Autonomous Vehicles Scenario Testing Framework and Model of Computation

Ala' J. Alnaser
Advanced Mobility Institute
Florida Polytechnic University
Lakeland, FL USA
aalnaser@floridapoy.edu

Mustafa İlhan Akbaş
Advanced Mobility Institute
Florida Polytechnic University
Lakeland, FL USA
makbas@floridapoy.edu

Arman Sargolzaei
Advanced Mobility Institute
Florida Polytechnic University
Lakeland, FL USA
asargolzaei@floridapoy.edu

Rahul Razdan
Advanced Mobility Institute
Florida Polytechnic University
Lakeland, FL USA
rrazdan@floridapoy.edu

***Abstract*—Autonomous Vehicle (AV) technology has the potential to fundamentally transform the automotive industry, reorient transportation infrastructure, and significantly impact the energy sector. Rapid progress is being made in the core artificial intelligence engines which form the basis of AV technology. However, without a quantum leap in test and verification, the full capabilities of AV technology will not be realized. Critical issues include finding and testing complex functional scenarios, verifying that sensor and object recognition systems accurately detect the external environment independent of weather conditions, and building a regulatory regime which enables accumulative learning. The major contribution of this paper is to outline a novel methodology for solving these issues with the use of the Florida Poly AV Verification Framework (FLPolyVF).**

I. Introduction

In the recent years, AVs have attracted great attention from academia, industries, and governments. Perception, decision making, and action are the three major processes required for driving a vehicle. Currently, decision-making and perception are controlled by human drivers, and the action process is performed by the vehicles.

According to a report of the National Highway Traffic Safety Administration (NHTSA), 94 percent of the 37,461 traffic fatalities in 2016 were due to human error [1], [2]. AVs are designed to conduct the decision making and perception aspects of driving, and it is hoped that will reduce accidents related to human error. In addition, studies also show that autonomous driving technologies can positively impact economy, safety, and traffic congestion [3]. Despite all of these advantages, the major barrier for wide-scale adoption of AVs is the test and verification regime to assure safety. To address this barrier, a process, which builds an engineering argument for assuring safety, must be developed. Typically, this argument is built based on the following principles:

1) Conceptual model: A conceptual understanding of the problem is built and supported through virtual models.
2) Test Regime: Using the conceptual model, a test regime is built to test the model and build an argument for correctness.
3) Completeness: The state space of tests is examined within the modeling environment to develop metrics for completeness.
4) Accumulative Learning: A structure is constructed where field testing feeds back into this flow such that safety is always rising.

Intertwined with the above methodology is the classic V paradigm model [4], [5] which is used as a mechanism to enable concurrent design and test. In this paradigm, mathematical models, which have been correlated with a bottom-up component level characterization stage, are used early in the design stage. As the design is refined, physical components can be substituted to a point when system level tests can be performed on the whole physical design. Modeling issues are often corrected with a virtual to physical diagnostics flow. The combination of the conceptual safety regime and the V design process have been effectively used to build robust safe systems in many domains.

In fact, the above flow has been used very successfully by the automotive industry to verify conventional cars for many years. However, the addition of the perception and decision making has added an order-of-magnitude level of complexity to solving the safety problem. Critical open issues can be listed as follows:

1) Conceptual model: What are the conceptual models, which are appropriate for the perception and decision making stages of AV operation?
2) Test Regime: What is the test regime, which can build confidence as to the operation of the AV?
3) Completeness: How do we understand the state space sufficiently to understand the risks surrounding completeness? How do we address the issues of tester bias?
4) Accumulative Learning: How do we know the next version of the vehicle or even software update is actually increasing safety?

We observe that today none of the above questions is answered for AVs. The current commercial solutions are using ad-hoc methods such as miles driven [6] to provide some indication of safety, but no fundamental structure has been offered to demonstrate the robustness of AV products. Further, without the answers to above mentioned questions, regulators do not have the means to address safety issues or even to communicate clearly safety issues to operators or the public.

In the remainder of this paper, we will describe a conceptual framework for addressing the key verification issues for AVs, and propose a mathematical modeling abstraction which is critical to enabling the effective functioning of the environment. The intent of this work is to answer the aforementioned questions for AVs.

II. Characteristics of the Solution

A. Hardware Design

We observe that the realm of AV verification has quite a number of similarities as compared to complex hardware verification. In the realm of hardware, millions of extremely complex components (transistors) are assembled together to form a higher level function which is embodied in a semi-conductor chip. The cost of the development of these semi-conductor chips is very high and simulating the whole chip at the physics level is impossible. Even at the highest level of abstraction, most semi-conductor chips can only be simulated in the low kilo-hertz range while the chips themselves run in the giga-hertz range. Thus, in any design project, a very limited simulation budget exists (a day of real-world operation) to run all the tests to assure safe operation for the lifetime of the part.

What are the analogies to AV verification? First, much like hardware, AVs consist of complex components (sensors, object recognition systems, radar, etc). Simulating everything at the most detailed level is similarly impossible. Second, much like hardware, AVs need to compress 18 years of human traffic learning into a reasonable development cycle. Third, AVs have the same need for robustness relative to environmental conditions, and finally, they have the same need for completeness and accumulated learning.

World class hardware verification teams solve these problems with a variety of approaches. The first and most important of these is the use of abstraction as a powerful tool to decompose the problem. Within hardware, various abstraction levels have been developed (transistor, gate, RTL, micro-architecture, architecture, network stack) which separate concerns and build an inductive proof for verifying the whole semiconductor chip.

As an example, the transistor design team focus entirely on the task of making sure the semi-conductor physics process produces a behavior consistent with a transistor under all environmental conditions. A cell designer relies on this behavior to build larger components and only verifies that the combination works as expected. Thus, via this recursive process, a chip is built and verified. These separation of concerns and abstractions are so powerful that whole multi-billion dollar markets (fabless, asic) [ref] have been built based on these concepts.

This inductive process is very effective in demonstrating equivalence between abstraction levels, but this still leaves the verification of the highest level of abstraction. In this area, hardware verification has used a variety of techniques such as formal verification, constrained-random test generation, and real-world test injection to model and test the overall function [ref] . Finally, a deep concept of coverage analysis exists in order to model completeness. The combination of all of the above has created an environment where most semi-conductor chips are typically functional on the first pass of manufacturing despite the enormous complexity and size.

How can AV verification use the powerful methods developed for hardware verification? The key is the development of an enabling abstraction level.

B. AV Framework

In this section, we introduce a key enabling abstraction approach to aid in the task of AV verification, which we will term FLPOLY Scenario Abstraction (FLPolySA). FLPolySA has the following high level characteristics:

1) Wireframe/Building Block: Components are very simple recto-linear objects which model the physical characteristics of the environment.
2) Dynamic and Static: There are two types of objects. Static objects which do not move and dynamic objects which move with a pre-determined vector. These components are not responsive.
3) Assertions: Both the dynamic and static components contain meta-data which assert expected behavior. Example: A static component such as a Stop-sign might assert that all cars approaching it must stop.
4) Newtonian Physics: The behavior of the components follows the rules of simple Newtonian physics. A critical idea which is modeled is the notion that mass with velocity/acceleration/gravity will lead to the expected behavior.
5) Unit Under Test (UUT): An ego car can enter this test and has the potential to achieve success or failure if none of the error assertions fire.

The precise details of the model will be explained in succeeding sections. However, at this point, it is important to motivate the design of FLPolySA. There are many powerful consequences for choosing this higher level simple abstraction structure:

1) Mathematical language: The abstraction allows for a clear capture of the modeling environment and provides a basis to reason about this environment.
2) Extended Coverage: A single abstract model contains within it many underlying combinations. As an example, the black-box car could be used to model vehicles from any brand.
3) System Coverage: A signature process for the whole model can be used to drive a test coverage process. A signature process is required to understand what scenarios have been examined earlier and when a scenario is indeed new.
4) Separation of Concerns: The abstraction allows the sensor/object recognition problem to be addressed independently from the decision making problem.

Overall, FLPolySA helps address many of the critical issues mentioned in the introduction as missing aspects for AV verification. Figure 1 shows the conceptual layering of this abstraction approach.

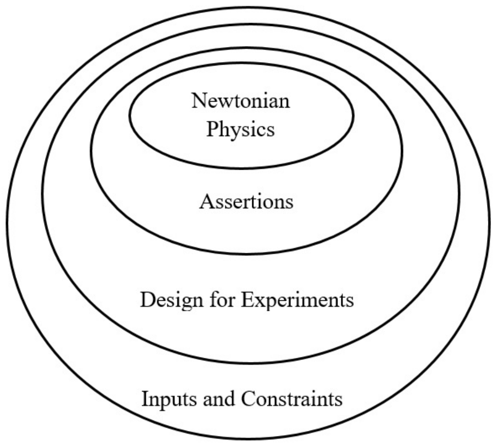

Fig. 1: Scenario Abstraction Levels

1) Newtonian Physics: The function of this layer is to model movement, momentum, and detect collisions. This is the physical world.
2) Assertions: Layered over the physical world is the idea of good or bad.
3) Design for Experiments: This environment is setup such that the test has no memory and thus is repeatable.
4) Inputs and Constraints: Layered in the outermost layer is the machinery for inputs and constraints which drive a singular test. This machinery is setup to be able to run constrained pseudo-random tests and collect ongoing coverage.

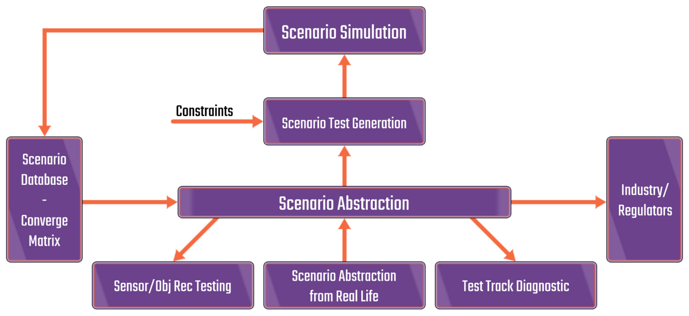

Fig. 2: Scenario Abstraction and FLPolyVF Framework

With the introduction of FLPolySA, an overall architecture can be developed for FLPolyVF. Figure 3 shows the blocks and flows among the blocks in this environment. The critical pieces include:

1) Scenario Test Generation: Using a seed and a constraints matrix, pseudo random experiments can be generated and tested automatically against the unit under test (UUT). The objective of this part of the flow is to simulate millions of configurations in simulation and try to find test cases which cause ego-car to fail. This part of the flow requires the mathematics to understand input constraints and enable a signature flow for deciding whether a test has been generated earlier.
2) Scenario Database/Coverage: The notion of scenario coverage is critical in FLPolyVF. This is the method which can be used as a basis for regulatory approval, limit redundant simulation, and form the basis of an ongoing test and verification database. The scenario abstraction provides an excellent method to capture this information and further the defined mathematics has a strong concept of equivalence classes which allows for a deeper optimization of the coverage database.
3) Test Track Diagnostic: Once a test fails, there is a need to diagnose the test in a deeper level of abstraction. This can be more detailed simulation models all the way to a physical test track environment, and FLPolyVF allows for this path through a synthesis process from the scenario test to a physical instantiation.
4) Sensor Test and Verification: A critical part of an AV is the object recognition system and sensors. The scenario abstraction can provide test cases with built-in criteria of success such that the sensor/object recognition problem can be verified separately.
5) Scenario Abstraction: Just as there is a need for synthesis from the scenario level, there is a need to abstract the scenario construct from the physical environment. A classic example is a "real-world" accident which must be analyzed much more completely in the simulation framework. In addition, it is common to test for "cousin" bugs [add ref] with this flow.
6) Industry/Regulators Communication: There is a need to communicate in an unambiguous manner among the industry participants and the regulators. The FLPolyVF scenario abstraction provides an excellent means for this communication.

The remainder of this paper organized as follows: Section III discusses the related work, Section IV defines the scenario level of abstraction in a mathematically unambiguous fashion and also provides theorems and proofs critical to enable the proposed AV test and verification flow. Section V provides an illustrative example. Section VI provides details on the FLPolyVF instated in software, and Section VII offers come conclusions.

III. Related Work

Several simulation based approaches have been proposed previously for the testing and verification of AVs [7]–[12].

Hallerbach et al. [7] proposed a simulation-based critical-scenario identification platform through the use of a metrics based filter to define critical situations. This method is useful in generating interesting test cases. However, there is no concept of coverage/completeness and the method is highly dependent on the definition of the critical scenario metric which has the danger to be arbitrary.

A methodology for verifying safety of connected and autonomous vehicles (CAVs) is discussed in [8]. The proposed approach verifies safety of CAVs during its online operation, via reachability analysis to account for different possible mathematical models of behavior under uncertain conditions such as existence of disturbances and sensor noise. This approach is a companion to the core system for just verification. The model has some interesting properties, but creates another system which must itself be verified. Without a framework for verification, neither the core nor the verification system can be evaluated from a safety point-of-view.

An accelerated evaluation for AVs using piecewise mixture models is presented in [11]. The method addresses major issues with todays best practice adopted by the automotive industry. The proposed method uses a enhanced statistics of the surrounding vehicles and the importance sampling theory. Despite the fact that the proposed method can reduce the evaluation time, their method is limited in terms of implementation and it only ensure that the evaluation results are statistically accurate.

One of the main methods in robot testing, the procedural content generation (PCG) [13], also tries to overcome the challenge of time-consuming and costly manual test creation. PCG is used in fault discovery for robot control software [14] and is considered to have the potential to be adapted for AV software verification since it generates environments and creates, executes, and prioritizes the scenarios. However, these approaches suffer from the fundamental issue that the software can be built correctly, but actually do the wrong function.

Khastgir et al. [15] proposed an automated constrained randomized test scenario generation framework for ADAS and AVs. The framework first creates random-based scenarios by varying the conditions such as environment variables or vehicle trajectories in a driving simulator. Then, each base case is randomized in real-time by using constrained randomization again during the test to find the edge cases. Bagschik st al. [12] propose a knowledge based scene generation for AVs. The process of scenario generation is a big challenge in above mentioned methods. A framework is proposed by authors in [16] to improve the process of automatic test case generation for high delity models which has long execution times. The framework uses low delity models to evaluate certain properties analytically or computationally with provable guarantees. Although, this improves the process, the test scenario coverage is an important challenge for both of these proposed framework.

Li et al. [17] proposed a framework to combine scenario-based and functionality-based testing methodologies. The framework defines a new semantic diagram for driving intelligence using the relationship among testing scenarios, tasks, and functionalities. In that diagram, scenarios and functionalities were shown as two transverses with opposite directions and used to test the AVs in simulation. Li et al. [17] also designed a simulation platform that creates a digital twin of the real testing ground to record and reproduce the behaviors in real life. The focus of the work is to optimize test construction from a cost and efficiency point-of-view. However, evaluation of the edge test scenarios and a framework for coverage remains as an important challenge for this framework.

Heinz et el. [18] verified safety benefits of automated driving functions leveraging two defacto open standards (i.e. OpenDRIVE and OpenSCENARIO) for describing road networks and dynamic content in driving simulation. Their proposed method focuses on building interesting scenerio tests within the constraints of physical test resources. While this is a very useful effort, it does not address the issue of overall safety.

Shai et al. [19] proposed a scalable sensing system and safety standards for AVs, based on a classic sense-plan-act robot control architecture, which is able to maximize safety through minimizing sensing and planning errors. They presented a data fusion technique which enabled them to validate sensing errors with significantly smaller amount of data. To minimize planning errors, they proposed a responsibility-sensitive model which helps to identify the responsible entity in case of an accident. Safe longitudinal distance and safe lateral distance were the two major variables considered for evaluating safety benefits of AVs. This is an important measure of safety, however the conditions under which these might be violated are not addressed.

Overall, the core contributions of this paper and the FLPolyVF framework are differentiated from the existing solutions by providing a coherent verification framework which is enabled with the FLPolySA abstraction. The proposed framework addresses all above mentioned challengs.

IV. Scenario Mathematical Model for FLPolySA

A. Overview

In this section, we will formally define FLPolySA. This formal definition is critical in order to reason at this level of abstraction. Figure 3 shows the core "engine" of the abstraction which consists of a Newtonian evaluation whose inputs are the current scene/situation, the performance characteristics of the UUT, and the desired action of the UUT. The combination of these produces the next time step scene.

The remainder of this section define structures for FLPolySA

- Assertions: These layer the concept of pass/fail.
- Equivalence Classes: These provide the basis for coverage.
- Concatenation: These provide the rules for combining scenerios.

All of these concepts form the core elements for a discrete event engine which can be built in a virtual simulation environment. It should be noted that while we are focused on vehicle dynamics, the mathematics are not limited to AVs. We observe that the same infrastructure can be used for marine, drones, and even pedestrian robots.

B. Preamble

We will consider a euclidean three-dimensional space, with time as one additional dimension. We will be using the Real Cartesian coordinate system where the position, velocity and acceleration of all moving objects are determined by three dimensional vector valued functions with respect to time [20].

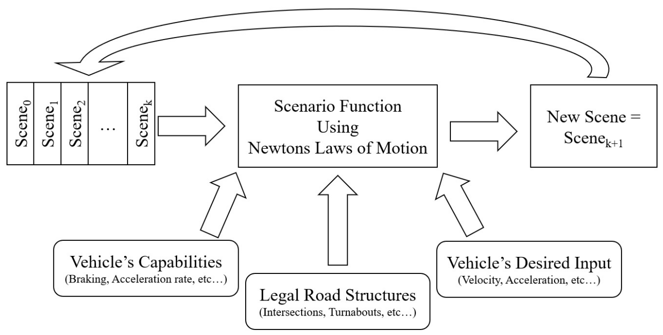
Scene$_0$
Scene$_1$
Scene$_2$
...
Scene$_k$
Scenario Function Using Newtons Laws of Motion
New Scene = Scene$_{k+1}$
Vehicle's Capabilities (Braking, Acceleration rate, etc...)
Legal Road Structures (Intersections, Turnabouts, etc...)
Vehicle's Desired Input (Velocity, Acceleration, etc...)

Fig. 3: FLPolySA Overview

All motion and mechanics of motion in our space will follow the basic Newtonian Laws of Motion. That is, all motion is described using the centers of mass of the actors and the UUT. However, the *Relative Distances* (RD) between the actors, constants and the UUT will be measured edge to edge.

We let the map $\mathbf{r}(t) : \mathbb{R} \longrightarrow \mathbb{R}^3$ be the position function for a moving or static object in our space then $\mathbf{v}(t) = \dot{\mathbf{r}}(t)$ and $\mathbf{a}(t) = \ddot{\mathbf{r}}(t)$ are the velocity and acceleration vectors respectively. Further more, we will use Newton's Principle of Determinacy which states that all motions of a system of, say n, objects are uniquely determined by their initial positions and initial velocities.

In addition, since all the objects in this environment will be treated as bounded rectangular boxes, then the relative edge to edge distance is computed as a distance between two bounded planes (including their edges). More explicitly, suppose we have two rectangular three dimensional objects O and T and let $O_1 : a_1x + b_1y + c_1z + d_1 = 0$ be one of the sides of O and $T_1 : A_1x + B_1y + C_1z + D_1 = 0$ be one of the sides of T where the x, y *and* z are bounded for both planes. Also, let $P(x_1, y_1, z_1)$ be any point on the plane O_1. Then we can calculate the distance between the two sides O_1 and T_1 as follows;

$$\begin{aligned} D(O_1, T_1) &= \min_{P \in O_1} \{\textit{Distance between } P \textit{ and the plane } T_1\} \\ &= \min_{P \in O_1} \left\{ \frac{|A_1x_1 + B_1y_1 + C_1z_1 + D_1|}{\sqrt{A_1^2 + B_1^2 + C_1^2}} \right\}. \end{aligned} \tag{1}$$

Thus we will define the relative edge to edge distance between O and T (denoted by $RD(O, T)$) as:

$$RD(O, T) = \min\{D(O_i, T_j) \mid 1 \leqslant i \leqslant 6, 1 \leqslant j \leqslant 6\}.$$

C. Scene Definition and Properties

Definition 1. *A **Scene Vector** $\mathbf{C}_k$ is a vector of real numbers representing the 3-D spherical environment around the Vehicle or UUT within N_k units of distance at a moment of time (or time-step) $k = t\Delta t$. The distance N_k will be called the **Radius of the scene vector**. Furthermore, a scene vector is broken down to four main components or sub-vectors:*

1) *The parameters describing the Ego or UUT; such as its dimensions, position, velocity and acceleration vectors, etc.*
2) *The dynamic actors; which are the moving components in this sphere. Each dynamic actor has its own velocity and acceleration vectors as well as the relative distance between the actor and the UUT.*
3) *The constants or static components; which would include the relative distance between the UUT and any non-moving object or structure in the scene in additions to the dimensions of the object.*
4) *The communication between the UUT and the other actors and components in the scene; which would include physical, electronic and wireless communications between the actors and other components in the scene such as an actor signalling to change lanes in front of the UUT.*

Definition 2. *A **Scene** $\varkappa$ is the 3-D environment constructed by accumulating the scene vectors with consecutive time steps up to the present time step. where*
$N = \max\{$*scene radii for all the scenes vectors up to the present time step* $k\}$

We model the **A Scene** $\varkappa$ as;

$$\varkappa = \left[\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_{k-1}, \mathbf{C}_k\right]$$

Remarks:

- The scene vectors forming the columns of a scene matrix can possibly have different sizes. Therefore, we will fill in the empty entries with zeros and define the size of the scene matrix to be:

$$\max\{\textit{number of rows in } C_i \mid i = 0, \ldots, k\} \times (k+1).$$

- The scene vector $\mathbf{C}_k$ is computed at a specific time step and it represents the scene as a snap shot of the reality around the UUT at that time step. A scene represents the space-time (4 dimensional space) around the UUT up to the current point of time.
- N is chosen to be large enough to capture effects near the UUT, but small enough for all of the reasonably small number of equivalence classes.

D. Scenario Definition and Properties

With the basic components of the model and their behaviour defined, we now integrate mathematically the intersection of basic physics with decision-making by the UUT. To be clear, between decision points, Newtonian physics is the driving paradigm, and at decision points, the UUT can change desired behaviours.

Definition 3. *Given a scene $\varkappa$, the **Next State function** ς is a function with input of the current scene vector, communication from other actors and the desired action of the ego and it's* *output a newly computed scene vector following the Newtonian laws of motion. That is, suppose $\mathbf{C}_k$ is the scene vector at time step k. Then we define the new vector $\mathbf{C}_\varsigma = \mathbf{C}_{k+1} = \varsigma(\mathbf{C}_k,$ Ego's Desired Input, Communication from other actors$)$ to be the next state vector. A **Scenario** occurs when the ego makes a decision. That is, when the ego wants to change its parameter (mainly velocity and/or acceleration vectors) to new **Desired Inputs**.*

Notice that, the communication from other actors in the scene can be in many formats such as a turn signal of another vehicle or the flashing lights from an emergency sign of vehicle. Also, the **Desired Inputs** which would be new values for the the velocity and/or acceleration vectors of the ego cannot happen suddenly.

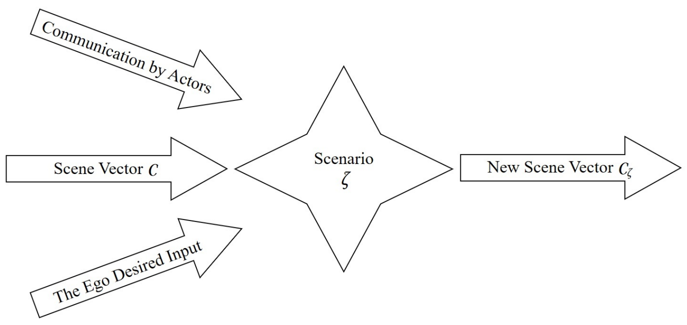

Fig. 4: A Scenario as a function of current and desired inputs.

$\mathbf{C}_\varsigma$ will have the new values for the parameters/variables in the scene vector based on the actions of the UUT within the capabilities of the UUT. So that, $\mathbf{C}_\varsigma$ would incorporate the decisions made by the UUT at the Scenario in the scene. Therefore, $\mathbf{C}_\varsigma$ could have many entries identical to $\mathbf{C}_k$ except it will also have entries, such as velocities and relative distances, computed using the basic laws of motion and based on new vectors for the velocity and acceleration (both longitudinal and latitudinal) depending on whether the UUT increased of decreased it's speed or changed it's direction or lane.

For instance, assume we have a scene in which the UUT will be driving along a road and at time step t an obstacle (such as a stop sign) is detected in the path of the UUT. Here we have a scenario, call it ς, where the UUT will make a change to its current velocity and/or acceleration vectors to avoid collision. These changes that the ego desire cannot happen suddenly (for example, the ego cannot come to an immediate stop, or change lanes instantly). Suppose the vector $\mathbf{C}_k = [r(k), v(k), a(k), *]$ is a scene vector at time step k where r, v and a are the position, velocity and acceleration vectors of the UUT and $*$ represents the other entries. Also, assume the ego's desired action is to stop then its input is $v = 0$. Thus,

$$\varsigma\Big([r(k), v(k), a(k), *], \text{Communication by actors}, v_{desired}\Big) = \mathbf{C}_\varsigma = \mathbf{C}_{k+1}.$$

The new scene vector $\mathbf{C}_\varsigma$ is computed by using Newtonian laws of motion taking in as input the current location, velocity and acceleration as functions of time and all other needed input from the original scene vector $\mathbf{C}_k$ and the desired action of the ego, be it change in speed or direction, and produces realistic values for all the variables and parameters such as a realistic stopping distance or change in direction vector using formulas such as:

$$\text{Deceleration rate} = D(k) = 2\left[\frac{(\textit{Braking Distance}) - v(k)}{k^2}\right], \tag{2}$$

where

$$\textit{Braking Distance} = \frac{v^2(k)}{2\mu g}, \tag{3}$$

and μ is the coefficient of friction between the road surface and the tires and g is gravity. Hence, if we denote the new position and new velocity vector $r_\varsigma(k), v_\varsigma(k)$ respectively, we have;

$$\mathbf{C}_\varsigma = [r_\varsigma(k), v_\varsigma(k), D(k), **].$$

E. Assertion Definition and Properties

To this point, we have defined the inner core as shown in Figure 1 around the general intersection of Newtonian physics, vehicle dynamics, communication, and decision-making by the UUT. To this point, there is no concept of good/bad or pass/fail. Assertions allow us to overlay these concepts on the model.

Definition 4. *Given a scene $\varkappa$ with a radius N, we define the **Assertion function** ϑ as a function with the domain being the set of scene matrices and the range is the interval $[0, 1]$ which has a predetermined set of weighted assertions. The output of this function is a probability (or a percentage) calculated as weighted average based on the predetermined assertions where an output of 0 means the UUT fails and an output of 1 represents the UUT passing.*

As previously mentioned, we consider the relative distances between the UUT and any other actors (dynamic or constant) as an edge to edge distance. Therefore, we identify the threshold constant as being a drivable surface by asserting that the relative distance between the UUT and that actor is zero and can't be positive. Otherwise, the vehicle is floating above the road which is a non-realistic situation. Similarly, we identify the nondrivable surfaces such as sidewalks by asserting that the distance between them and the UUT must remain strictly positive. That is, if the relative distance between the UUT and the sidewalk (for instance) becomes non-positive, the assertion must then fire.

The Formulation:

Let $\varkappa$ be a scene with radius N given by a matrix of size $n \times (k+1)$ where k is the current time step. Now assume that there are m assertions placed on $\varkappa$, let A be an $m \times n \times (k+1)$ matrix called ***the Assertion Matrix***. The j^{th} layer of A is an $m \times n$ matrix which corresponds to the j^{th} column (scene vector) in

$\varkappa$ where each row represents an assertion and each column represents one of the parameters in the scene vector. In other words, the entries in each row are the weights representing the relation between the assertion and the parameters in the scene vectors. Next, we define the **Assertion function** ϑ as follows:

$$\vartheta(\varkappa) = A\varkappa - \varkappa_0,$$

Here the product $A\varkappa$ is obtained by multiplying the j^{th} layer of A by the j^{th} column of $\varkappa$. In addition, $\varkappa_0$ is an $m \times 1 \times (k+1)$ matrix with each layer consisting of the vector (denoted by $C_{0,j}, j = 0, \ldots, k$) of acceptable values of the parameters for each assertion for each time step.

Notice that, since we are using matrix operations, then we can conclude that the **Assertion function** ϑ is a well defined function. This is very important, because we would like a verification system which is *predictably repeatable*.

Suppose we have a scene then we can attach an assertion to almost every actor in the scene. For example, we define the assertion that the UUT shall not collide with any obstacle or other actors in the scene which means that the *Relative Distance* to the obstacle or actor must remain strictly positive. In other words, we use the assertions per entity as observers to the UUT and the scene as a whole to give meaning to anything that might happen.

All the assertions placed on any scene can be formulated as functions of actual physical parameters of the scene. For instance, one could use the safe distances, as defined in [19], as parameters (entries) in the scene vector. Thus, by comparing these computed distance values using the actual velocity and acceleration of the UUT and the other actors with known safe and legal distances we will have a measure of passing or failure of the UUT.

F. Equivalence Classes

The scenes are represented by real vectors. Then it follows immediately that two scenes are equivalent if they have the same constant radius and their corresponding scene vectors are equivalent.

Remarks:
Assume two scenes ε and ϱ with the same radius N are equivalent. Then:

- For each actor in a scene vector, the distance between that actor and the UUT is an entry in the scene vector. Hence for each actor in one of the scenes $-$ say ε $-$ there is an actor in the other scene $-$ ϱ $-$ such that both actors are the same distance away from the UUT. Otherwise, the vectors will not be equivalent since the entries in the vectors corresponding to the distances between the actors and the UUT would be different.
- If at some time step k in a scene $-$ say ϱ $-$, that is for a scene vector represented by $\mathbf{C}_k$ a situation happened and a decision was made by the ego, in other words a scenario ς was generated. That will produce a new scene vector $\mathbf{C}_\varsigma$. Here, unless the vectors $\mathbf{C}_\varsigma$ and $\mathbf{C}_k$ are equivalent then the scenario ς would have moved us from one equivalence class to another.
- Due to the limitations of reality and the finite number of actors which a scene can have, the number of equivalence classes for each particular value of the radius N is finite. Moreover, there are a finite number of values of N that we would need in real life.

In a pseudo random test generation paradigm, it is important to efficiently build new tests. In this paradigm, this means building tests in different equivalence classes. Fortunately, with our formulation, we can use the Gram Schmidt Algorithm [21] to accelerate this process.

G. Combining Scenes and Scenarios

In a real life "trip", the AV will move through different road environments (such as in-town, highway, etc.). Therefore, it is necessary to define how one can transition smoothly from one scene to another. To that end we define the operator which will allow us to concatenate scenes in a continuous manner;

Definition 5. *Let ϱ and ε be two scenes vectors with radii N_ϱ and N_ε respectively. We define the operator $\circledS$ as follows:*

$$\Delta = \varrho \circledS \varepsilon \tag{4}$$

where the concatenation of the scenes satisfying the following:

- *Let $\mathbf{r_1}(k)$ and $\mathbf{r_2}(k)$ are the position vector functions in ϱ and ε respectively, and let $k = k_0$ be the moment in time where the transition from ϱ to ε will happen. Then $\mathbf{r_1}(k_0) = \mathbf{r_2}(k_0)$.*
- *The situations that involve immediate sharp turns or shifting from, say, two lanes to three are impossible in real life. The transition operator $\circledS$ must connect $\mathbf{r_1}$ and $\mathbf{r_2}$ using legal road structures. That can be accomplished by inserting a scene (or more) with a small radius between ϱ and ε. In addition, we use smoothing functions to ensure that the roads are connected* smoothly *(that is, the edges of the roads are connected) on the time interval $(k_0-\epsilon, k_0+\epsilon)$ where ϵ is a suitable fixed positive constant.*
- *When concatenating 2 scenes, say $\Lambda_1 = \big[\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_k\big]$ and $\Lambda_2 = \big[\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_\tau\big]$, the $\circledS$ operator must connect the last column of the first scene with the first column of the second scene. That is accomplished by identifying $\mathbf{D}_0 = \mathbf{C}_{k+1}$. In addition, we assume that the scenes are not overlapping (or might have a slight overlap).*

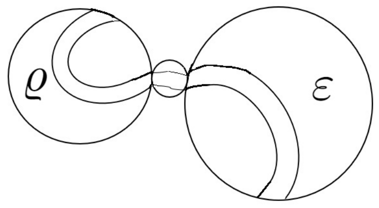

Fig. 5: concatenation of the scenes

Let $\Lambda_1 = \big[\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_k\big]$ be a scene matrix of size $n \times (k+1)$ and $\Lambda_2 = \big[\mathbf{C}_{k+1}, \mathbf{C}_{k+2}, \ldots, \mathbf{C}_\tau\big]$ be a second scene matrix with size $l \times (\tau - k)$. Assume also that Λ_1 has the assertion function ϑ_1 with the assertion matrix A of size $m \times n \times (k+1)$ and Λ_2 has the assertion function ϑ_2 with the assertion matrix B of size $p \times l \times (\tau - k)$. Now, let $\Delta = \Lambda_1 \circledast \Lambda_2$.

Next, we define the new operator $\diamondplus$ and *Concatenations Assertion Function* ζ as follows:

$$\begin{aligned}
\zeta(\Delta) &= [\vartheta_1(\Lambda_1) \diamondplus \vartheta_2(\Lambda_2)] \\
&= \big[(A\Lambda_1 - \Lambda_{1,0}) \diamondplus (B\Lambda_2 - \Lambda_{2,0})\big] \\
&= \Big[\big[(A_0\mathbf{C}_0 - C_{1,0}), \ldots, (A_t\mathbf{C}_k - C_{1,k})\big] \\
&\qquad \diamondplus \big[(B_0\mathbf{C}_{k+1} - C_{2,k+1}), \ldots, (B_\tau\mathbf{C}_\tau - C_{2,\tau})\big]\Big]
\end{aligned}$$

where A_j and B_j are the $j-$th layers of the assertion matrices and $C_{x,y}$ is the vector of acceptable values corresponding to the scene vector $\mathbf{C}_y$. The result of the $\diamondplus$ operator will be a large multi-layer matrix listing all the results with respect to the time step.

For example, if both scene matrices consist of a single column, i.e. $\Lambda_1 = \big[\mathbf{C}_1,\big]$ and $\Lambda_2 = \big[\mathbf{C}_2\big]$ with m and p assertions respectively, then:

$$\begin{aligned}
\zeta(\Delta) &= [\vartheta_1(\mathbf{C}_1) \diamondplus \vartheta_2(\mathbf{C}_2)] \\
&= \big[(A \cdot \mathbf{C}_1 - C_{1,0}) \diamondplus (B \cdot \mathbf{C}_2 - C_{2,0})\big]
\end{aligned}$$

The output $\zeta(\Delta)$ is a two-layer matrix with size $(m+p) \times 1 \times 2$ column matrix listing the output of $\vartheta_1(\mathbf{C}_1)$ and then $\vartheta_2(\mathbf{C}_2)$ with zeros filling in all of the extra entries.

$$\textit{The Result} = \begin{bmatrix} A_1\boldsymbol{C}_1 - C_{1,0} \\ A_2\boldsymbol{C}_2 - C_{2,0} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ \vdots \\ c_1 \end{bmatrix} - \begin{bmatrix} \textit{acceptable } a_1 \\ \textit{acceptable } b_1 \\ \vdots \\ \textit{acceptable } c_1 \end{bmatrix} \\ \begin{bmatrix} a_2 \\ b_2 \\ \vdots \\ c_2 \end{bmatrix} - \begin{bmatrix} \textit{acceptable } a_2 \\ \textit{acceptable } b_2 \\ \vdots \\ \textit{acceptable } c_2 \end{bmatrix} \end{bmatrix}$$

Fig. 6: The result of the verification function $\zeta(\Delta)$.

Notice that;

- ζ and the $\diamondplus$ operator are well defined since the result is essentially a matrix listing the results of $\vartheta_1(\mathbf{C}_1)$ and then $\vartheta_2(\mathbf{C}_2)$.
- The scene vector $\mathbf{C}_2$ is treated as a result of a scenario function which occurs at the time step right after $\mathbf{C}_1$. That is, $\mathbf{C}_2$ is a function of the time-step at which the transition from Λ_1 to Λ_2 happens. In other words, concatenating two scenes creates a scenario between the scenes.

H. Discussion of Key Properties and Future extensions

At this point, it is worthwhile to summarize the impact of this foundational mathematical work and point to the direction of future extensions of this work. With the work to date, a model of computation has been created with clearly integrates Newtonian Physics and decision making from the ego car. Further, this is done in a manner which allows for easy inclusion of vehicle properties. This core baseline formulations the world of Physics.

Above the world of Physics sits the world of humans with notions of success and failure. These are cleanly captured with the definitions of assertions and the intersection of assertions with the physics model yields an overall pass/fail score. Along with the assertion infrastructure, there exists an infrastructure for detecting equivalent tests through the equivalent class structure. Note, that all items are defined with a scope (called the Radius) which enables a much higher level of reuse. This method of definition enables the creation of a coverage matrix. Finally, the concatenation operators allows for the building of larger trips by combining atomic tests.

Overall, this MOU answers the questions of conceptual model, test regime, completeness, and accumulated learning outlined in the introduction. Further, since all the tests are non-responsive, the accumulation of the tests can be used in a sign-off regime.

With a formal definition, in the future, we envision the ability to use branch-and-bound algorithms such as those found in Automatic Test Pattern Generation [ref rahul's paper] to automatically generate "edge" test cases which are driven by attempting to trigger failures on the assertions. In a future paper, we will describe the mathematics of this Co-NP complete problem.

V. Implementation Example for Scenario Generation

In this section we will present a simple example to demonstrate the main computational steps.

Let us start with a scene of radius N, in which we have the UUT driving behind another car (actor 1). Assume that $r(k), v(k),$ and $a(k)$ are the position, velocity and acceleration vectors of the ego at time step k. Let $r_1(k), v_1(k),$ and $a_1(k)$ be the position, velocity and acceleration vectors of the of the actor at time step k. Also, let $d_1(k)$ be the distance between actor 1 and the UUT. In addition, suppose that the actor brakes suddenly at time step $k = 0$.

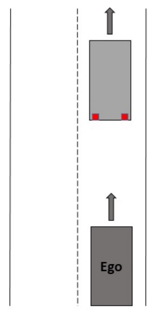

Fig. 7: Scene at time step $k = 0$.

Here we can construct the scene vector:

$$\mathbf{C}_0 = \begin{bmatrix} r(0) \\ v(0) \\ a(0) \\ N \\ d_1(0) \\ r_1(0) \\ v_1(0) \\ a_1(0) \\ * \end{bmatrix}$$

where the " $*$ " represents that other entries in the vector.

A scenario ς will take place at this time step. The input of ς will be $\mathbf{C}_0$ as well as the desired input of the ego, which would be reducing the speed in order to maintain safe relative distance d_1.

$$\varsigma\big(\mathbf{C}_0, v_{desired}\big) = \mathbf{C}_\varsigma = \mathbf{C}_1 = \begin{bmatrix} r(1) \\ v(1) \\ a(1) \\ N \\ d_1(1) \\ r_1(1) \\ v_1(1) \\ a_1(1) \\ ** \end{bmatrix}$$

Here the scene can be constructed as follows:

$$\varkappa = \Big[\mathbf{C}_0\mathbf{C}_1\Big] = \left[\begin{bmatrix} r(0) \\ v(0) \\ a(0) \\ N \\ d_1(0) \\ r_1(0) \\ v_1(0) \\ a_1(0) \\ * \end{bmatrix} \begin{bmatrix} r(1) \\ v(1) \\ a(1) \\ N \\ d_1(1) \\ r_1(1) \\ v_1(1) \\ a_1(1) \\ ** \end{bmatrix}\right]$$

Observe that there is nothing inherently good or bad about the previous vectors and their entries. They are simply physical quantities describing motion in 3D space. Next, we consider the *assertions* that will enable the labeling of the behaviour as pass or fail:

- The velocity of the ego $v(k)$ does not exceed the speed limit v_{limit} for any time step k.
- The relative distance between the UUT and the actor is less that the radius of the scene.
- The relative distance, $d_1(k)$, exceeds the minimum safety distance d_{min} as defined in [19].

$$d_{min} = \Bigg[v_r\rho + \frac{1}{2}a_{max,accel}\rho^2 + \frac{\big(v_r + \rho a_{max,accel}\big)^2}{2a_{min,brake}} - \frac{{v_f}^2}{2a_{max,brake}}\Bigg]_+$$

where v_r is the velocity of the rear car (the ego) and v_f is the velocity of the car in the front (the actor).

Now, we define the assertion function ϑ on the scene $\varkappa$ as follows:

$$\vartheta(\varkappa) = \Big[\big[A_0\mathbf{C}_0 - C_{0,0}\big]\big[A_1\mathbf{C}_1 - C_{0,1}\big]\Big]$$

where A_0 and A_1 are the 2 layers of the assertion matrix A. Here, we can use

$$A_0 = A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix},$$

$$C_{0,0} = \begin{bmatrix} v_{limit} \\ N \\ d_{min} \end{bmatrix}, \text{ and } C_{0,1} = \begin{bmatrix} v_{limit} \\ N \\ d_{min} \end{bmatrix}.$$

Thus

$$\vartheta(\varkappa) = \left[\begin{bmatrix} v(0) - v_{limit} \\ d_1(0) - N \\ d_1(0) - d_{min} \end{bmatrix}, \begin{bmatrix} v(1) - v_{limit} \\ d_1(1) - N \\ d_1(1) - d_{min} \end{bmatrix}\right]$$

Note that we are always checking if the distance between the actor and the UUT is more than the radius N since if that is the case then the actor is no longer of any consequence in the scene and can be ignored. Furthermore, the UUT would have failed any of the assertions if the corresponding entry is positive or negative depending on the setup. For this particular example, the UUT would be considered as passed if the entries in $\vartheta(\varkappa)$ have the following signs:

$$\left[\begin{bmatrix} - \\ - \\ + \end{bmatrix}, \begin{bmatrix} - \\ - \\ + \end{bmatrix}\right].$$

VI. Software Implementation

The scenario testing framework and model of computation described in this paper must be implemented in a simulation system, which offers the characterization of the physical world, test scenario generation and coverage analysis. Therefore, the scenario generation software is going to be the core of the practical testing functionality that supports the model of computation. The framework generating the tests will be providing high performance computing, high data storage, high network bandwidth and simulation execution. The simulation platform is going to be accompanied by several software tools for the simulation and control, which include rendering, debugging and analysis software.

The simulation model architecture in Figure 8 provides the framework for the implementation of FLPolyVF conceptual layers given in Figure 1. 'Newtonian Physics Layer' in the conceptual model is provided by the simulation tool's main physics engine and it specifies the basic structure of the scenario environment. This consists of objects with mass, rectilinear bounding-box shapes and vector trajectory. This method of implementation allow focusing only on the necessary components for realization of the model of computation.

The 'Assertions Layer' is implemented by functional blocks that can be adjusted by the simulation designer according to the scenarios. The 'Design for Experiments' include instantiating critical inputs and seeding them using constrained random generation. These functions are implemented by the main program, simulation controls and scene generation.

The 'Inputs and Constraints' is the active control layer of the simulation architecture, consisting of critical inputs provided for every object such as the trajectory or the acceleration.

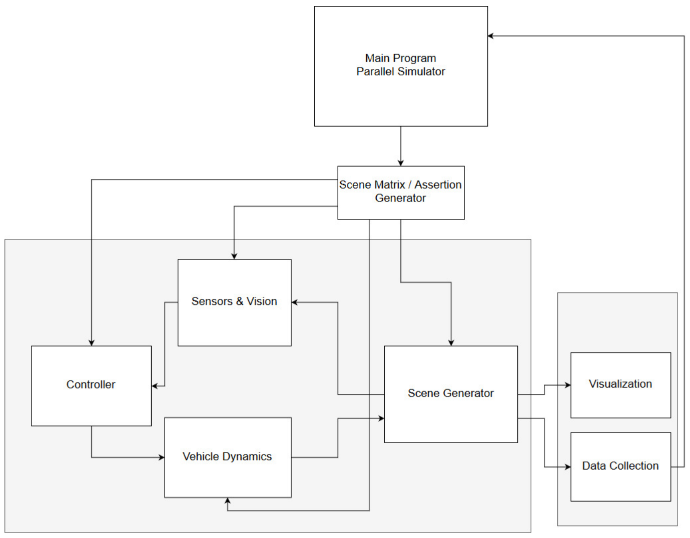

Fig. 8: Simulation model architecture

The simulation framework uses MATLAB Automated Driving System Toolbox Release R2018b [22] as its foundation for the proof-of-concept implementation and extends it for the necessary FLPolyVF-specific functionality.

The software implementation of the simulation framework is divided into several functional blocks as shown in 8. The 'Main Program' block of the simulation model serves as an interface for the user when running the simulations. This interface allows users to control their simulation preferences and set the number of runs. The 'Main Program' block also manages programmable generation and classification of scenarios. Hence, when the scenarios are automatically generated, the system classifies them by their characteristics.

The 'Scene Matrix/Assertion Generator' block takes input from the main program to run a simulation. The matrices represent the scenario inputs and these inputs provide various attributes to each object in the scenario, including subcategories such as location and orientation. The system provides options to create matrices by either using the matrix generator, or manually from the main program. When the parameters are set for a simulation, they are passed as input to the 'Scene Generator' block for the definition of scenes.

The 'Scene Generator' block in the simulator takes the matrices defined in the 'Scene Matrix / Assertion Generator' block and first stitches together a road scene. Then the vehicles and other actors in the scene are placed in this road scene. The simulation keeps track of the inputs to the scene and records them along with the results of the simulation such as the passing or failing of the vehicle or any other details surrounding the scenario. This information is passed back to the main program as feedback.

A sample scenario created in our simulation system is given in Figure 9. The scenario contains two actors; the vehicle under test and another additional actor. In this simple simulation scenario, a vehicle cuts close in front of the ego vehicle.

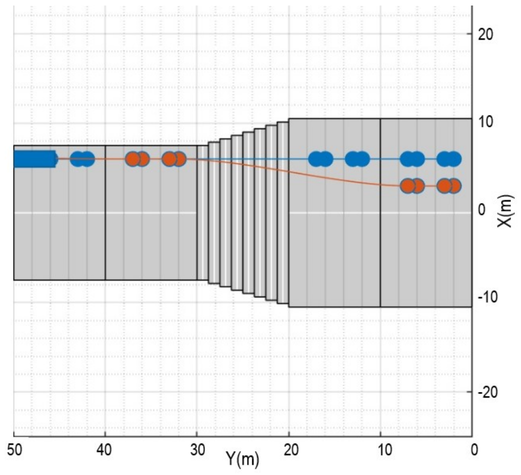

Fig. 9: Simulation scenario example

There are two input matrices to define this scenario, the road matrix, $rMatrix$, and the actor matrix, $aMatrix$. The $rMatrix$ represents the conditions of the road and the $aMatrix$ represents the actors on the road. In order to define our example scenario, $rMatrix$ and $aMatrix$ are created as follows:

$$rMatrix = \begin{bmatrix} 1 & 20 & 3 & 2 & 1 & 1 & 50 & 0.3 & 0 \\ 1 & 20 & 2 & 2 & 1 & 1 & 50 & 0.3 & 0 \end{bmatrix}$$
$$aMatrix = \begin{bmatrix} 1 & 2 & 1 & 55 & 1 & 1 & 1 & 0 \end{bmatrix}$$

In the road matrix, each index represents a particular part of the condition in the simulation. These are the number of road pieces, road piece length, number of lanes in the road piece, the lane of the ego vehicle, bidirectional or unidirectional information, existence of a mid lane, speed limit and road slickness angle. The number of rows in the $rMatrix$ depends on the number of pieces on the road. Since the example scenario consists of two road pieces, these pieces are stitched together in the simulation.

The actor matrix, $aMatrix$, contains information about the actors on the road. The actor information in our initial simulation settings consists of actor type, path type, car type, moving speed, dimensions and starting location. In the $aMatrix$, the fifth index is also a matrix that describes the dimensions of the actor.

The implementation example demonstrates the potential capabilities of the implementation methodology to use the model of computation and also the usability of the model of computation for practical testing purposes. However, it is important to note that the current simulation model will be extended to fully cover the requirements of the model of computation.

VII. Conclusion

The AVs are making their way into our lives. Different companies in the transportation industry are racing to put these vehicles on the road. There are many potential benefits of using AV technology such as providing safe rides and reducing traffic congestion. However, AVs have to be verified before they hit the roads.

The major contributions of this paper are proposing the FLPolyVF framework and the introduction of the scenario level of abstraction, which is at the center of this framework. The FLPolyVF framework connects functional verification, sensor verification, diagnostics and industry/regulatory communication with the scenario abstraction level. The scenario level follows a four layered scheme: the Newtonian physics layer, the assertions, the design for experiment and the constraints on the input. A mathematically consistent definition is offered for the scenario level of abstraction which includes the mathematical machinery to support higher level functional flows. Finally, with the definition of the FLPolyVF framework and the scenario abstraction, we are building the experimental framework (using MATLAB) to implement the abstraction.

References

[1] K. Brubaker, *Artificial Intelligence: Issues of Consumer Privacy, Industry Risks, and Ethical Concerns*. PhD thesis, Utica College, 2018.

[2] M. Bertoncello and D. Wee, *Ten ways autonomous driving could redefine the automotive world*. PhD thesis, mckinsey.com.

[3] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[4] J.-L. Boulanger *et al.*, "Requirements engineering in a model-based methodology for embedded automotive software," in *Research, Innovation and Vision for the Future, 2008. RIVF 2008. IEEE International Conference on*, pp. 263–268, IEEE, 2008.

[5] W. Lee, S. Park, and M. Sunwoo, "Towards a seamless development process for automotive engine-control system," *Control Engineering Practice*, vol. 12, no. 8, pp. 977–986, 2004.

[6] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[7] S. Hallerbach, Y. Xia, U. Eberle, and F. Koester, "Simulation-based identification of critical scenarios for cooperative and automated vehicles," tech. rep., SAE Technical Paper, 2018.

[8] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[9] J. Ma, F. Zhou, C. L. Melson, R. James, and X. Zhang, "Hardware-in-the-loop testing of connected and automated vehicle applications: A use case for queue-aware signalized intersection approach and departure," tech. rep., 2018.

[10] F. H. A. (FHWA), "Hardware in the loop testing of connected and automated vehicle applications: An update," tech. rep., 2017.

[11] Z. Huang, H. Lam, D. J. LeBlanc, and D. Zhao, "Accelerated evaluation of automated vehicles using piecewise mixture models," *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[12] G. Bagschik, T. Menzel, and M. Maurer, "Ontology based scene creation for the development of automated vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1813–1820, IEEE, 2018.

[13] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[14] J. Arnold and R. Alexander, "Testing autonomous robot control software using procedural content generation," in *International Conference on Computer Safety, Reliability, and Security*, pp. 33–44, Springer, 2013.

[15] S. Khastgir, G. Dhadyalla, S. Birrell, S. Redmond, R. Addinall, and P. Jennings, "Test scenario generation for driving simulators using constrained randomization technique," tech. rep., SAE Technical Paper, 2017.

[16] C. E. Tuncali, S. Yaghoubi, T. P. Pavlic, and G. Fainekos, "Functional gradient descent optimization for automatic test case generation for vehicle controllers," in *Automation Science and Engineering (CASE), 2017 IEEE International Conference on. IEEE*, 2017.

[17] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: A new approach," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 158–166, 2016.

[18] A. Heinz, W. Remlinger, and J. Schweiger, "Track-/scenario-based trajectory generation for testing automated driving functions," in *8. Tagung Fahrerassistenz*, 2017.

[19] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[20] C. Grosche, G. S. Pogosyan, and A. Sissakian, "Path integral discussion for smorodinsky-winternitz potentials: I. two-and three dimensional euclidean space," *Fortschritte der Physik/Progress of Physics*, vol. 43, no. 6, pp. 453–521, 1995.

[21] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[22] MATLAB, *MATLAB and Automated Driving System Toolbox Release R2018b*. Natick, Massachusetts, United States.: The MathWorks Inc., 2018.