks. Vehicle autonomy is actively being researched and developed by many large commercial and government entities. John Deere makes automated harvesters, Caterpillar makes automated mining trucks, Google makes self-driving passenger vehicles, and the Army makes automated transport vehicles. In all this effort, arguably the greatest near-term value of this technology lies with the military, as automating military logistics vehicles will both increase productivity and keep Americans away from combat risks. The military's needs correspond to what is likely to be the first transformative fielding of large-vehicle autonomy.

PROBLEM DESCRIPTION

Now, autonomous large vehicles are carefully watched and guided. John Deere's harvesters have monitors in the cab, Caterpillar restricts application, Google has passive drivers, and the Army tests vehicles in cleared and walled areas. The disappointing lack of fielding in many domains comes from concerns over safety. The autonomy algorithms are not sufficiently robust, and the consequences of failure—a runaway multi-ton vehicle—too high.

Researchers are trying to address the worry over safety through hardware testing. Systems are run through exercises and studies in the field. But these are often seen as inadequate. Funding is (always) limited, and it is impossible to reproduce the great variety of events that a fielded autonomous system will encounter over its life. The main obstacle to acceptance is uncertainty in how the systems will react to untested conditions. As a result, the tests do not convince safety

regulatory boards and the autonomy technology languishes unfielded.

The best and only answer to this problem lies in digital simulation. Simulation can support many more hours of testing in a greater variety of situations at a lower cost. It scales to more and larger computers, runs day and night, and is 100% safe. Digital simulation can be combined with hardware components (a hardware-in-the-loop approach) to add realism and incorporate proprietary algorithms (such as those in sensors and autopilots). A large body of research supports sensor simulation, including for cameras, lidar, GPS, and radar, as well as simulation of vehicles and terrain. Many new sensor modeling techniques leverage GPU programming to speed calculation and quality. And parallel implementation across multiple computers, each with multiple graphics cards, has created a faster computing environment for simulation.

Digital simulation, though, carries two pressing issues. It has in the past offered poorer fidelity than hardware testing, with results that did not carry the engineering or psychological weight of field tests. Simulation also has not, through parametric and Monte Carlo studies, offered adequate coverage of the situations and environments that lead to failure. The corner cases that fail, in the real world, appear as vanishing probability events in most simulation models. This is due significantly to inaccurate probability distributions being applied to Monte Carlo simulation parameters.

The problem has deep roots, as many probability distributions are not even knowable. They may depend on unstudied phenomena, human activities, or one-time events. Variables not even having probability distributions are regarded as *profoundly unknown*. It is the neglect of profoundly unknown variables that drives many simulations to discredit and disuse—they do not provide results that are accurate enough, based on wise assessment from subject matter experts, to support mission planning and execution.

Yet the problem runs even deeper than this. Often in real-world situations there are multiple agents involved, each attempting to maximize their own set of metrics and produce the outcome most desirable to them. The optimal behavior of each agent is dependent on its competitors' behavior. The resulting problem can be staggeringly complex: multiple independent systems, each adapting to the others' behavior and to the environment, with system dynamics, sensors, control algorithms and other factors—some profoundly unknown—all determining the resulting system performance and safety. These effects compound to render the problem intractable (and prohibitively expensive) to solve via traditional hardware testing. A better way must be found.

PREVIOUS WORK

Basic Autonomy Validation

Currently, there are a number of different approaches to validating autonomy algorithms. Traditionally, and most simplistically, there is the standard test matrix approach. This approach sets up and tests combinations of conditions and behaviors in an attempt to determine what produces a system failure. This approach is reasonable for systems with a small number of discrete variables. However, as the number of variables grows, the number of possible combinations grows exponentially, and the problem becomes intractable. This is exacerbated by the introduction of continuous variables that can take any value. Many researchers have recognized these shortcomings, and have devised alternative means of validation that seek out failures. Schultz *et al.* proposed using genetic algorithms to attempt to find failure modes [1]. This transformed evaluation into an optimization problem that attempts to identify failure modes by maximizing a usefulness metric. Similarly, Wegener and Bühler used an evolutionary algorithm to evaluate an autonomous parking system [2].

Game Theory in Robotics

A related body of work also currently exists that uses game theoretic constructs in the control of cooperative robot teams. Emery-Montemerlo *et al.* posed a robotic soccer team as a partially observable stochastic game to determine the best actions for each individual player to take, given the uncertainty surrounding the roles of the opposing players. They used the partial observability approach to logically reason about uncertainty in the world state, in determining these actions [3]. Vidal *et al.* utilized game theory in a two team pursuit problem, in which a team attempts to evade pursuit by an opposing force of unmanned ground and aerial vehicles [4]. In this, the pursuing team has no knowledge of the pursuit area, and must simultaneously reason about the environment and construct a model of it as the game evolves. By casting the game as a probabilistic search problem, they established both local and global maxima search routines, and demonstrated success in tracking and pursuit. Lygeros *et al.* demonstrated the successful use of a game theoretic construction of an automated highway system [5]. In their approach, a set of autonomous vehicles is controlled, and multiple metrics are optimized on a simulated fully autonomous highway. The focus is on when it is optimal for individual vehicles to join/leave a "platoon" of other vehicles traveling on a similar trajectory, and how best to safely change lanes.

Pareto Optimality

The science of Multi-Objective Optimization (MOO), where it is desired to extremize multiple objective functions (such as autonomy metrics) is relevant. A good overview is given in [6]. This is closely related to set-based design, as described in [7]. Using the nomenclature of [8], given a set of functions of vector **x,** $\{ F_i(\mathbf{x}) \}$, MOO concerns the following

$$\underset{\mathbf{x}}{\text{minimize}}\ \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots F_k(\mathbf{x})] \quad (1)$$

i.e., minimize a vector of independent functions, subject to

$$g_j(\mathbf{x}) \le 0, \quad j = 1,2,\ldots,m \quad (2)$$

and

$$h_l(\mathbf{x}) = 0, \quad l = 1,2,\ldots,e \quad (3)$$

In general, the solution over **X** is not defined. Except in the rarest of cases, there is no one value of **x** that optimizes all the independent $F_i(\mathbf{x})$ functions. What is often used in literature and study as a substitute for a complete solution is a set of domain values that provide a special property called *Pareto Optimality* [8]. This is a value of **x** that cannot be changed without making at least one of the $F_i(\mathbf{x})$ worse.

The randomizing-optimizing framework presented in this paper gives a segmented form of Pareto optimiality. Instead of there being no change in that improves any function $F_i(\mathbf{x})$ while leaving the other functions unchanged or improved, instead no change in segment $\mathbf{x}_1$ can improve $F_1(\mathbf{x})$, no change in segment $\mathbf{x}_2$ can improve $F_2(\mathbf{x})$, and so forth. This solution form has a different set of properties.

APPROACH

Monte Carlo Simulation

The techniques in this paper build on Monte Carlo methods. This approach uses random number generation to solve engineering, science, and math problems. It was invented in the 1940s by Stanislaw Ulam and first implemented by John von Neumann [9]. Randomization had been used before the invention to estimate uncertainties—what distinguishes Monte Carlo methods is the use of randomization to solve deterministic problems. For use with simulation, Monte Carlo methods typically take probability distribution functions for input parameters—mass properties, friction, sensor noise—and calculate probability

distribution functions for output parameters. Monte Carlo simulation can convert probability distributions of terrain slope into probability of successful vehicle navigation, for example.

Monte Carlo simulation is powerful because simulation without randomization is inaccurate. It is never possible to precisely capture and simulate a real-world parameter. Weight is always a little different, friction a little different, lengths a little different. Temperature alone changes physical parameters. The power of Monte Carlo simulation lies in its ability to capture a range of values that contain the true values. The parameters for any real scenario all lie within the probability distributions of a correctly configured Monte Carlo simulation.

Optimization

The framework being developed in this paper also builds on existing methods for numerical optimization. Numerical optimization is widely used for control and analysis. For control, it enables calculation of the best inputs to the system to achieve a desired outcome. Minimizing travel time or distance traversed is used widely and regularly, for example. Optimization produces algorithms that drive engines, communications, and autopilots. For analysis, optimization provides perspective on how well or poorly a system can possibly perform. It allows the potential of a system to be measured and improved, and it can be used to find flaws. A design of a modern vehicle requires analyses to be performed on the many components use in choosing shape, size, structure, materials properties, and placements.

Randomizing Optimizing

The framework described in this article is built on a core algorithm called the randomizing optimizer. This core, depicted in Figure 1 is intended to embrace the uncertainty inherent in modeling complex systems, and use it to actively seek out failure modes and conditions. It captures the capabilities of both Monte Carlo simulation and numerical optimization. The general idea behind this approach is that a metric based on output statistics from a Monte Carlo analysis is optimized over a set of domain values (that can represent either unknowns or design parameters).

For any randomizing-optimizing run there are four important sets of data: 1) fixed a priori information, 2) domain values, 3) random variables, and 4) result values. The fixed a priori information remains the same over all simulation runs. This information represents all parameters of the system that are assumed to be known precisely, and which remain constant regardless of system state or inputs. For an autonomous vehicle system, this might include engine gear ratios, tire radius, and other similar mechanical system properties. The domain values $\boldsymbol{X}$ are the variables over which the system is optimized. These variables represent profoundly unknown values in the system, for which even probability distributions are unknown. Random variables represent a set of system parameters which are known only through their statistical distributions. For example, in an autonomous vehicle. The total cargo and passenger weight will vary depending on how many people and what items are in the vehicle. However, the maximum possible loading, as well as the typical number of passengers can be estimated. These randomized variables are then sampled for each individual run within a Monte Carlo experiment set. A particular set of result parameters y is then logged for each run. These result values might be total time to execute, distance to closest collision, or other similar high-level parameters that may indicate a lower-level behavioral change during the run. From these multiple runs, statistics on the result values are calculated and measured (with a scalar metric μ). This whole process can be viewed as a multivariable function with scalar output, taking the domain values as input and the measurement of the result statistics as the output. This function is optimized (minimum or maximum equally fits)

using the Nelder-Mead algorithm [10,11] (or similar) over the domain values.

The set of all available domain variables in the simulation is $\mathcal{B}$. The domain variables being modified in a given experiment is $\mathbf{X}$, where

$$\mathbf{X} \in \mathcal{B} \tag{4}$$

Similarly, if the time varying simulation state is defined as $\boldsymbol{s}(t)$, the output variable for a single simulation sample i is defined as y_i, and which is calculated using the output variable mapping f:

$$y_i = f(\boldsymbol{s}(t)) \tag{5}$$

The state itself is affected by the two inputs to a given run: the value of the randomized variables for trial i (r_i), and the domain variable values. (2) can therefore be expressed more directly as:

$$y_i = f(\mathrm{r}_i, \mathbf{X}) \tag{6}$$

These output values are concatenated, and the resulting collection $\mathbf{Y}$ and the distribution metric function definition g are used to calculate the distribution metric μ:

$$\mu = g(\mathbf{Y}) \tag{7}$$

In this process, friction, for example, could fall within any of the first three data sets. If considered *a priori* knowledge, friction would be known precisely and we would be inquiring into behavior with that friction value present. If part of the domain variables, we would be seeking to understand the effect of unknown friction and, for example, find any friction values that produce anomalous behavior. If part of the randomized variable set, friction would be known though its statistics, and we would be studying the effect of some other parameter on behavior within this friction environment.

In this approach, an important distinction is made of three levels of knowledge about a parameter. We can know its value, know its statistical distribution, or know only theoretical or common-sense limits on its value. If a parameter's value is known, it is just part of the fixed information. If its statistics are known, it is a random variable. And if its statistical distribution is not even known, it becomes a domain value.

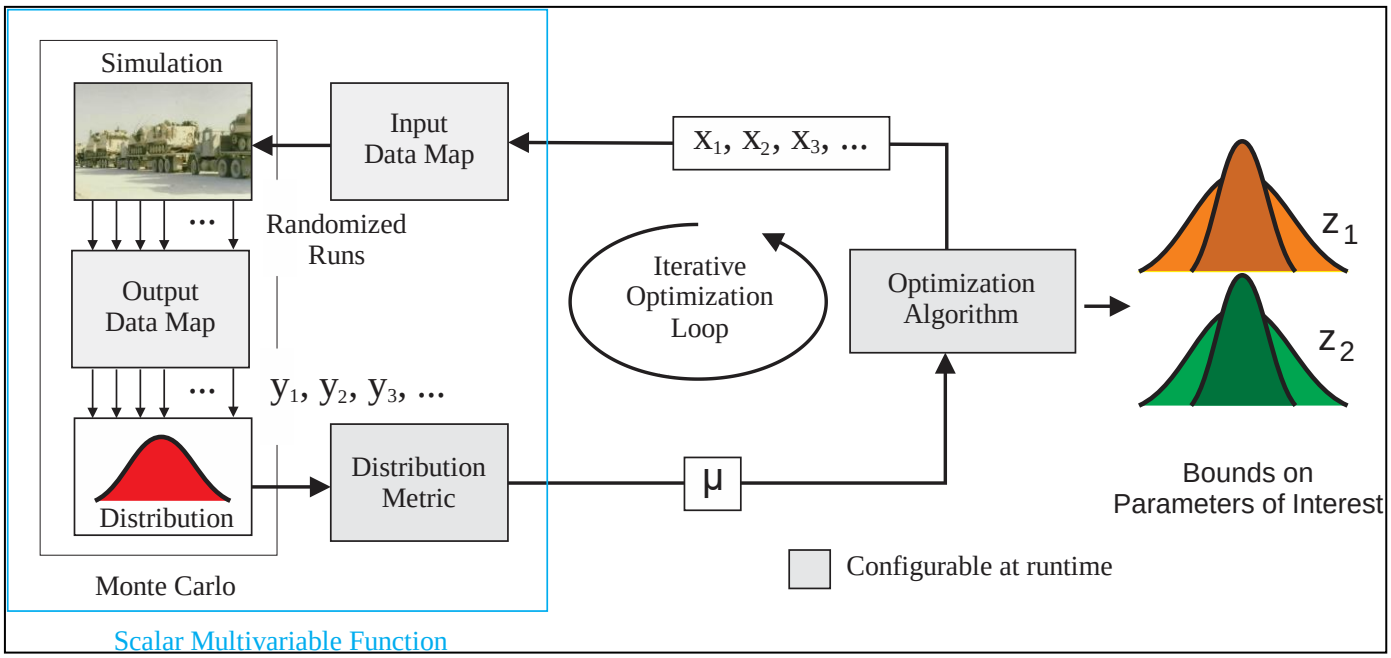

Figure 1: Starting architecture for randomizing/optimizing simulation. This approach randomizes one metric on a distribution over one set of variables.

The randomizing-optimizing framework accommodates all levels of knowledge. Its overall architecture is that shown in Figure 1.

The idea behind this architecture is to formally define algorithms and selections that can be configured and exchanged. The optimization algorithm in the initial implementation, for example, can be Nelder-Mead [10-12], Hook-Jeeves[13-15], multidirectional search [16,17], or implicit filtering [18,19]. The input and output data maps, distribution metric, and simulation can similarly be exchanged.

Multioptimization

In many of the issues raised by vehicle autonomy, there is no single metric that can be optimized in an isolated manner and yield meaningful results. Rather, the system is operating in an environment of improvement, with multiple independent agents attempting to maximize their own payoff. For example, consider the problem of a vehicle travelling over unknown terrain. It is desirable to know what type of terrain is likely to cause a failure in the system. Here, the behavior of the vehicle autonomy algorithm is able to react and adapt to the terrain. In our approach, the character of the terrain is also treated as an adaptive agent, in that it too can adapt to the driver's response, in an attempt to always provide an environment it considers most likely to result in vehicle failure. To accommodate such scenarios, the core randomizing-optimizing architecture above can be extended by providing multiple metrics, which may be competing, and optimizing them over subsets of the unknown parameters. This *multioptimization* approach is, in essence, an application of the *alternating maximization* algorithm, with the integration of the randomizing-optimizer to account for system uncertainty. It is illustrated in Figure 2.

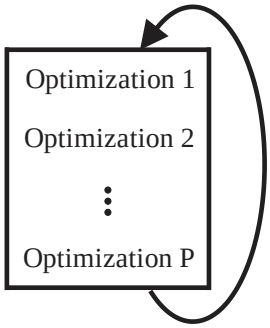

Figure 2: The multioptimization approach. The idea is to sequentially transition among a set of P optimizations. One way to sequence is to repeatedly go through them in order. Each optimization optimizes a different metric over a different subset of the state, using as the other part of the state the results from the previous optimization. Other possibilities may provide advantage, however.

Together, the ideas of Figure 1 and Figure 2 give an architecture as shown in Figure 3. Note that the starting architecture shown in Figure 1 is one configuration of the more general Figure 3 architecture (found by using only P=1 components).

Here, as before, the domain variables for randomizing optimizer j are defined to be

$$\mathbf{X}_j \in \boldsymbol{\mathcal{B}} \tag{8}$$

Similarly, the output of trial i for randomizing optimizer j are defined as

$$y_{i,j} \in \boldsymbol{\mathcal{S}}(t) \tag{9}$$

The simulation output value for sample i is a function of the sampled values of the randomized variables for this randomizing optimizer and trial, $\mathbf{r}_{i,j}$, and the domain variable values $\mathbf{X}_j$ for this Monte Carlo run:

$$y_{i,j} = f(\mathbf{r}_{i,j}, \mathbf{X}_j) \tag{10}$$

The distribution metric μ_j for optimizer j is then is then calculated from $\mathbf{Y}_j$, the set of all $y_{i,j}$:

$$\mu_j = g(\mathbf{Y}_j) \tag{11}$$

Multioptimization can be used to better find corner cases in an environment of improvement (such as one guided by a human or one subject to adaptive control algorithms). In these cases, the

worst case environment can be sought out subject to the best case response. This approach is inspired by game theory where multiple agents seek disparate goals [20,21].

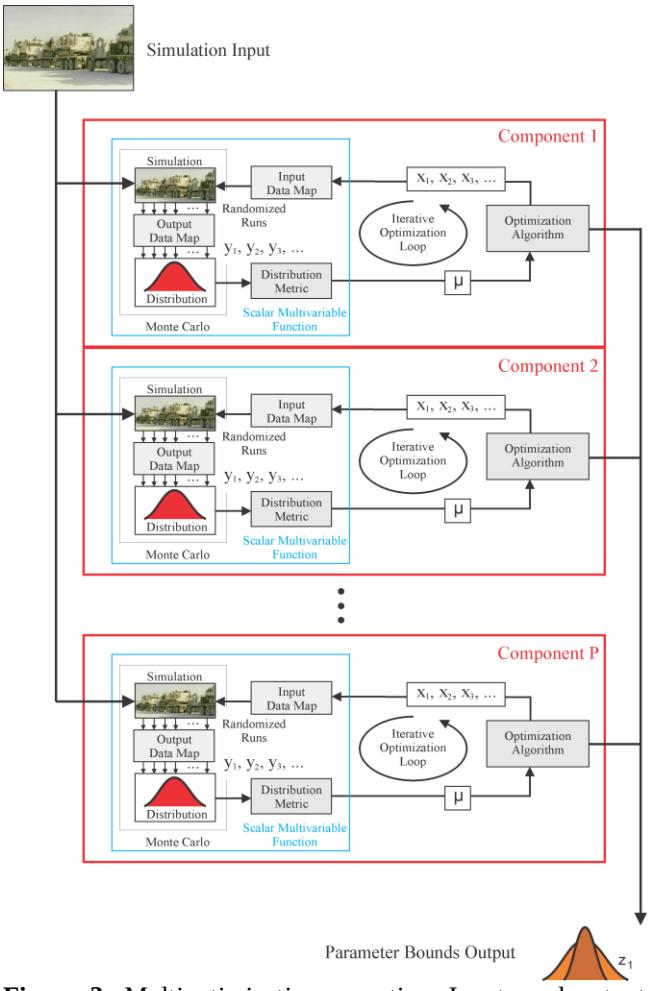

Figure 3: Multioptimization operation. Inputs and outputs are shared among optimization components, each of which can be run any number of times in any order to achieve a final result. All the algorithms and data maps (shown as gray boxes) and the data selections (x and y) can be unique to each component.

Alternating maximization has been shown to yield a globally optimal solution when the metric function is smooth and jointly convex over each of the member's strategies [22]. That is, when each μ_j is smooth and convex over all $\mathbf{X}_j$. For the general application to vehicle problems, these assumptions cannot be made, and the system can only be assumed to be locally optimal. However, by taking a simulated annealing approach and randomizing the starting strategies, a globally optimal solution can be found. In practice, this is equivalent to finding various failure modes of the system.

Nested Optimization

The above model does not always find strictly optimal behavior. That this is so can easily be seen by looking at a simple representative function. Say there are two agents playing the game rock-paper-scissors. In this game, each player choses one of the three options, with rock beating scissors, paper beating rock, and scissors beating paper.

Applying this game to the above framework with two components, one for each player, gives cycles. Say player 1 starts with an always-play-rock strategy. Optimizing for player 2 will give an always-play-paper strategy. Then optimizing for player 1 will give an always-play-scissors strategy. This will then on optimizing again for player 2 give an always-play-rock-strategy. This cycle will continue unabated.

When seeking strict optimization, what we would like to see is the discovery of simultaneous optimization. To do this, we select a nested optimization approach, as shown below.

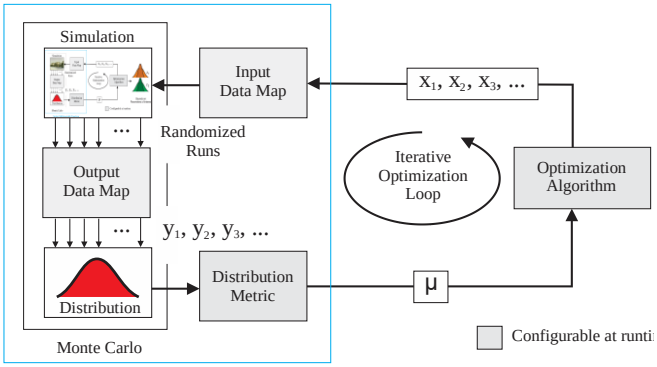

Figure 4: The nested optimization approach. This technique seeks strict optimization.

Generally, the nested-optimization framework gives better results on basic problems, but takes a

long time to run because each step out the optimization in the outer loop must execute a full optimization in the inner loop.

SIMULATION CAPABILITIES

To meaningfully apply the frameworks discussed above, the ability to accurately model the behaviors of all involved systems is critical. These behaviors can be driven by system mechanics and dynamics, sensor capabilities and behaviors (including noise and distortion), control system functionality and the interaction between various systems, and between each system and its environment. We have developed a comprehensive, high-fidelity simulation engine that can capture these behaviors and their complex interplay.

Dynamic Simulation

Once the sensors provide an estimated state, and autonomy control algorithms determine the appropriate action, the final necessary piece is to ensure the simulated system responds appropriately and accurately to these inputs, and interactions realistically with its virtual environment. To accomplish this successfully, a number of components must be simulated. Constrained mechanisms must behave properly, reaction forces between the system and other objects must be computed, the interaction of the system and surrounding terrain must be modeled to a sufficient level of fidelity, and the system actuators must produce a realistic response, given the command provided by the autonomy algorithms. The autonomy testing environment under development provides all of these capabilities, while allowing users to integrate their own simulation components, as discussed above. The following details our approach to modelling the major components of autonomous systems.

To begin, dynamic simulation requires calculating an accurate force response. In the autonomy testing environment, force response is modeled as a function of material type, object positions, and object velocities. A physics-based force response model that returns a force as a function of these properties is applied within the software framework.

Since several models may be employed for force response, and developers of the toolkit may want to add their own, the architecture was constructed to be easily extendable. Any number of force processors can be used within the simulation, and they can be selected using the surface properties of the objects in contact within the simulated environment. By architecting the system in this way, developers have complete control over the force feedback algorithms used. It accommodates, for example, changing the way that frictional forces are calculate without having to change other aspects of the force calculation.

A simple spring displacement model for the collision response force is widely used. It is implemented using knowledge of the penetration distance between two intersecting physical extents. This model has the advantage of returning force as a function of force applied and is well suited for the robotics application domain.

Figure 5: shows a graphical representation of the displacement model. The line of action is the line normal to both colliding surfaces and points in the direction of the resulting force for Surface 1, and in the direction opposite the force for Surface 2. The line of action is obtained as a byproduct of the shape or GJK penetration depth calculation. Friction forces can then calculated using a breaking-spring model.

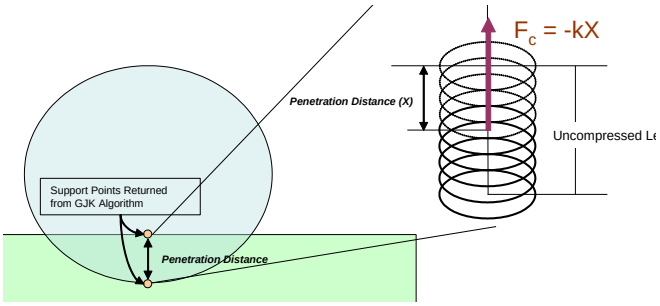

Figure 5: Spring Displacement Model for Force Computation.

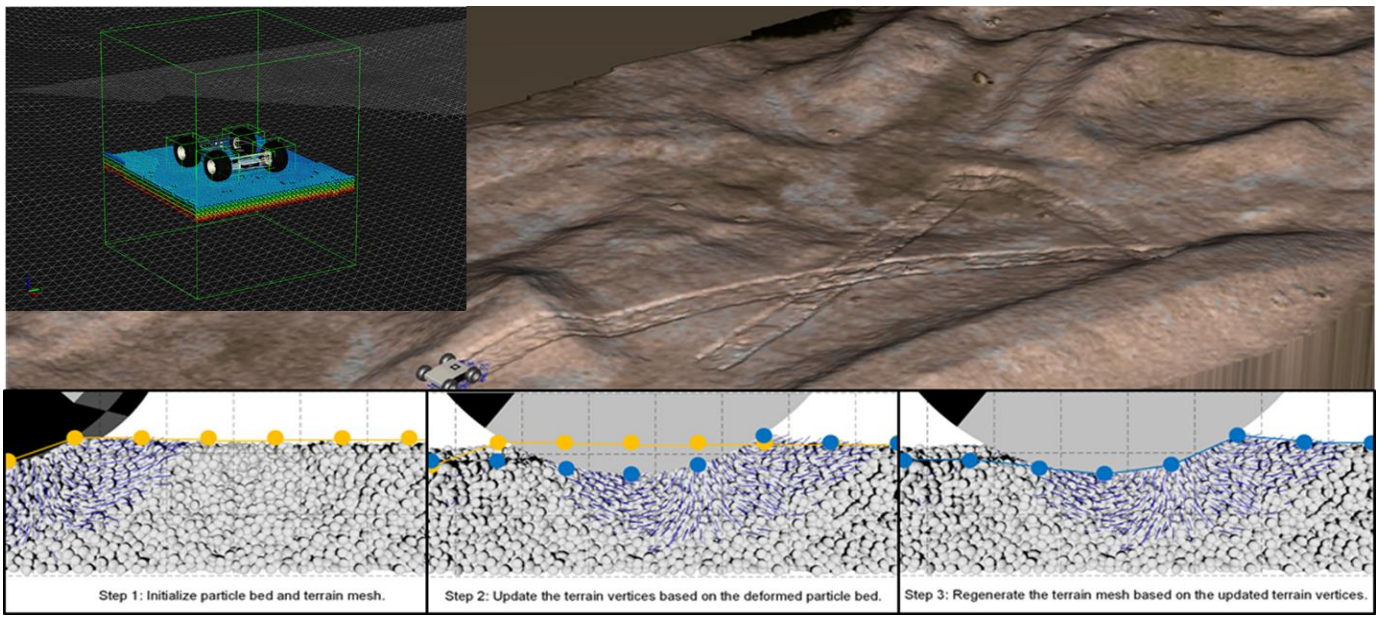

Figure 6: The terrain update functionality allows high fidelity vehicle terrain interaction over large areas. The Particle Simulation can translate over large terrains when an terrain shape is present in the model and the particle system "manipulator" is attached to a mobile robot (as shown in the inset).

After contact forces are calculated as described above, the software applies these forces to an articulated dynamics model. The software includes two dynamic simulation methods: the Articulated Body Inertia Algorithm and the Composite Rigid-Body Inertia Algorithm. The Order(N) Articulated Body Inertia algorithm is best for very large manipulators. The Order(N^3) Composite Rigid-Body Inertia algorithm is best for smaller manipulators. These techniques are implemented for both fixed-base and free-base manipulators.

Particle Simulation

The autonomy testing environment offers a particle-based terrain simulation for complex modelling of granular material in Actin. This simulator is tightly integrated with Actin's dynamic simulation system for robotic manipulators and is implemented using NVIDIA's CUDA platform for processing on compatible graphical processing units (GPUs).

Interaction between particles and other manipulators in Actin is based on the penetration distance between particle spheres and Actin shape primitives used as bounding volumes for the manipulator links. The general simulation loop is shown in the figure below.

The particle system simulator has the ability to translate over large terrains. When a model is loaded that has a compatible terrain shape then this capability is enabled. The particle system can be attached to a mobile robot and will then translate over the terrain simulating a specified depth of the terrain surface under the robot.

The terrain can be dynamically updated to reflect deformations in the underlying particle bed. At every time step, the terrain mesh update algorithm performs a check across all of the terrain vertices to fit the nearest particle position. The algorithm is implemented completely in parallel using Nvidia's CUDA platform for processing on compatible graphical processing units (GPUs). As shown in Figure 6 this powerful utility can be used with Actin's terrain windowing techniques to simulate high fidelity vehicle terrain interaction over large areas.

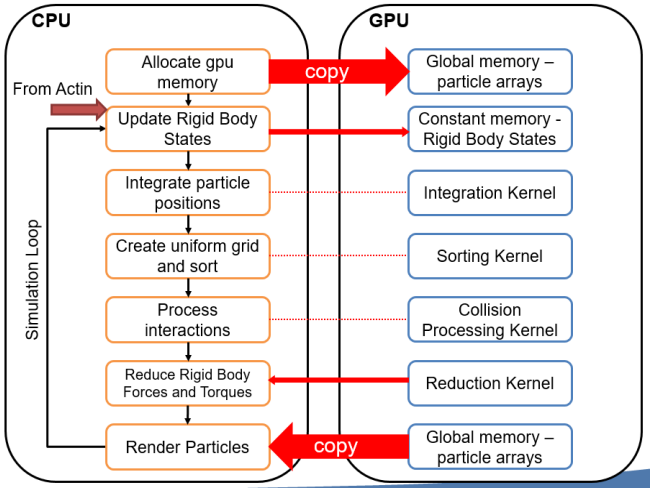

Figure 7: The particle simulation manages calling methods and marshaling data between the CPU and GPU.

Vehicle Drivetrain Simulation

Actin also provides the ability to simulate components of a vehicle drivetrain, including the engine, transmission and shifting logic, and differentials. These components can then be combined to model the actual vehicle drivetrain more completely. Each of these components has been developed such that they can be augmented or replaced by users who wish to develop their own models within Actin, or integrate third-party simulation packages. Example drivetrain configurations for various vehicles drive types are shown below.

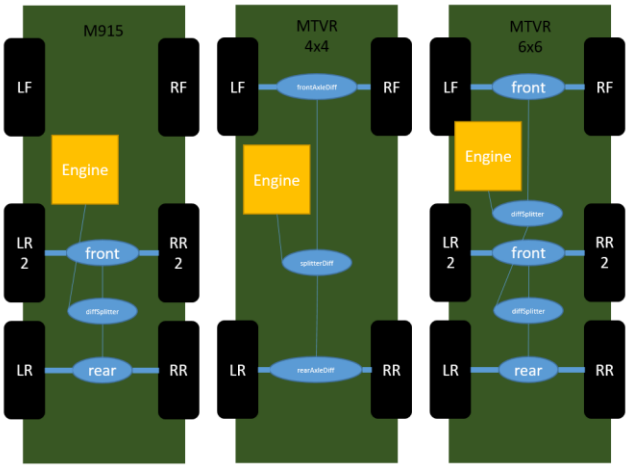

Figure 8: Drivetrain configuration for different vehicles.

Sensor Simulation

The ability to accurately simulate sensor outputs is critical to providing a realistic estimate of the true behavior of an autonomous system. Because these systems rely on sensing to determine not only their own state, but the state of the world around them, a faulty sensor reading (or faulty interpretation) can result in catastrophic failure. Just as real world systems use these to analyze the physical environment, Actin allows simulated systems to use these sensor models to analyze their virtual environment. The autonomy testing environment offers a range of different sensor simulation capabilities, and provides an architecture that makes the development of new sensor types easy. Each of these sensors can be attached to any portion of an autonomous system, including distal links of articulated mechanisms. In addition to the sensor simulations discussed below, the software offers the capability to simulate GPS sensors, accelerometers, and other common sensor types used for navigation, obstacle avoidance and other control tasks in robotics and autonomous systems.

Camera/Image Sensor Simulation

Several camera simulations are included, ranging from simple but efficient OpenGL-based rendering of textured geometry to a state-of-the-art GPU-based ray tracing system. Users can select the level of accuracy and computation needed for their particular application.

As a first step beyond basic textured rendering, users can utilize a simulated High Dynamic Range camera sensor, and an associated configuration plugin. This tool makes use of the OpenGL Shading Language (GLSL), allowing for efficient, highly parallelized processing of images. It also provides the capacity for users to define additional effects to be applied to the scene. These effects could include starbursts (due to diffraction) around bright light sources, or (as is already implemented) bloom effects where light sources also affect their surrounding pixels.

Figure 9: Example HDR image with bloom effect (left), and the targeted real-world image, right.

To address the shortcomings of local rendering discussed above, we also offer ray traced rendering in a form that can be executed on PC hardware in real time. Ray tracing is a global illumination approach that generates a rendering by modeling how light rays react and interact within a scene. This allows the modelling of advanced environmental effects, such as bright sunlight, cloud cover, rain, fog, and dust. This allows image sensor degradation to be more accurately modeled. There are three key components to this approach: the simulated eyepoint or camera, the virtual screen through which rays are cast, and the objects within the scene itself. To generate an image of the scene, rays are cast from the eye point through the pixels of the virtual screen, and into the scene, as shown in Figure 10. These rays then traverse the scene until they either impact an object, or pass out of the bounds of the scene.

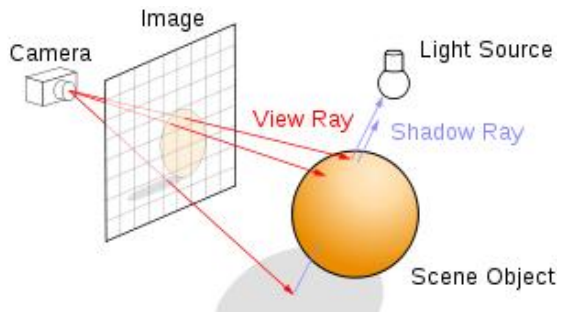

Figure 10: Basic ray tracing algorithm (image: wikimedia.org). The software supports ray tracing in real time on ordinary PCs.

When rays intersect an object, there are several possible behaviors (or combinations of behaviors) that can take place. A ray may be absorbed by the surface, it can reflect off of the surface, or it can refract through the surface (and the body of the object). Any combination of these three behaviors is also possible (and is in fact, likely for realistic materials). For example, the surface of a glass sphere will both reflect a portion of the light incident upon it, and refract a portion of it. In addition, some small amount may be attenuated (absorbed) by the object itself. In addition to reflective and refractive rays, a shadow ray is sent from the intersection point toward each light in the scene. If this ray intersects another object prior to reaching the light, the point lies in shadow, and will be shaded as such. The virtual screen or image is used to map the rays cast into the scene to pixels that will be displayed by the raster-based display.

This capability leverages NVIDIA's Optix ray tracing library. This library utilizes NVIDIA's popular CUDA parallelization library, both of which are constructed specifically for use on NVIDIA's GPU computing hardware.

The following image was generated using ray-traced rendering within the Actin framework. The scene graph was automatically converted, and textures transferred to the GPU. On lower end hardware (Quadro K1100M graphics card), a framerate of approximately 2-3FPS is achieved, while framerates of approximately 25-30FPS are achieved using the Titan X GPU from NVIDIA.

Figure 11: Example real-time ray traced image using the software, including shadows and texturing for terrain and truck surfaces. Shadowing and reflection are emergent behaviors.

Radar Sensor Simulation

A simulation was developed for the Delphi ESR radar used in many automotive and autonomous vehicle applications. This radar simulation models the major behaviors of a radar system, and leverages the rendering and parallelized GPU-based processing pipelines.

The first several steps build on a depth map generated from the scene, then thresholded and contoured to identify individual potential targets. The total reflected power from each target is then calculated. This power is a function of a surface's distance from the radar sensor, its surface normal relative to the sensor normal, and the reflectivity properties of the surface. Figure 12 illustrates the effect on power each of these contributions can have. In these images, a color of white corresponds to a factor of 1.0 (that is, the reflected power is not reduced at all), while black corresponds to factor of 0.0, with no power being reflected from that point. The overall process to identify radar targets is outlined in Figure 13.

During the target pruning phase, identified targets which would not be identified by the sensor are removed from the identified targets list. These targets may either be reflecting too little power to be detected by the sensor, or outside the range of the sensor.

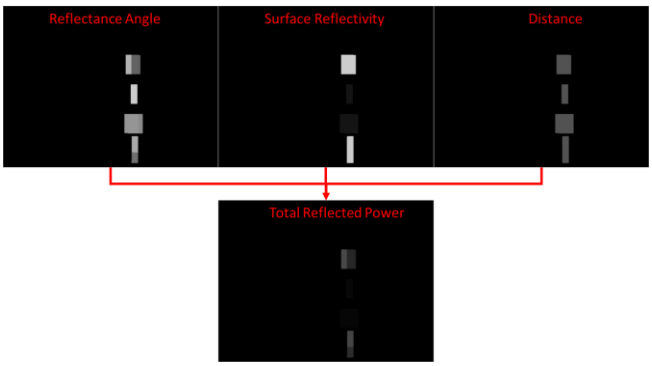

Figure 12: Illustration of how surface angle, reflectivity and distance are combined to provide an estimate of total reflected power. In this example, the top and bottom blocks have relatively high surface reflectively, while the middle boxes exhibit low reflectivity.

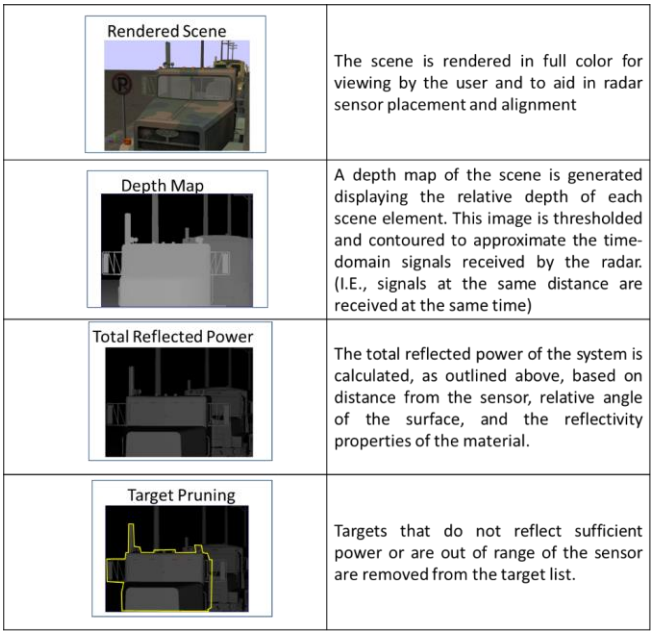

Figure 13: Simulated radar sensor processing pipeline. To ensure efficiency, all steps with the exception of target pruning are performed on the GPU.

LIDAR Simulation

A simulation was also developed for the Velodyne HDL-32E LIDAR sensor. This LIDAR unit has 32 lasers mounted vertically on a rotating head. The LIDAR head rotates at 600 rpm providing angular resolution of 0.16 degree in the horizontal and 1.33 degrees in the vertical. The unit returns 700,000 points per second in the range of 1 to 70 meters from the head with 360 degree horizontal by 40 degree vertical field of view.

The approach taken to simulate the Velodyne HDL-32E is to use the OpenGL Z-buffer generated by the graphics card to obtain a representative sample of 3D points in a 360 degree view around the LIDAR unit. The depth buffer (also known as z-buffer) is used in OpenGL to resolve the distance between two nearby objects to determine which objects should be hidden behind which others. To simulate the Velodyne HDL-32E efficiently, the 360 degree scene around the unit is rendered by a virtual camera rotated by a set angle at each simulation time step. The Z-buffer of each rendered frame is sampled to obtain the proper number of 3D scan points and plotted in the scene as a 3D point cloud. The simulation frame rate is the main constraint for accurate capture of motion effects. The simulation frame rate is the main constraint for accurate capture of motion effects. The relationship between the scan rate (α_{scan}) of the virtual camera with respect to the simulation frame rate (r_{sim}) and actual LIDAR angular velocity (ω_{lidar}) is as follows.

$$\alpha_{scan}\left(\text{rad/frame}\right) = \frac{\omega_{lidar}\left(\text{rad/s}\right)}{r_{sim}\left(\text{frames/s}\right)} \tag{12}$$

Therefore to achieve the desired scan rate of 10 revolutions per second at a simulation frame rate of 30 Hz, the virtual camera must scan 120 degrees at each time step.

The non-linear scan pattern of the HDL-32E must be taken into account. In Actin, this is implemented based on a mapping of scan azimuth and elevation to depth image pixel space using a non-linear image distortion model. In this way, the depth image is warped based on the distortion model and sampled to extract depth measurements corresponding to each scan ray. Figure 14 shows the resulting simulated non-linear scan projection on a flat vertical wall and surrounding flat terrain surface.

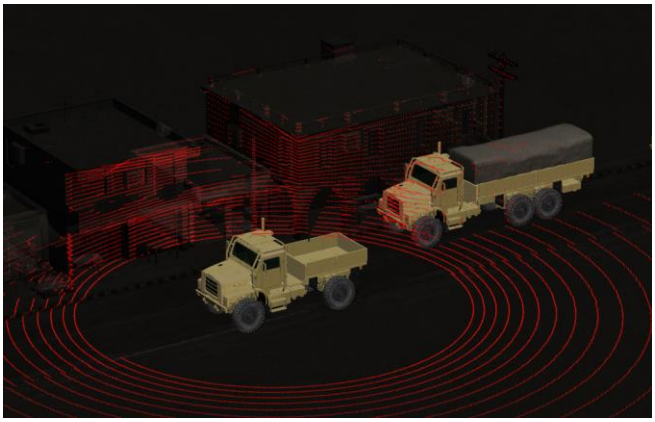

Figure 14: LIDAR simulation with convoy vehicles in Actin Viewer showing the correctly simulated hemispherical projection of the Velodyne HDL-32E Lidar on to both flat terrain and complex shapes.

Substantial effort has been taken to optimize the Velodyne Lidar simulation. The first optimization was to render the Lidar views with a multithreaded render camera objects and enable OpenGL context sharing. In this way the Lidar view is rendered using a graphics context that is shared with all other views of the scene graph. This optimization eliminated latency from frequent graphics context switching.

EXAMPLE APPLICATION AND RESULTS

This example application uses a single randomizing optimizer to identify a failure that occurs when the collision avoidance system commands a rapid braking maneuver to avoid colliding with an obstacle. The sudden deceleration results in the front end of the vehicle diving downwards, and the obstacle exiting the field of view of the radar system. With no obstacle in view, the vehicle then begins to accelerate once again. This failure illustrates the complex interplay between sensors, control logic, and vehicle dynamics. The scene setup is shown in Figure 15 below.

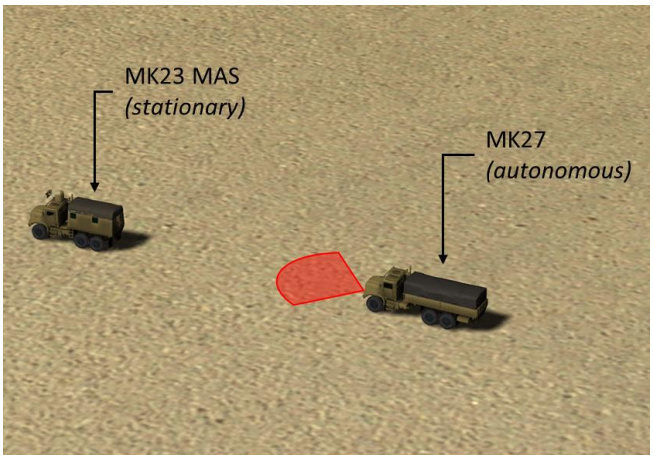

Figure 15: Collision avoidance scenario.

Collision Avoidance System

The collision avoidance logic is based on feedback from a simulated radar sensor mounted on the autonomous vehicle, as illustrated through red markup above. For this scenario, the control logic of the collision avoidance system is relatively simple to allow specific parts of this logic to be more thoroughly studied. No attempt is made to steer around an obstacle in the path of the vehicle—rather, once the object passes within a certain distance of the vehicle, the brakes are applied and the vehicle brought to a stop. The acceleration/braking response is defined by two key parameters: the maximum braking distance, and the braking slope. The maximum braking distance defines the farthest range at which a radar target will trigger the brakes to begin to be applied. The braking slope defines how quickly the braking intensity is increased as the target is approached. This behavior is illustrated in Figure 16 below.

When the system is within the specified braking distance, the braking intensity is increased linearly based on the braking slope and max braking range.

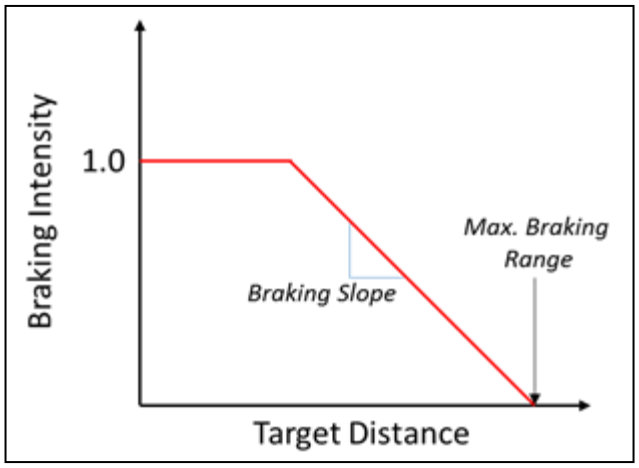

Figure 16: Collision avoidance control logic. A braking intensity of 0 indicates no braking action is being applied, while an intensity of 1 is sufficient to lock to the wheels.

Randomizing Optimizer Configuration

Domain Variables: To enable the randomizing optimizer to efficiently find faults, an appropriate set of domain variables is selected to form the parameter space of the system. For this demonstration, we have chosen to focus solely on the effects of the collision avoidance control. Given this, the domain variables were chosen to be the braking slope and maximum braking range defined above.

Randomized Variables: The randomized variables for this scenario were chosen to be items that would significantly affect the response of the vehicle to sudden accelerations. Here, we've chosen to randomize both the vehicle payload mass and the stiffness of the suspension system (both front and rear). Additional factors that could be considered include tire and ground coefficient of friction, obstacle reflectance properties, and radar noise properties.

Output Variables: As in the previous demonstration, the output variable was selected as a simple distance measurement. In this case, the distance between the autonomous vehicle and the obstacle. Note that here, this is the "true" distance in the simulation, calculated from its state, rather than the distance measured by the radar sensor

(this eliminates any artificial variation in the output due to the sensor).

Distribution Metric: The optimizer attempts to maximize the value of the distribution metric, which is calculated from the output variables. For this demonstration, the distribution metric was chosen to be the variance of the output distance. This results in the optimizer seeking out areas of the parameter space where the system behavior varies greatly across the values of the randomized variables. This can indicate a transition in system behavior, often indicative of an impending failure.

Results: The newly developed randomizing optimizer was run using a representative configuration, as illustrated in Figure 15. After approximately 118 iterations of the Nelder Mead optimizer, the failure point was found. Here, the failure mode was substantially similar to that observed during live vehicle testing. A sudden braking motion causes the front end of the vehicle to dive, resulting in a loss of obstacle recognition by the radar. The collision avoidance controller then allows the system to speed back up. This behavior repeats several times before finally bringing the vehicle to a stop. An example of the changes in vehicle pitch is given in the figure below.

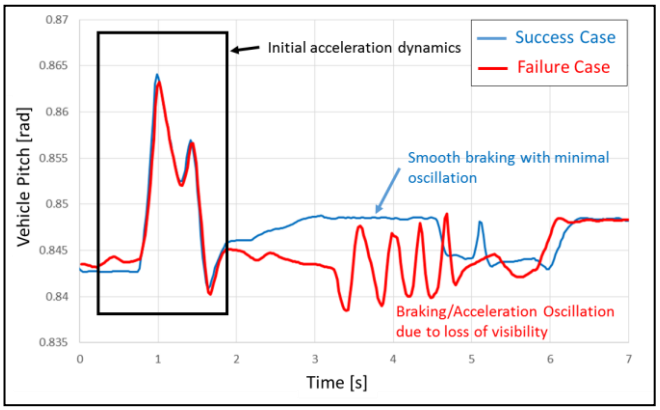

Figure 17: Difference in vehicle pitch in normal smooth braking versus the oscillatory behavior found using 118 iterations of failure seeking with the randomizing/optimizing framework.

This behavior was found to be especially pronounced when the stopping distance was relatively far away from the obstacle, and the braking slope was set to a steep value. This is logical, since at a farther distance from the obstacle, smaller diving angles will result in the line of sight between sensor and obstacle being lost. Similarly, more rapid braking will excite more significant dynamics within the vehicle. The evolution of both braking distance and slope to achieve this result is shown in Figure 18 below.

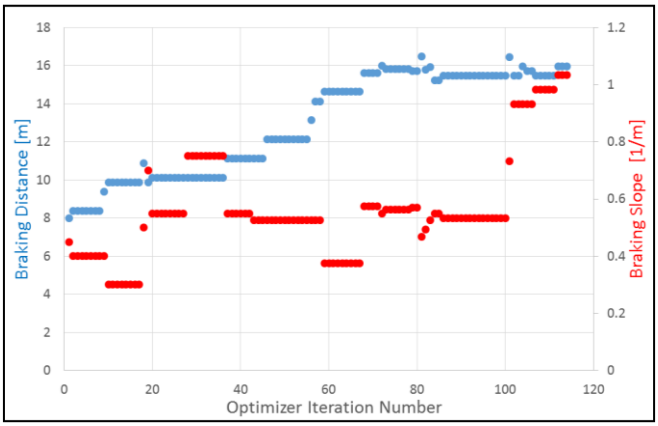

Figure 18: Evolution of collision avoidance parameters using the Nelder-Mead optimizer.

During the configuration and testing of these results, several interesting behaviors were observed within the randomizing-optimizer framework. First, the number of trials within each Monte Carlo run has a significant impact on the performance of the optimizer, and should be chosen carefully. This minimum number required to obtain a reasonable estimate of the distribution function depends on several factors. As the number of randomized variables increases, the number of samples must be increased to obtain a statistically relevant sample. Similarly, as the probability distribution becomes wider, a larger number of samples will need to be taken to provide coverage of the distribution. For this demonstration, to estimate the number of samples required, the Monte Carlo function was run with a varying number of sample runs, and the

distribution metric recorded. For the two randomized parameters under consideration in this scenario, the results are shown in Figure 19 below.

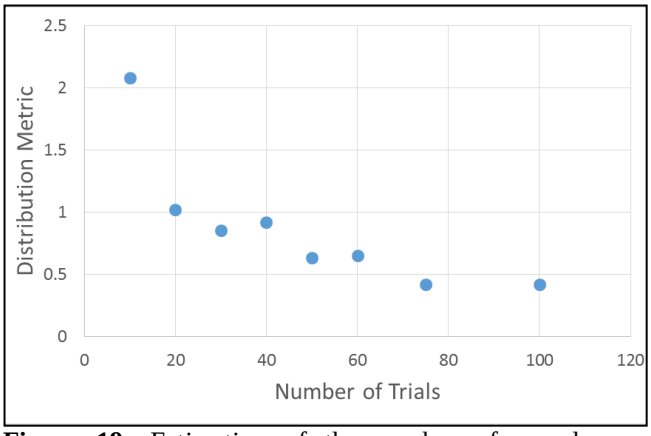

Figure 19: Estimation of the number of sample runs required within the Monte Carlo function. Here, both payload mass and suspension stiffness were varied, and the distribution metric calculated for a varying number of trials. Approximately 75 trials yields stable results.

Based on these results, 75 trials were used for each run of the Monte Carlo function in this demonstration. This was selected to yield stable, accurate results while limiting the number of simulation runs required.

Another behavior that drives the configuration of the framework is the ability of the system to seek out failures. For example, in this demonstration, if the payload and suspension distributions are relatively narrow, it is much less likely that the system will experience a failure. As these distributions are widened, the probability of observing a failure on a given iteration increases. Once some trials begin to observe failures, this drives changes within the distribution metric, and allows the system to more effectively seek out the "hot spots" of these failures. For example, in Figure 18, prior to iteration 30, the system appears to be hunting with no particular direction. At this point, the braking distance increases rapidly, indicating a strong gradient. Shortly after, similar behavior can be observed in the braking slope.

CONCLUSIONS AND FUTURE WORK

Conclusions

The framework presented in this paper provides the ability to actively seek out failure modes for a variety of systems, including autonomous ground vehicles and convoys. It offers a number of advantages over traditional Monte Carlo-style simulations. First, it provides users with a pathway to consider profoundly unknown parameters in their system analysis. These parameters can have a profound effect on the behavior of the system, and are difficult or impossible to fully consider in traditional approaches. The approach is also highly extensible, and can be extended to any system which can be digitally modeled. The approach also allows users to adjust the accuracy of the simulation to meet their specific needs. For highly accurate vehicle performance models, third party modules can be plugged in to more precisely model the components such as the vehicle drivetrain, tires and chassis. Finally, because of the numerous independent simulation runs, the approach lends itself particularly well to deployment across multiple processors and machines. This parallelization significantly improves run times.

This approach does, however, have its limitations. Like any simulation-based approach, the results obtained are dependent on the accuracy of the underlying simulation. A misunderstanding of the underlying physics, or an incorrectly tuned set of parameters can identify a failure mode that is not present in the true system, or miss a failure mode altogether. Additionally, because of the large number of simulation runs and the complexity of the underlying systems, this is an extremely computationally intensive approach. This can be mitigated somewhat by the parallelization discussed above, but with current technologies, the system is suited well for offline simulation only. Real-time analysis of failure modes would require an order of magnitude increase in computing power.

Future Work

While the system presented here is a powerful and flexible approach, there are several ways to meaningfully extend its capabilities. One important planned improvement is to incorporate concepts from artificial intelligence/machine learning for classification algorithm when the system is faced with each new scenario. If it can be closely matched with a previous scenario, that can provide a logical place to begin looking for failures. Here, the machine learning system would be trained based on simulation data collected over previous analyses. This approach would also allow the transfer of knowledge to a fielded system, where the classification system can be used to identify behaviors and situations where failure is imminent, and take the appropriate corrective actions to avoid it.

ACKNOWLEDGEMENTS

The authors wish to thank the United States Army Tank Automotive Research, Development and Engineering Center for their support of this work through Small Business Innovation and Research grant W56HZV-15-C-0171.

REFERENCES

[1] Schultz, Alan C., John J. Grefenstette, and Kenneth A. De Jong. "Test and evaluation by genetic algorithms." *IEEE Expert* 8, no. 5 (1993): 9-14.

[2] Wegener, Joachim, and Oliver Bühler. "Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system." In *Genetic and Evolutionary Computation–GECCO 2004*, pp. 1400-1412. Springer Berlin Heidelberg, 2004.

[3] Emery-Montemerlo, R., Gordon, G., Schneider, J., & Thrun, S. (2004). Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (AAMAS), (pp. 136–143). Washington, DC, USA. IEEE Computer Society.

[4] Vidal, Rene, Omid Shakernia, H. Jin Kim, David Hyunchul Shim, and Shankar Sastry. "Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation." *Robotics and Automation, IEEE Transactions on* 18, no. 5 (2002): 662-669.

[5] Lygeros, John, Datta N. Godbole, and Shankar Sastry. "Verified hybrid controllers for automated vehicles." *Automatic Control, IEEE Transactions on*43, no. 4 (1998): 522-539.

[6] R.T. Marler and J.S. Arora, "Survey of Multi-objective Optimization Methods for Engineering," *Struct Multidisc Otpim*, 26, 369-395 (2004).

[7] S.E. Hannapel, N. Vlahopoulos, and D.J. Singer, "Including Principles of Set-based Design in Multidisciplinary Design Optimization," *12th AIAA ATIO Conference and 14th AIAA/ISSM* 17-19 September 2012, Indianapolis.

[8] V. Pareto, Manuale di Economica Politica, Societa Editrice Libraria. Milan, 1906. (https://books.google.com/books?hl=en&lr=&id=oA6oZpaIQwoC&oi=fnd&pg=PA1&dq=Pareto+1906&ots=Z0ZxzKiTtZ&sig=BTlU6C-pRczFHPSb1zumq6JcPDQ)

[9] Eckhardt, Roger. "Stan Ulam, John Von Neumann, and the Monte Carlo Method."*Los Alamos Science* 15 (1987): 131-136.

[10] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes in C, Cambridge University Press, 1990.

[11] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," Computer Journal, vol. 7, pp. 308-313, 1965.

[12] C.T. Kelly, "Detection and Remediation of Stagnation in the Nelder-Mead Algorithm Using a Sufficient Decrease Condition," *SIAM Journal on Optimization*, vol. 10, no. 1, pp. 43-55, 1999.

[13] V. Torczon, "On the Convergence of Pattern Search Algorithms," *SIAM J. Optimization*, v. 7, no. 1, pp 1-25, 1997

[14] C.T. Kelly, "Iterative Methods for Optimization," SIAM, 1999.

[15] A.F. Kaupe, "Algorithm 178, Direct Search," *Comm. ACM* v. 6, no. 6, 1963.

[16] V. Torczon, Multi-Directional Search: A Direct Search Algorithm for Parallel Machines, Ph.D. Thesis, Rice University, 1989.

[17] V. Torczon, "On the Convergence of Pattern Search Algorithms," *SIAM J. Optimization*, v. 7, no. 1, pp 1-25, 1997.

[18] R. Hooke and T. A. Jeeves, "'Direct Search' Solution of Numerical and Statistical Problems", *Journal of the ACM*, v. 8, pp. 212-229, April 1961.

[19] P. Gilmore, C.T. Kelly, C.T. Miller, and G.A. Williams, "Implicit Filtering and Optimal Design Problems," *Progress in Systems and Control Theory*, vol. 19, Birkhauser, Boston, pp. 159-176, 1995.

[20] Glicksberg, Irving L. "A Further Generalization of the Kakutani Fixed Point Theorem, with Application to Nash Equilibrium Points." *Proceedings of the American*

Mathematical Society 3, no. 1 (1952): 170-174.

[21] Muench, Paul, Bednarz, Dave, Singh, Amandeep and Mange, Jeremy. "Mobility as a Game of Timing." In *Proceedings of the 2013 Ground Vehicle Systems Engineering and Technology Symposium.* Troy, MI.

[22] Bezdek, James C., and Richard J. Hathaway. "Convergence of alternating optimization." *Neural, Parallel & Scientific Computations* 11, no. 4 (2003): 351-368.

[23] C.T. Kelly, "Detection and Remediation of Stagnation in the Nelder-Mead Algorithm Using a Sufficient Decrease Condition," SIAM Journal on Optimization, vol. 10, no. 1, pp. 43-55, 1999.

[24] V. Torczon, "On the Convergence of Pattern Search Algorithms," SIAM J. Optimization, v. 7, no. 1, pp 1-25, 1997