GAS: Generating Fast & Accurate Surrogate Models for Simulations of Autonomous Vehicle Systems

Keyur Joshi*, Chiao Hsieh†,*, Sayan Mitra*, Sasa Misailovic*
* University of Illinois Urbana-Champaign † Kyoto University
{kpjoshi2, mitras, misailo}@illinois.edu hsieh.chiao.7k@kyoto-u.ac.jp

***Abstract*—Modern autonomous vehicle systems (AVS) use complex perception and control components. Developers gradually change these components over the vehicle's lifecycle, requiring frequent regression testing. Unfortunately, high-fidelity simulations of these complex AVS for evaluating safety are costly, and their complexity hinders the development of precise but less computationally intensive surrogate models.**

We present GAS, a novel approach for expediting simulation-based safety testing of AVS with complex perception and control components. GAS creates a surrogate of the *complete* vehicle model (i.e., those with complex perception, control, and dynamics components). The surrogates execute faster than the original models and are used to precisely estimate two key properties: the probability that the AVS will violate safety assertions and the bounds on global sensitivity indices of the AVS.

We evaluate GAS on five scenarios involving crop management vehicles, self driving carts, and unmanned aircraft. Each AVS in these scenarios contains a complex perception or control component. We generate surrogates of these vehicles using GAS and check the accuracy of the above properties. Compared to the original simulation, GAS models enable estimating the probability of violating a safety assertion 3.7 times faster on average and analyzing sensitivity 1.4 times faster on average.

I. Introduction

Autonomous vehicle systems (AVS) are becoming increasingly common. Besides self-driving cars, many autonomous vehicles perform specialized tasks (e.g., plant and animal monitors, self-driving carts in factories, and drones and other unmanned aircraft), drastically increasing the diversity of designs and applications. Yet, they all navigate by perceiving the vehicle's state (position, heading, etc.), making a control decision based on this *perceived* state, and moving according to the rules of physics. To ensure safety, developers specify properties that the vehicle must satisfy in certain scenarios (e.g., the vehicle stays on the assigned path). However, 1) many sensors have a nondeterministic output (e.g., GPS and LIDAR) and 2) the vehicle's software processes sensor values or makes decisions using complex, possibly imperfect components such as neural networks (NNs) and lookup tables. Given the uncertainty in the output of the perception and control systems, developers focus on showing that the vehicles will satisfy the safety properties *with high probability*.

Simulation-based testing of safety properties during development of AVS can detect software faults that would otherwise require much more expensive real-world tests [2, 15, 34, 38, 39] and are commonly used in industry for testing AVS [2]. *Monte Carlo Simulation* (MCS) is the most commonly used method for providing statistical guarantees when checking AVS safety properties. However, using MCS can be prohibitively expensive, especially for developers of specialized utility vehicles, who may not have the resources available to major members of the automotive industry. The presence of uncertainty in inputs necessitates a large number of resource-intensive system simulations to get sufficiently accurate estimates of the probability that the vehicle will violate safety properties [2, 29, 33]. This makes the use of MCS in regression test suites for vehicle systems impractical except in limited scenarios.

In many engineerings systems, *surrogate models* aim to provide an accurate and faster replacement for the original costly models [3]. Such surrogate models help developers configure and test the system in many ways: 1) checking application-specific assertions (e.g., whether a vehicle follows the path), 2) checking the robustness of the system to changes in the environment (e.g., weather changes, sensor noise levels), and 3) serving as an anomaly detector during fault injection studies in the original model/simulator.

A key concern when developing surrogate models for these tasks is that the surrogate should be a close representative of the simulated model w.r.t. the properties of interest. However, complex perception and control components – which dominate the vehicle model simulation time – hinder the application of existing surrogate model construction techniques (e.g., [26, 29]) to creating surrogate models of a *complete vehicle system*, which in the context of this paper is the system with complex perception, control, and dynamics components.

Our work. We present GAS (**G**PC for **A**utonomous Vehicle **S**ystems), the first approach for creating surrogate models of complete AVSs with complex perception and/or control components. The resulting surrogate models closely approximate the original vehicle models while being significantly faster, thus significantly reducing simulation cost when developers experiment with AVS components or tune system parameters.

GAS first creates a *perception model* to calculate the *distribution* of error in the output of the perception system for any ground truth state. GAS directly samples this error distribution, minimizing the need for costly experimentation with environmental parameters, image generation, and neural networks. Second, GAS constructs a surrogate model of the *complete* vehicle system (perception, control, and dynamics). In this paper, we primarily create surrogate models using Generalized Polynomial Chaos (GPC) [40] as this approach

Fig. 1: Real-life TerraSentia crop monitoring vehicle [9].

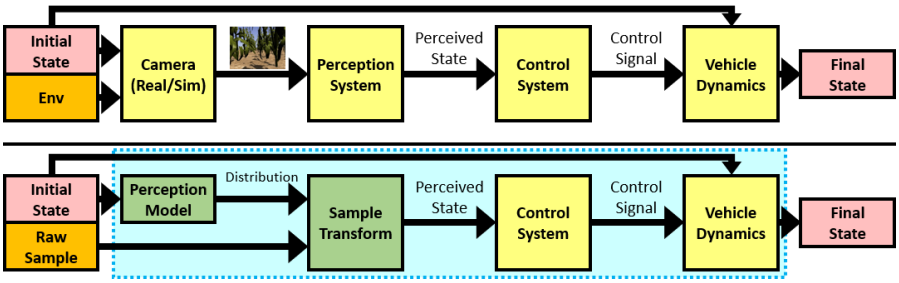

Fig. 2: Crop-Monitor vehicle: original model M_V (top) and abstract model M'_V (bottom). GAS *replaces* the outlined section with a surrogate model M_{GPC}.

produced the best results in our evaluation; GAS also supports other types of surrogate models (e.g., polynomial regression or small NNs). Because GAS's perception model is created independently of downstream components, GAS can reuse it when developers alter vehicle control and dynamics properties, saving significant time.

We demonstrate the advantages of using the GAS-generated surrogate model for estimating two key statistics of utility vehicle systems. First, we *estimate the probability that the vehicle will violate a safety property over time* in five realistic scenarios. These scenarios model systems used in crop management vehicles, self driving utility carts, and unmanned aircraft, with associated safety properties. Each system uses a complex perception component (ResNet-18 or LaneNet) or a complex control component (neural network controllers or lookup tables). We show that the probability of violating a safety property calculated by the surrogate model closely matches that calculated via the original model for 97% or more of time steps, while being $3.7\times$ faster on average (minimum $2.1\times$). Second, we use the surrogate model for *global sensitivity analysis of the vehicle system* to initial state perturbations for the same scenarios by calculating Sobol sensitivity indices [36] $1.4\times$ faster on average (minimum $1.3\times$) with an average error of 0.0004 (maximum 0.06). Lastly, we also investigate how various hyperparameters of the perception model affect the overall accuracy and speedup of GAS.

Contributions. This paper makes several contributions:

- **Surrogate models for complex autonomous vehicle systems.** We present GAS, a novel approach for creating fast, accurate surrogate models of complete autonomous vehicle systems with complex perception and control.
- **Perception models.** We present models that estimate the error distribution of complex perception systems, and use these perception models as part of our GAS approach.
- **Implementation.** We implement GAS as a tool which automates the creation of the perception model and the creation and usage of the surrogate model. GAS is available at https://github.com/uiuc-arc/GAS.
- **Evaluation.** We evaluate GAS on five realistic scenarios that model specialized autonomous agricultural vehicles, utility vehicles, and unmanned aircraft. GAS provides fast and accurate estimates of the probability that the AVS reaches an unsafe state and the global sensitivity indices of the AVS.

II. Overview

Consider an autonomous vehicle that travels between rows of crops to inspect them. We have adapted this scenario from the Terrasentia robot [9, 35]. Figures 1-2 illustrate the scenario.

The top half of Figure 2 shows a block diagram representation of the system model M_V responsible for driving the vehicle between two rows of crops. First, a camera captures the area in front of the vehicle. The image depends on the current vehicle state as well as environmental variables such as crop type, crop growth stage, and lighting conditions. A regression neural network (NN) analyzes the image to perceive the current vehicle state. The relevant state variables in this scenario are: 1) the *heading* angle h, which is the angle between the vehicle's current heading and the imaginary centerline between the two rows of crops, and 2) the *distance* d of the vehicle from the centerline. h can take values in $[-\pi, \pi]$ radians while d can take values in $[-0.38, 0.38]$ meters. The vehicle state space is therefore $\mathbb{D}_S = [-\pi, \pi]_h \times [-0.38, 0.38]_d$. As NNs are inherently approximate, this *perceived* state may not match the ground truth. The control system uses this perceived state to calculate a steering angle. Finally, the vehicle moves according to its constant speed and commanded steering angle.

Safety property. We wish to avoid two undesirable outcomes: 1) if $|d| > 0.228m$, the vehicle will hit the crops, and 2) if $|h| > \pi/6$, the NN output will become highly inaccurate. Since the vehicle makes control decisions based on approximate data, we cannot be certain that it will remain within the *safe state space* $\mathbb{D}_S^{safe} = [-\pi/6, \pi/6]_h \times [-0.228, 0.228]_d$. Instead, we want to answer the question: *What is the probability that the vehicle will remain safe over a period of time?* Moreover, we want to ensure that the proposed changes to the vehicle system do not lead to a sudden increase in the probability of reaching an unsafe state.

Monte Carlo Simulation. In *Monte Carlo Simulation* (MCS), we simulate the vehicle's movement a large number of times and count the number of times the vehicle violates a safety properties. We use Gazebo [23] for precise control over the simulation environment. We simulate 1,000 sample vehicles over 100 time steps of 0.1 seconds each. We randomly sample environmental conditions from an environment distribution $\mathcal{D}_E$ that contains two crop types, four crop growth stages, and various lighting conditions. We also choose the initial vehicle state from a normal distribution. At each time step, we count

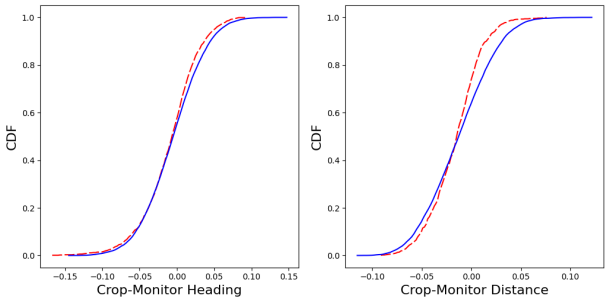

Fig. 3: State distribution at the final time step: heading (left) and distance from the centerline (right)

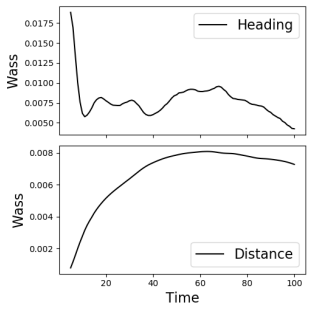

Fig. 4: Wasserstein metric (distribution similarity) over time

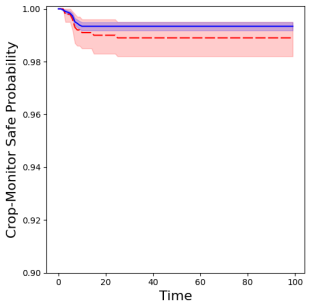

Fig. 5: Probability of staying in a safe state over time

how many samples are still in a safe state.

While such simulation is faster and cheaper than real-life tests and can still find issues in the vehicle system [2, 15, 34, 38, 39], it is still expensive enough that testing every proposed change to the system in this manner would be impractical.

A. GAS: using Generalized Polynomial Chaos

We present GAS, a novel approach for creating fast surrogate models of complex vehicle systems. Here, we show an example construction using Generalized Polynomial Chaos (GPC). Using GPC to create surrogate models of vehicle dynamics only as in previous work [22] is insufficient as image capture and processing contributes to over 99% of the simulation time. Instead, GAS creates a surrogate for the *entire* vehicle model – perception, control, and dynamics.

Perception model construction. The output of the vehicle's perception NN depends on the image captured by the front camera. This image depends not only on the vehicle's state, but also environmental variables. For a given distribution of environmental variables, there is a corresponding output distribution of the perception NN for each ground truth state. Based on real-world data from [44], we found that this NN output distribution is approximately Gaussian [1, Section A].

GAS must sample this output distribution when creating the GPC model. However, directly using the numerous environmental variables increases the input space over which GAS must construct the GPC model, which increases model construction time. GAS instead abstracts away the actual environmental variables by creating a *perception model*. The perception model does not have to be a perfect abstraction of the NN (creating such an abstraction is at least as hard as verifying the NN). Instead, GAS trains the perception model by selecting an 11×11 grid G of ground truth states in $\mathbb{D}_S^{safe}$. At each grid point $g \equiv (h, d)$, it randomly samples images from $\mathcal{D}_E$, and records the NN's outputs. It calculates the mean $\mu(g)$ and covariance $\sigma^2(g)$ of the outputs at each grid point. GAS trains a polynomial regression model M_{per} to predict each component of $\mu(g)$ and $\sigma^2(g)$, thus encoding the NN's output distribution at every state.

GPC Surrogate model construction. Next, GAS creates an abstract vehicle model M'_V (bottom half of Figure 2). M'_V first uses the perception model to obtain the NN output distribution in the current state. It next transforms a sample from a 2D standard normal distribution into a sample from this output distribution by multiplying by σ and adding μ. The rest of the vehicle model uses this transformed sample as the perceived state. GAS now uses GPC to create a 4^{th} degree polynomial model of *all* of M'_V (outlined section of Figure 2). GAS *replaces* the original model M_V in the MCS procedure described above with this surrogate model M_{GPC} to estimate the safe state probability of the vehicle over time. Because M_{GPC} is much faster to simulate than M_V, we also increase the number of samples for safe state probability estimation to 10,000 in order to reduce sampling error.

B. GAS results for crop monitoring vehicle

Accuracy. Figure 3 compares the heading and distance distributions after 100 time steps. The X Axis shows the variable value and the Y Axis shows the cumulative probability. The red dashed and blue solid plots show the distributions estimated using M_V and M_{GPC}, respectively. We compare the two distributions using the Kolmogorov-Smirnov (KS) statistic and the Wasserstein metric. Figure 4 shows how the Wasserstein metric (Y Axis) evolves over time steps (X Axis). The low Wasserstein metric value indicates good correlation between the two distributions at all times. The KS statistic also remains below 0.14. Figure 5 shows the probability of remaining in a safe state, i.e., not violating the safety property (Y Axis) over time steps (X Axis). The shaded regions around each plot show the 95% bootstrap confidence interval, which illustrates the extent of sampling error. As GAS evaluates M_{GPC} for $10\times$ more samples than M_V, its confidence interval is smaller. We use the t-test to check if the safe state probabilities are similar: it passes for 99 of 100 time steps, indicating high similarity.

Time. MCS using M_V required 19.5 hours on our hardware. To create the surrogate M_{GPC}, we first had to create M_{per}. Gathering the training data for M_{per} required 8.5 hours. The time required for training M_{per}, constructing M_{GPC}, and using M_{GPC} was negligible in comparison ($<$ 1 minute). Overall, this means that GAS is $2.3\times$ faster than MCS using M_V. Increasing the number of samples or time steps would further increase this speedup, since the time required to construct M_{per} is a one-time cost.

Incremental analysis and regression testing. When developers make changes to the vehicle control and dynamics systems, GAS can reuse M_{per}. Because gathering training data for M_{per} constitutes over 99% of GAS's analysis time, this reusability saves a significant amount of time when testing changes to these subsystems. GAS must create a new perception model if developers make changes to the perception system, but the results above show that creating a new perception model is still cheaper than using M_V. Crucially, if a change to the vehicle causes a sudden increase in the probability of violating a safety property, *this increase will also be reflected in the results of the GAS surrogate model*, thus providing developers with an approach for accelerating regressing testing of vehicle systems.

III. Background: Generalized Polynomial Chaos

We present key definitions pertaining to GPC. Dedicated books (e.g., [40]), discuss GPC in more details.

Orthogonal polynomials. Assume X is a continuous variable with support S_X and probability density $p_X : S_X \to \mathbb{R}$. Let $\Psi = \{\Psi_n | n \in \mathbb{N}\}$ be a set of polynomials, where Ψ_n is an n^{th} degree polynomial. Then Ψ is a set of orthogonal polynomials w.r.t. X if for all $n \neq m$, $\int_{S_X} \Psi_n(x)\Psi_m(x)p_X(x)dx = 0$. The orthogonal polynomial Ψ_n has n distinct roots in S_X. Orthogonal polynomials exist for several probability distributions. For example, the Legendre, Hermite, and Laguerre polynomials are orthogonal for the uniform, normal, and gamma distributions respectively.

Orthogonal polynomial projection (GPC). Let $f : S_X \to \mathbb{R}$. Then the N^{th} order orthogonal polynomial projection of f, written as f_N, w.r.t. a set of orthogonal polynomials Ψ, is:

$$f_N = \sum_{i=0}^{N} c_i \Psi_i \quad \text{where} \quad c_i = \frac{\int_{S_X} f(x)\Psi_i(x)p_X(x)dx}{\int_{S_X} \Psi_i^2(x)p_X(x)dx} \quad (1)$$

If f is an N^{th} degree polynomial, then $f_N = f$. Otherwise, f_N is the *optimal* N^{th} degree polynomial approximation of f w.r.t. X, in the sense that it minimizes ℓ_2 error, which is calculated as $\int_{S_X}(f(x) - f_N(x))^2 p_X(x)dx$. As $N \to \infty$, the ℓ_2 error approaches 0, that is, we can construct arbitrarily good approximations of f. f_N is called the *N^{th}-order generalized polynomial chaos (GPC) approximation* of f.

Lagrange basis polynomials. Given N points (x_i, y_i), $1 \leq i \leq N$, where all x_i are distinct, Equation 2 shows the *Lagrange basis polynomials* L_i for each i.

$$L_i(x) = \prod_{\substack{1 \leq j \leq N \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \quad (2)$$

Gaussian quadrature. To use Equation 1, we must perform multiple integrations to calculate the coefficients c_i ($i \in \{0 \ldots N\}$). For any non-trivial function g, we must use numerical integration by approximating the integral with the following sum:

$$\int_{S_X} g(x)p_X(x)dx \approx \sum_{i=1}^{N} w_i g(x_i); \quad w_i = \int_{S_X} L_i(x)p_X(x)dx \quad (3)$$

We choose w_i and x_i so as to minimize integration error. In *Gaussian quadrature*, we choose x_i to be the N roots of Ψ_N, the N^{th} order orthogonal polynomial w.r.t. X. We calculate the corresponding weights w_i using the Lagrange basis polynomials L_i (Equation 2) passing through $x_j \; \forall j \neq i$.

Multivariate GPC. GPC can be easily extended to the multivariate case if *all input variables are independent*. Let $\mathbf{X} = (X_1, \ldots, X_d)$ be the d independent random variables (not necessarily following the same distribution) and let f be a function over $\mathbf{X}$. The orthogonal polynomials $\mathbf{\Psi_i}$ for $\mathbf{X}$ are simply the products of the orthogonal polynomials $\Psi_{i_1}, \ldots, \Psi_{i_d}$ for $X_1, \ldots, X_d$ respectively. The GPC approximation closely resembles the one for the univariate case:

$$f_N = \sum_{\mathbf{i}} c_{\mathbf{i}} \mathbf{\Psi_i} \quad \text{where} \quad c_{\mathbf{i}} = \frac{\int_{S_\mathbf{X}} f(\boldsymbol{x})\mathbf{\Psi_i}(\boldsymbol{x})p_\mathbf{X}(\boldsymbol{x})d\boldsymbol{x}}{\int_{S_\mathbf{X}} \mathbf{\Psi_i}^2(\boldsymbol{x})p_\mathbf{X}(\boldsymbol{x})d\boldsymbol{x}} \quad (4)$$

We calculate $c_{\mathbf{i}}$ using a variant of Equation 3 in which we sum over all dimensions of $\mathbf{i}$.

Global sensitivity (Sobol) indices. Sobol indices [36] decompose the variance of the model output over the entire input distribution into portions that depend on subsets of the input variables X_i. The *first order* sensitivity indices show the contribution of a single input variable to the output variance. For a variable X_i, the sensitivity index is $S_i = V_i/V$. Here, $V = Var_\mathbf{X}(f(\boldsymbol{x}))$ is the total variance and

$$V_i = Var_{X_i}(E_{\mathbf{X}_{\neg i}}(f(\boldsymbol{x})|X_i = x_i)) \quad (5)$$
$$\text{where} \quad \mathbf{X}_{\neg i} = \{X_1, \ldots, X_d\} \setminus \{X_i\}$$

We can evaluate Equation 5 analytically when f is a polynomial (such as those generated via GPC) and when it is possible to calculate the moments of each independent component of $\mathbf{X}$ analytically. For more complex functions and distributions, we must use Monte Carlo estimators [36, Equation 6].

IV. GAS Approach

We present the GAS approach, which has three main steps:

1) Create a deterministic complete vehicle model.
2) Train a perception model to replace the regression NN used for state perception (Algorithms 1-2).
3) Construct a complete vehicle surrogate (Algorithm 3).

GAS automates almost the entire process of constructing and using the surrogate model. The user provides the environment distribution ($\mathcal{D}_E$), distribution of other relevant random variables ($\mathcal{D}_R$), initial state distribution ($\mathcal{D}_S^0$), and other simulation parameters to GAS.

A. Creating a deterministic vehicle model

First, we represent the complete vehicle model as a function of independent random variables, $M_V : \mathbb{D}_S \times \mathbb{D}_E \times \mathbb{D}_R \to \mathbb{D}_S$. $S \in \mathbb{D}_S$ is a vector of state variables, $E \in \mathbb{D}_E$ is a vector of environment-related random variables that affect the image processed by the NN (e.g., weather and lighting conditions), and $R \in \mathbb{D}_R$ is a vector of random variables that do not affect the NN, but affect other parts of M_V. Making M_V deterministic allows us to explicitly sample the output distribution of M_V for any given state. We remove any input

variable dependencies by isolating independent components of input variables as necessary.

Algorithm 1 Training the perception model (M_{per})

Input G: set of ground truth states; $\mathcal{D}_E$: distribution of environment variables; n_i: num. images to capture for each $g \in G$
Returns M_{per}: perception model; d_{per}: perception model degree

function TRAINPERCEPTIONMODEL($G, \mathcal{D}_E, n_i$)
  $TrainTestData \leftarrow \{\ \}$
  for $g \in G$ **do**
    $I_g \leftarrow [\ ]$
    for i from 1 to n_i **do**
      $E \sim \mathcal{D}_E$
      $Img \leftarrow$ CAPTUREIMAGE(g, E)
      $I_g \leftarrow I_g :: Img$
    $O_g \leftarrow$ NEURALNETWORK(I_g)
    $\mu_g \leftarrow$ MEAN(O_g)
    $\sigma_g^2 \leftarrow$ COVARIANCE(O_g)
    $TrainTestData \leftarrow TrainTestData[g \mapsto (\mu_g, \sigma_g^2)]$
  $M_{per}, d_{per} \leftarrow$ POLYREGMODEL($TrainTestData$)

B. Replacing the perception system

The output of regression NNs which use camera images to perceive the vehicle's state is affected by environmental factors. Given a ground truth state and environment distribution, there is a corresponding *output distribution* of the NN. To enable faster and deterministic sampling of this output distribution, GAS replaces the perception system with a perception model.

Algorithm 1 shows how GAS creates the perception model M_{per}. GAS chooses a set of ground truth states G from the set of safe states $\mathbb{D}_S^{safe}$ (i.e., states that do not violate safety properties). For each ground truth state $g \in G$, GAS 1) captures a list of images I_g in environments E sampled from the environment distribution $\mathcal{D}_E$, 2) passes I_g through the perception NN to obtain a list of outputs O_g, and 3) calculates the mean μ_g and covariance σ_g^2 of O_g. GAS trains a polynomial regression model M_{per} to predict the mean μ_S and covariance σ_S^2 of the output distribution O_S at any ground truth state $S \in \mathbb{D}_S^{safe}$. GAS also infers the optimal polynomial degree d_{per} to maximize accuracy while preventing overfitting.

Algorithm 2 Abstracted vehicle model (M'_V)

Input S: initial state of vehicle; N: raw sample to be transformed into neural network output sample; R: other random variables; M_{per}: trained perception model
Returns S': state of vehicle after one time step

function $M'_V(S, N, R, M_{per})$
  $\mu_S, \sigma_S^2 \leftarrow M_{per}(S)$
  $O_S \leftarrow$ TRANSFORM(N, μ_S, σ_S^2)
  $S' \leftarrow$ VEHICLECONTROLANDDYNAMICS(S, O_S, R)

We create an *abstracted* vehicle model $M'_V : \mathbb{D}_S \times \mathbb{R}^n \times \mathbb{D}_R \to \mathbb{D}_S$ (Algorithm 2) which uses M_{per} instead of the NN. For the input vehicle state S, M'_V first calculates the perception NN output distribution O_S. Specifically, GAS assumes that $O_S = \mathcal{N}(\mu_S, \sigma_S)$, where μ_S and σ_S^2 are the output distribution parameters at S predicted using M_{per}. Instead of a sample from $\mathcal{D}_E$, M'_V accepts a sample N from a multivariate standard normal distribution $\mathcal{N}(0, 1)$. M'_V transforms N to a sample from O_S. M'_V uses this sample as the perceived state for the rest of the model consisting of the vehicle's control and dynamics systems.

GAS's assumption that the perception NN output is distributed according to $\mathcal{N}(\mu_S, \sigma_S)$ is based on the the output distribution for real-world images, which has a normal distribution. However, GAS can use the same method for other distributions if the parameters of the fitted distribution vary smoothly as the ground truth changes.

Algorithm 3 GPC surrogate model (M_{GPC}) construction

Input $\mathcal{D}_S^{safe}$: distribution over $\mathbb{D}_S^{safe}$; $\mathcal{D}_R$: distribution of other random variables; o_{gpc}: order of GPC model; M'_V: abstracted vehicle model
Returns M_{GPC}: GAS surrogate model

function CREATEGPCMODEL($\mathcal{D}_S^{safe}, \mathcal{D}_R, o_{gpc}, M'_V$)
  $J \leftarrow$ JOIN($\mathcal{D}_S^{safe}, \mathcal{N}(0,1), \mathcal{D}_R$)
  $\Psi \leftarrow$ GENERATEORTHOGONALPOLYNOMIALS(o_{gpc}, J)
  $X, W \leftarrow$ GENERATEQUADNODESWEIGHTS(o_{gpc}, J)
  $Y \leftarrow [M'_V(x)$ for $x \in X]$
  $M_{GPC} \leftarrow$ QUADRATUREANDGPC(Ψ, X, W, Y)

C. GPC for the complete vehicle system

Algorithm 3 shows how GAS constructs the GPC approximation of the abstracted vehicle model M'_V. GAS first constructs a joint distribution J over all input variables to M'_V by taking the product of a distribution for each input variable. GPC will produce a polynomial approximation that minimizes ℓ_2 error weighted by the probability distribution of J. For the state variables, GAS chooses a normal or truncated normal distribution $\mathcal{D}_S^{safe}$ over the safe state space $\mathbb{D}_S^{safe}$. For the raw sample that is transformed into a sample from the perception system output distribution O_S, GAS uses $\mathcal{N}(0, 1)$, as that ensures that the transformed sample is indeed distributed according to O_S. For the other random variables, GAS uses their actual distribution $\mathcal{D}_R$. Then, $J = \mathcal{D}_S^{safe} \times \mathcal{N}(0,1) \times \mathcal{D}_R$. Next, GAS calculates the basis polynomials Ψ which are orthogonal w.r.t. J. To efficiently calculate the coefficients of the polynomials in Ψ, GAS uses Gaussian quadrature for numerical integration. The surrogate model M_{GPC} is the sum of the orthogonal basis polynomials multiplied by these calculated coefficients, as per Equation 4.

Categorical state variables. Some vehicle models have categorical state variables. For example, many control systems operate in multiple modes. The control system can switch modes if certain conditions are met, and the current mode affects the control decisions. In this case, the current mode is a categorical state variable. Unlike categorical variables, polynomial inputs and outputs are continuous intervals. Therefore, we cannot use GPC for predicting categorical variables, or accept a categorical variable as an input to the GPC model. One option is to create a different type of surrogate model for such vehicle models. However, GAS can still enable the use of GPC by using multiple GPC sub-models and using a separate classifier for predicting categorical variables. This procedure is known as *multi-element GPC* (ME-GPC).

Suppose a vehicle model's state includes a categorical variable X with the domain $\mathbb{D}_X = \{x_1, \ldots, x_k\}$. GAS uses

GPC to create a separate surrogate model for each $x_i \in \mathbb{D}_X$. The compound surrogate model chooses which of these sub-models to use based on the value of X. In this way, GAS calculates all output state variables except X. For predicting X, GAS creates an ancillary classifier. GAS trains the ancillary classifier in the same manner as the perception model in Algorithm 1, with the main distinction being that it creates a classification model as opposed to a regression model.

Algorithm 4 Estimating the safe state probability over time

Input n_s: number of samples to use for distribution estimation; T: number of time steps; $\mathcal{D}_S^0$: initial state distribution; $\mathcal{D}_R$: distribution of other random variables; $Pred$: safety predicate; M_{GPC}: constructed GAS surrogate model
Returns P_{safe}: probability that the vehicle remains in a safe state until each time step

function ESTIMATESAFEPROB($n_s, T, \mathcal{D}_S^0, \mathcal{D}_R, Pred, M_{GPC}$)
  $X \leftarrow [\,]$; $P_{safe} \leftarrow \{\,\}$
  for i from 1 to n_s **do**
    $x \sim$ JOIN($\mathcal{D}_S^0, \mathcal{N}(0,1), \mathcal{D}_R$)
    $X \leftarrow X :: x$
  for t from 1 to T **do**
    $X' \leftarrow [\,]$
    for $x \in X$ **do**
      $S' \leftarrow M_{GPC}(x)$
      if SAFE($S', Pred$) **then**
        $N' \sim \mathcal{N}(0,1)$
        $R' \sim \mathcal{D}_R$
        $X' \leftarrow X' :: (S', N', R')$
    $X \leftarrow X'$
    $P_{safe} \leftarrow P_{safe}[t \mapsto |X|/n_s]$

D. Applications of the GAS surrogate model

Calculating the probability of remaining in a safe state over time. GAS uses the surrogate model to estimate the probability that the vehicle will remain in a safe state over time (Algorithm 4). GAS creates initial state samples S from the initial state distribution $\mathcal{D}_S^0$. At each time step, GAS chooses random samples N and R from $\mathcal{N}(0,1)$ and $\mathcal{D}_R$ respectively. GAS evaluates the surrogate model on each joint sample to get the next state. Finally, GAS calculates the fraction of samples that always remained in the safe region.

Algorithm 5 Using estimators to calculate sensitivity indices

Input n_s: number of samples to use for sensitivity estimation; i: index of state variable to calculate sensitivity for; $\mathcal{D}_S$: current state distribution; $\mathcal{D}_R$: distribution of other random variables; M: model (M_{GPC} or M_V')
Returns S_i: sensitivity index of selected state variable

function ESTIMATESENSITIVITY($n_s, i, \mathcal{D}_S, \mathcal{D}_R, M$)
  $Y_0 \leftarrow [\,]$; $Y_1 \leftarrow [\,]$
  for i from 1 to n_s **do**
    $x_0, x_1 \sim$ JOIN($\mathcal{D}_S, \mathcal{N}(0,1), \mathcal{D}_R$)
    $y_0 \leftarrow M(x_0)$; $y_1 \leftarrow M(x_1[i \mapsto x_0[i]])$
    $Y_0 \leftarrow Y_0 :: y_0$; $Y_1 \leftarrow Y_1 :: y_1$
  $S_i \leftarrow (\text{MEAN}(Y_0 * Y_1) - \text{MEAN}(Y_0)^2)/\text{VAR}(Y_0)$

Computing Sobol indices. GAS uses M_{GPC} to calculate Sobol sensitivity indices in two ways. In the *analytical* approach, GAS calculates sensitivity indices by first calculating conditional expected values as polynomials and then calculating their variance (Equation 5). In the *empirical* approach, GAS instead uses Monte Carlo estimators ([36, Equation 6] as implemented in Algorithm 5).

While the analytical approach precisely calculates sensitivity indices, it is relatively slow as GAS must calculate expected values as a function of the variable whose sensitivity is being calculated. The empirical approach becomes more accurate as the number of samples increases. Despite this, it can be faster than the first approach due to the speed of evaluating M_{GPC}.

Rapid iteration. During development of an autonomous vehicle, developers may rapidly make or propose changes to the vehicle model. GAS enables faster testing of these prototypes with its compositional approach. If developers modify the control or dynamics, then GAS reuses the perception model, saving a significant amount of time since Algorithm 1 is the major contributor to GAS's runtime. If developers modify the perception system, GAS must create a new perception model. Even then, GAS still saves time if creating a new perception model is cheaper than using the original vehicle model.

E. Properties of the GAS approach

Accuracy. Multiple GAS parameters affect the accuracy of the GPC model: the size of the tensor grid $|G|$, the number of images taken for each grid point n_i, the degree of polynomial regression used for the perception model, and the GPC order o_{gpc}. Under certain conditions, GAS converges in distribution to the exact solutions.

Lemma 1 (Perception Model Convergence)**.** *Assume that 1) for the given environment distribution $\mathcal{D}_E$, the distribution of the outputs of a perception NN $\mathcal{N}$ in any ground truth state S is Gaussian over the perceived state, and 2) each component of the distribution parameters (μ_S, σ_S^2) is an analytic function of S. Then, the output distribution of the perception model M_{per} in any state approaches $\mathcal{N}$'s output distribution at that state as $|G|$, n_i, and d_{per} increase.*

Proof Sketch. Increasing $|G|$ increases the number of states S used to train the perception model. Increasing n_i increases the accuracy of $\mathcal{N}$'s output distribution parameters calculated at each S. These distribution parameters are analytic functions of S, so they can be calculated using a Taylor series. After increasing the number and accuracy of training data points, the accuracy of the perception model can be arbitrarily increased by increasing d_{per} (Increasing d_{per} without also increasing $|G|$ leads to overfitting.). □

We use standard statistical tests such as the Shapiro-Wilk test to check if $\mathcal{N}$'s outputs have a Gaussian distribution for the environment distribution $\mathcal{D}_E$ used in Section IV-B. We have observed this to be true in practice [1, Section A][44]. We can also use a different base distribution (and corresponding orthogonal polynomials for GPC) if fits the data better across the state space. Practically, controlling the error of the perception model (or any approximation of a NN) is an open problem [31]. Precise analytic calculation of the perception model error is intractable, but we can empirically estimate the error.

TABLE I: GAS benchmarks

Benchmark	Perception	Control	Preprocessing
Crop-Monitor	ResNet-18×2	Skid-Steer	Perc→Poly Reg
Cart-Straight	LaneNet	Pure Pursuit	Perc→Poly Reg
Cart-Curved	LaneNet	Pure Pursuit	Perc→Poly Reg
ACAS-Table	Ground Truth	ACAS-Xu Table	Ctrl→Dec Tree
ACAS-NN	Ground Truth	ACAS-Xu NN	Ctrl→Dec Tree

Lemma 2 (GPC Error Bound)**.** *Assume the control system and vehicle dynamics in M'_V are differentiable. Then, the root mean square (RMS) error of the output of the GAS model M_{GPC} w.r.t. the output of M'_V is bounded.*

Proof Sketch. From [40, Theorem 3.6] and Ernst et al. [10], which state that the RMS error of a GPC approximation is proportional to o_{gpc}^{-p}, where p is a positive value that depends on the differentiability of the function being approximated. The process of generating a NN output sample through the perception model is a polynomial evaluation followed by an affine transform – both are differentiable operations. The control system and dynamics are differentiable by assumption. Finally, composing differentiable functions yields a differentiable function. □

M_{GPC} is the optimal polynomial model of M'_V for any o_{gpc}, in terms of ℓ_2 error [40][Equation 5.9]. In practice, control systems may not be differentiable everywhere (e.g., due to mode switching), but the differentiability of vehicle dynamics, coupled with a short time step, limit negative effects on accuracy.

Corollary 1 (GPC Asymptotic Convergence)**.** *As $o_{gpc} \to \infty$, RMS error of GPC approaches 0, that is, M_{GPC} can be an arbitrarily close approximation of M'_V.*

Proof Sketch. From Lemma 2, the RMS error is proportional to o_{gpc}^{-p}, where p is positive. Then, $\lim_{o_{gpc}\to\infty} o_{gpc}^{-p} = 0$. □

Theorem 1 (GAS Convergence)**.** *Assume that the distribution of the outputs of a perception NN $\mathcal{N}$ in any ground truth state is Gaussian. Then, the GAS model M_{GPC} converges in output distribution to the original vehicle model M_V.*

Proof Sketch. We can use arbitrarily accurate perception models (Lemma 1) to get accurate NN output samples for any state in $\mathbb{D}_S^{safe}$. The GPC model can be made an arbitrarily accurate approximation of M'_V (Corollary 1), and thus of M_V. □

Runtime. The dominant factor for runtime is the required number of evaluations of M_V. To gather data for the perception model, GAS requires $\Theta(|G|n_i)$ evaluations of M_V. The amount of time required to train M_{per} and construct M_{GPC} is insignificant in comparison. For state distribution estimation over time, we must evaluate M_V or the much faster M_{GPC} $\Theta(n_s T)$ times. For estimating sensitivity indices using estimators, we must evaluate either M'_V or M_{GPC} $\Theta(n_s)$ times and then calculate mean and variance of the samples.

V. Methodology

Benchmarks. We chose five benchmarks that include autonomous vehicle systems such as self driving carts, unmanned aircraft, and crop monitoring vehicles. Table I shows details of the benchmarks. Columns 2 and 3 state the vehicle's perception and control system, respectively. Column 4 indicates if GAS made a replacement in the perception (Perc) or control (Ctrl) system, and the nature of the replacement (Poly Reg: polynomial regression, Dec Tree: decision tree). Each benchmark has a total of 4 state or random variables. The benchmarks are:

- **Crop Monitoring Vehicle.** Our main example, described in detail in Section II.
- **Self-Driving Cart (Straight Path).** A vehicle that must drive within a marked path ($\mathbb{D}_S^{safe} \equiv |heading| \leq \pi/12 \wedge |distance| \leq 1.2m$). It uses LaneNet to perceive the lane boundaries and the pure pursuit controller. We derive this benchmark from [8] and use [28].
- **Self-Driving Cart (Curved Path).** Similar to the previous benchmark, but the vehicle must drive on a circular path of radius 100m.
- **Unmanned Aircraft Collision Avoidance (Lookup Table).** An unmanned aircraft that must avoid a near miss with an intruder ($\mathbb{D}_S^{safe} \equiv |separation| \geq 0.1524km$). The aircraft uses ACAS-Xu lookup tables from [21]. As this model's state includes a categorical variable (the previous ACAS advisory), we use ME-GPC and predict the next advisory using a decision tree as the ancillary model.
- **Unmanned Aircraft Collision Avoidance (Neural Network).** Similar to the previous benchmark, but uses an NN from [21] trained to replace the lookup table.

Implementation and experimental setup. We performed our experiments on machines with a Quadro P5000 GPU, using one Xeon CPU core. We implement GAS in Python, using `chaospy` [13]. We use Gazebo 11 [23] to capture images for the Crop-Monitor and Cart benchmarks. We run all vision DNNs on GPU, and the smaller ACAS-NN on CPU.

For our main evaluation, we create only GPC surrogate models with GAS. For estimating state distribution over time, we compare the GAS-generated M_{GPC} to a MCS baseline using M_V. We set GAS parameters as follows: G is a 11×11 grid in $\mathbb{D}_S^{safe}$, $n_i = 350$, and $o_{gpc} = 4$. We also experiment with alternate values for G and n_i, as they directly affect perception model training data generation time. We set the number of time steps $T = 100$. To keep MCS runtime within 24 hours, we set $n_s = 1,000$ for MCS. For GAS, we increase n_s to 10,000 as M_{GPC} is much faster than M_V and increasing the number of samples decreases sampling error for M_{GPC}.

We calculate sensitivity using both the analytical and empirical method described in Section IV. It is not possible to compare sensitivity indices against M_V, as M_V has a different set of inputs (environment specification instead of a sample from $\mathcal{N}(0,1)$). Therefore, we use sensitivity index calculation using M'_V as the baseline. For the empirical method, we set $n_s = 10^6$, but also monitor the results obtained by setting n_s to 10^4, 10^5, and 10^7. We calculate the sensitivity of state variables to those in the previous time step, as well as the sensitivity of the change in the state variables.

TABLE II: Metrics for comparing state variable distributions

Benchmark	Variable	μ_{GAS}/μ_{MCS}	$\sigma_{GAS}/\sigma_{MCS}$	$\mathbf{KS}_{max}$	$\mathbf{Wass}_{max}$
Crop-Monitor	Heading (rad)	-0.004/-0.008	0.04/0.04	0.11	0.02
	Distance (m)	-0.01/-0.02	0.03/0.03	0.14	0.01
Cart-Straight	Heading (rad)	0.0004/-0.0001	0.02/0.02	0.13	0.009
	Distance (m)	0.16/0.08	0.09/0.12	0.41	0.08
Cart-Curved	Heading (rad)	-0.003/-0.005	0.006/0.007	0.17	0.003
	Distance (m)	0.18/0.19	0.04/0.05	0.15	0.02
ACAS-Table	Crossrange (km)	-0.07/-0.03	0.85/0.88	0.05	0.04
	Downrange (km)	-0.61/-0.53	0.23/0.22	0.13	0.08
	Heading (rad)	0.63/-0.54	2.82/2.80	0.23	1.23
ACAS-NN	Crossrange (km)	-0.01/-0.01	0.93/0.91	0.02	0.03
	Downrange (km)	-0.45/-0.58	0.17/0.11	0.31	0.13
	Heading (rad)	0.42/0.15	2.70/2.75	0.06	0.27

TABLE III: Metrics for comparing the probability of remaining in a safe state

Benchmark	t-test	ℓ_2 err	X-Cor
Crop-Monitor	99/100	0.004	0.974
Cart-Straight	97/100	0.001	0.862
Cart-Curved	98/100	0.001	0.865
ACAS-Table	100/100	0.007	0.998
ACAS-NN	100/100	0.003	0.999

TABLE IV: Max diff. in sensitivity indices

Benchmark	$\mathbf{x_0 \rightarrow y_1}$	$\mathbf{x_0 \rightarrow dy_0}$
Crop-Monitor	0.00003	0.0004
Cart-Straight	0.0002	0.006
Cart-Curved	0.00003	0.003
ACAS-Table	0.00001	0.009
ACAS-NN	0.00002	0.061

Environmental factors. For the Crop-Monitor benchmark, our test scenario includes two crop types, four crop growth stages, and multiple models per stage (total 30). For the Cart benchmarks, our scenario includes the presence of 0-2 other carts, 0-2 pedestrians, and skid marks that obstruct lane markings. We also vary lighting conditions (midday to dusk). All images representing these conditions are captured from the simulators.

Distribution similarity metrics. We compare each dimension of the MCS and GAS state distributions at each time step using two complementary metrics. The conservative *KS statistic* quantifies the maximum distance between the cumulative distribution functions of the two distributions at any point. The *Wasserstein metric* quantifies the minimum probability mass that must be moved to transform one distribution into the other. For both metrics, a lower value indicates greater distribution similarity. We can use these distribution similarity metrics despite using more samples for GAS than for MCS.

Safe state probability similarity metrics. We also compare the fraction of simulated vehicles remaining in the safe region till each time step using three metrics. The two sample t-test is a statistical test to check if the underlying distributions used to draw two sets of samples are the same. The ℓ_2 error is the RMS of the differences in safe state probability at each time step. Lastly, we calculate the Pearson cross-correlation coefficient between the two sets of safe state probabilities. When plotting safe state probability, we also draw the 95% bootstrap confidence interval. This confidence interval does not directly compare the two plots, but rather, for each individual plot, it provides an estimate of the variation that can occur in that plot as a result of sampling error.

VI. Evaluation

A. Estimating the probability of violating a safety property

Table II compares the distributions calculated by GAS and MCS for each benchmark state variable. Columns 3-4 compare the mean and standard deviation of the distributions at the *final* time step. Columns 5-6 show the maximum values of the KS statistic and Wasserstein metric over *all* time steps. For most state variables, the mean and standard deviation of the distributions match closely up to the final time step. This is also indicated by the low values of the Wasserstein metric and the conservative KS statistic. The largest difference is for the Cart-Straight benchmark distance distribution. This occurs because the GAS model and the original vehicle model converge towards slightly different states around the center of the safe state space in later time steps. However, during the initial time steps where more simulated vehicles are in danger of entering unsafe states, the KS statistic does not exceed 0.15. A similar phenomenon affects the ACAS-NN downrange distance variable. For ACAS-Table, the GAS and original vehicle models occasionally turn in different directions to avoid an intruder approaching head-on, in situations where turning in either direction is equally beneficial. This leads to a large deviation in the heading variable.

Figure 6 shows the evolution of the probability that the vehicle remains in a safe state. The blue solid and red dashed plots show the probability estimates by GAS and MCS, respectively. The shaded region around each plot shows the 95% bootstrap confidence interval. Because we use $10\times$ more samples when estimating safe state probability with GAS as compared to MCS, the sampling error is smaller for GAS, which leads to a smaller confidence interval. Table III shows the metrics we use to measure the similarity of the safe state probabilities from Figure 6. Column 2 shows the number of time steps for which the t-test passed, meaning that we could not reject the null hypothesis that the probabilities are equal. Column 3 shows the ℓ_2 error, and Column 4 shows the cross-correlation. The similarity of the state distributions directly leads to the similarity of the safe state probability for most time steps. We extended the Crop-Monitor and Cart experiments to 500 time steps to confirm that the safe state probability does not deviate after 100 time steps. We did not similarly extend the ACAS experiments as the ACAS system is primarily relevant as the aircraft approach each other.

B. Estimating sensitivity indices

Table IV presents the maximum difference between sensitivity indices calculated using M_{GPC} and those calculated using M'_V. We found that about 10^6 samples are needed for

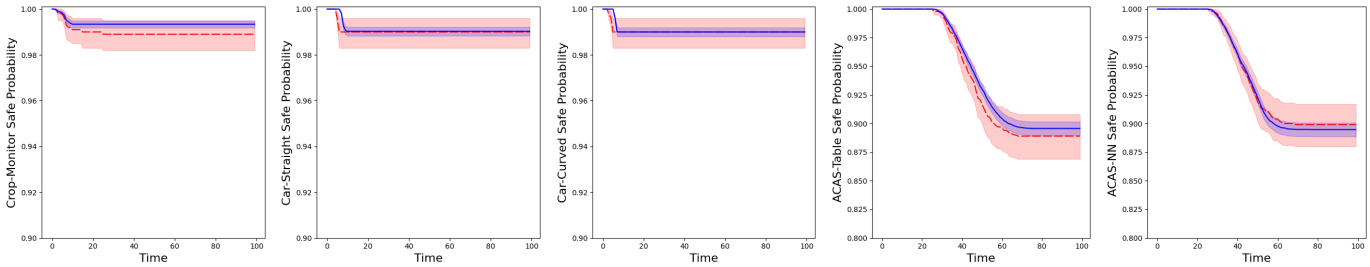

Fig. 6: Evolution of safe state probability over time. Blue solid: GAS, red dashed: MCS.

convergence of the sensitivity indices when using empirical estimators [1, Section B]. Column 2 shows the sensitivity of state variables in time step 1 to those in time step 0 ($x_0 \to y_1$) and Column 3 shows the sensitivity of the change in the state variables ($x_0 \to dy_0$, where $dy_0 = y_1 - y_0$). The sensitivity indices calculated using M_{GPC} and M'_V match closely – the average difference in sensitivity indices estimated using M_{GPC} and M'_V is 0.00003 in Column 2 and 0.005 in Column 3, indicating that GAS accurately estimates sensitivity.

C. Impact of perception model parameters on accuracy

Table V describes the effects of changing the perception model parameters G and n_i. Column 1 presents the parameter value. Columns 2-4 present the maximum KS statistic and Wasserstein metric ($\times 100$) for the Crop-Monitor, Cart-Straight, and Cart-Curved benchmarks, respectively. Column 5 presents the speedup caused by changing the parameter value, estimated based on the change in the number of images that must be processed. We exclude the ACAS benchmarks as they do not use a perception model.

TABLE V: Effect of changing perception model parameters on accuracy. An asterisk (*) indicates the primary value used in our evaluation. Changes of 10% or more are highlighted.

Parameter	$\mathbf{KS}_{\max} \times 100 / \mathbf{Wass}_{\max} \times 100$			**Relative**
Value	**C-Mon**	**C-Str**	**C-Cur**	**Speedup**
Ground truth grid dimensions (G)				
7×7	**20.3**$^{\uparrow}$ / 2.43	**46.8**$^{\uparrow}$ / **9.63**$^{\uparrow}$	**23.8**$^{\uparrow}$ / **3.24**$^{\uparrow}$	2.5×
9×9	**12.2**$^{\downarrow}$ / 2.25	42.4 / 8.39	16.2 / **2.00**$^{\downarrow}$	1.5×
11×11*	13.8 / 2.37	41.1 / 7.97	17.0 / 2.23	1.0×
Images captured per grid point (n_i)				
100	**33.1**$^{\uparrow}$ / 2.34	**49.6**$^{\uparrow}$ / **9.77**$^{\uparrow}$	**18.7**$^{\uparrow}$ / **2.59**$^{\uparrow}$	3.5×
225	**16.1**$^{\uparrow}$ / 2.35	40.5 / 7.91	17.0 / 2.43	1.6×
350*	13.8 / 2.37	41.1 / 7.97	17.0 / 2.23	1.0×

We focus on the cases where the error metrics change by 10% or more. The grid size G can be reduced to 9×9 without much loss of accuracy, but further reducing it to 7×7 increases the error for all benchmarks. Reducing the number of images captured at each grid point (n_i) to 225 does not cause much loss of accuracy, but further reducing it to 100 images increases the error for all benchmarks. The minimal change in accuracy caused by increasing G from 9×9 to 11×11 or increasing n_i from 225 to 350 also shows that further increases are unlikely to improve the accuracy of GAS.

D. Speed of GAS model compared to the original model

GAS model construction. Table VI shows the time required to construct the GAS model. Column 2 shows the time required to gather training data for the perception model, Column 3 shows the time required to create the perception and/or ancillary model, and Column 4 shows the time required to create M_{GPC}. Creating the perception, ancillary, and GPC models takes a few seconds, but gathering training data for the perception model takes several hours. Section VI-C shows how reducing the perception model parameters $|G|$ and n_i can reduce this time, but at the cost of accuracy.

State distribution estimation. Table VII shows the time required by GAS and MCS for state distribution estimation. Column 2 shows the time required for using M_V. Column 3 shows the total time required by M_{GPC} – this includes the *total* time required to construct M_{GPC} (from Table VI) and then use it for state distribution estimation via Algorithm 4. Column 3 also shows the speedup of GAS over MCS. The costly process of gathering and processing images contributes to over 99% of t_{MCS} for the Crop-Monitor and Cart benchmarks. While increasing n_s or T increases t_{MCS} significantly, the corresponding increase in t_{GAS} is negligible as the time required to create the perception model is independent of n_s and T. Consequently, the speedup of GAS for state distribution estimation increases for longer experiments or higher number of samples. For the ACAS benchmarks, the control component of M_V contributes to over 90% of t_{MCS}. M_{GPC} is faster than even the dynamics component of M_V, leading to significant speedups for these benchmarks.

Sensitivity analysis. Table VIII shows the time required by GAS and MCS for sensitivity analysis. Columns 2-3 show the required for calculating *all* sensitivity indices empirically with M'_V and M_{GPC} respectively. Column 4 shows the time required calculating all sensitivity indices analytically with M_{GPC}. Columns 3-4 also show the speedup of GAS for either approach. As both M'_V and M_{GPC} use the perception model for the first three benchmarks, we exclude the time required to train and construct the perception model for those benchmarks. The replacement of the perception system by the perception model significantly speeds up M'_V as compared to M_V and also enables vectorization. As this optimized version is the baseline for sensitivity indices calculation, the speedup of GAS for this application is lower than that for distribution estimation. The analytical and empirical methods for calculating sensitivity using M_{GPC} have similar accuracy,

TABLE VI: Time usage for creating GAS surrogate model M_{GPC}

Benchmark	t_{data}	$t_{per/anc}$	t_{GPC}
Crop-Monitor	8.5h	1.1s	1.4s
Cart-Straight	3.3h	1.1s	1.4s
Cart-Curved	3.1h	1.1s	1.4s
ACAS-Table	N/A	0.3s	0.3s
ACAS-NN	N/A	0.3s	0.4s

TABLE VII: Time usage for state distribution estimation

Benchmark	t_{MCS}	t_{GAS}
Crop-Monitor	19.5h	8.5h (2.3×)
Cart-Straight	6.8h	3.3h (2.1×)
Cart-Curved	6.5h	3.1h (2.1×)
ACAS-Table	8.0s	1.1s (7.3×)
ACAS-NN	10.7s	1.2s (8.9×)

TABLE VIII: Time usage for sensitivity analysis (excluding t_{dat} and t_{per} from Table VI)

Benchmark	t^{emp}_{MCS}	t^{emp}_{GAS}	t^{ana}_{GAS}
Crop-Monitor	9.5s	6.2s (1.3×)	11.4s (0.7×)
Cart-Straight	9.7s	6.2s (1.3×)	11.4s (0.8×)
Cart-Curved	9.8s	6.2s (1.3×)	11.3s (0.8×)
ACAS-Table	5.2s	5.6s (0.9×)	3.3s (1.6×)
ACAS-NN	4.8s	5.3s (0.9×)	3.1s (1.5×)

but are faster for different benchmarks.

Rapid iteration. For the first three benchmarks, if the perception system is altered, GAS must create a new perception model; the speedup of GAS stays the same, but the total amount of time saved over MCS increases with each iterative change. However, if only the vehicle control or dynamics are altered, then GAS does not need to create a new perception model. Because gathering data for the perception model is the major contributor to the runtime of GAS, this allows vehicle developers to rapidly make changes to the control and dynamics systems of the vehicle and reanalyze the system with these changes within seconds.

VII. Threats to Validity

GAS samples the provided environment distribution $\mathcal{D}_E$ when creating the perception model. If the scenario under test has a significantly different environment distribution, then this perception model may lead to a loss of accuracy. Developers must therefore choose $\mathcal{D}_E$ so that it represents the expected environment distribution that the vehicle will operate in. If developers change the environment distribution, we can still reuse existing training data that fits in the new distribution.

To use GPC, all input variable distributions must have corresponding orthogonal polynomials. This is true for many distributions (uniform, normal, beta, gamma, etc.), which together are usually sufficient for modeling relevant inputs.

As the GPC model is a polynomial, it has limited accuracy when modeling functions with limited differentiability [40, Theorem 3.6]. Consequently, the GPC model can have a systemic bias that can be reduced, but not eliminated. Despite our benchmarks having such non-differentiable points, GAS still provides excellent accuracy. We demonstrate the applicability of GAS to autonomous vehicle scenarios with various structures and associated challenges. The applicability of GAS for arbitrary autonomous vehicle scenarios is an open question and an interesting topic for future work.

Advances in simulation may reduce the speedup offered by GAS over time. We expect this reduction in speedup to be minimal, as Algorithm 1, which is the main contributor to GAS runtime, would also benefit from these advances.

Scalability. The GPC polynomial model suffers from the "curse of dimensionality" (combinatorial explosion of the number of polynomial terms) as it must consider all possible interactions between state variables. To overcome this general limitation of GPC, researchers have proposed solutions such as low-rank approximation [24]. Another solution is to omit higher-order polynomial terms in which multiple state variables interact [12]. Figure 7 shows the effects of this

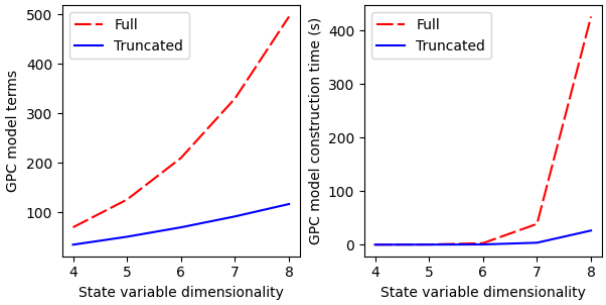

Fig. 7: Effect of truncation on GPC model terms and construction time for an order 4 GPC model.

polynomial truncation approach for an order 4 GPC model. For both plots, the X-Axis shows the number of dimensions in the state space. The Y-Axis of the left plot shows the number of polynomial terms in the constructed GPC model, and the Y-Axis of the right plot shows the model construction time. The dashed red lines and the solid blue lines show the results for the full polynomial and the truncated polynomial, respectively. The speedup from truncation increases rapidly with dimensionality. This method still retains lower-order interaction terms, ensuring that the constructed GPC model still accounts for interactions between state variables to some degree. We can adjust the aggressiveness of truncation to trade off between analysis time and accuracy. If creating a GPC model is still too expensive, we can forego the GPC model and directly use the abstracted vehicle model as we did in Section VI-B.

Alternate surrogate models. We experimented with different variations of the GPC-based approach shown in Section IV. We also experimented with alternative types of surrogate models, such as standard polynomial regression surrogates (average t-tests passed: 98.2) or NN surrogates with a similar number of parameters (average t-tests passed: 47.8). We found that the approach presented in this paper produced the best results (average t-tests passed: 98.8). We provide more details of the results for the alternative approaches in [1, Section C].

VIII. Related Work

Analysis and verification of vehicle systems. Many studies have shown that simulations can detect issues with vehicle software while saving resources over real-world testing or verification [15, 18, 34, 38, 39]. Recent surveys [2] show that developers in the vehicle industry use such simulation for regression testing of proposed vehicle system modifications, but that the high cost of simulation is a major roadblock to integrating and deploying such tests. Recent works, e.g. [27] have focused on test scenario selection and prioritization for

autonomous vehicles. GAS accelerates the testing of vehicles under the selected scenarios, and can save additional time when developers make incremental modifications.

DryVR [11] and Verse [25] are systems for verifying the safety of autonomous vehicle models containing whitebox mode-switching control logic composed with blackbox dynamics. They compute over-approximations of reachable states with a probabilistic guarantee on the learned sensitivity. In comparison, GAS focuses on perception and control systems which include blackbox components such as NNs. Pasareanu et al. [32] verify safety properties for vehicle systems with learning enabled components. They separately analyze the learning enabled components to derive guarantees of the component's behavior when certain assumptions about the input to that component hold. Such precise verification may not scale to large image processing NNs used to perceive the vehicle state from camera images. GAS is able to handle vehicle systems with such complex NNs by using sampling to provide precise estimates of the probability that the vehicle will violate safety properties.

Researchers have also studied the impact of environmental conditions, e.g., identifying conditions missing in the training data [42], certifying robustness to semantic environment perturbations [41], augmenting training data [7], and finding environmental conditions that can cause large perception errors [6]. Scenic [14] is a language for specifying scenes in a virtual world with varying environments for generating training images for perception NNs. Developers can use these works when sampling and prioritizing environments for the GAS perception model.

System simulation and modeling. Existing techniques can create surrogate models for vehicle *dynamics*, e.g., using GPC [22] or NNs [26]. Since dynamics simulation comprises 10% or less of the total simulation time, these approaches would provide negligible speedups. GAS creates surrogate models of *complete* vehicle systems, including the expensive perception and control components. ARIsTEO [29] uses abstraction refinement to create surrogate models of cyber-physical systems with low dimensional inputs. GAS handles high-dimensional image inputs by first creating a perception model, and then creating a surrogate model of the reduced-dimensionality abstract vehicle model. Morando et al. [30] use surrogate *safety measures* to calculate how close autonomous vehicles come to collision, which is orthogonal to creating surrogate models of vehicle systems. The ACAS safety property is one such surrogate safety measure. Several approaches [5, 17, 43] create surrogate models to estimate the *final result* of testing vehicles in particular scenarios, and use these surrogates to efficiently find potentially dangerous scenarios for additional in-depth testing. GAS can further speed up this additional testing of the selected scenarios.

Simplification of complex perception and control systems. Ghosh et al. [16] iteratively synthesize perception models and controllers guided by counterexamples to temporal logic safety properties. Hsieh et al. [20] create a perception model where the mean is calculated using piecewise linear regression and the allowable variance is calculated based on the controller code using program analysis tools like CBMC. Astorga et al. [4] synthesize perception contracts, which describe the uncertainty that the perception system can generate, and the control and dynamics system can tolerate, without violating safety properties. Researchers have used similar perception contracts for analysis of vision-based formation control [19] and automated landing [37]. Unlike these works, GAS's perception model is independent of the controller, and can be reused without requiring additional sampling when iterative changes are made to the controller during development.

IX. Conclusion

We presented GAS, the first approach for creating surrogate models of complete autonomous vehicle systems with complex perception and/or control components. GAS first creates a model of the perception system and uses it to generate a surrogate model of the entire vehicle system. GAS accurately models the system while being $3.7\times$ faster on average for safe state probability estimation and $1.4\times$ faster on average for sensitivity analysis than Monte Carlo Simulation on five scenarios used in agricultural vehicles, self driving carts, and unmanned aircraft. These speedups make GAS an attractive choice for efficiently testing iterative modifications to the vehicle system, such as regression testing during development.

Acknowledgements. This research was supported in part by NSF Grants No. CCF-1846354, CCF-1956374, and CCF-2008883. We thank Prof. Girish Chowdhary for providing us with the simulator for the TerraSentia crop monitoring vehicle, and Yifan Zhao for providing its photo (Figure 1).

References

[1] "Appendix to GAS," https://github.com/uiuc-arc/GAS.

[2] A. Afzal, D. S. Katz, C. L. Goues, and C. S. Timperley, "A study on the challenges of using robotics simulators for testing," in *ICST*, 2020.

[3] R. Alizadeh, J. K. Allen, and F. Mistree, "Managing computational complexity using surrogate models: a critical review," *Research in Engineering Design*, vol. 31, 2020.

[4] A. Astorga, C. Hsieh, P. Madhusudan, and S. Mitra, "Perception contracts for safety of ML-enabled systems," *OOPSLA*, 2023.

[5] H. Beglerovic, M. Stolz, and M. Horn, "Testing of autonomous vehicles using surrogate models and stochastic optimization," in *ITSC*, 2017.

[6] M. Biagiola and P. Tonella, "Testing of deep reinforcement learning agents with surrogate models," *Software Engineering and Methodology*, Nov. 2023.

[7] C.-H. Cheng, C.-H. Huang, and H. Yasuoka, "Quantitative projection coverage for testing ML-enabled autonomous systems," in *ATVA*, 2018.

[8] P. Du, Z. Huang, T. Liu, T. Ji, K. Xu, Q. Gao, H. Sibai, K. Driggs-Campbell, and S. Mitra, "Online monitoring for safe pedestrian-vehicle interactions," in *ITSC*, 2020.

[9] Earthsense, "A Growing Presence on the Farm: Robots," https://www.nytimes.com/2020/02/13/science/farm-agriculture-robots.html, 2020.

[10] O. Ernst, A. Mugler, H.-J. Starkloff, and E. Ullmann, "On the convergence of generalized polynomial chaos expansions," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 46, 2012.

[11] C. Fan, B. Qi, S. Mitra, and M. Viswanathan, "Dryvr: Data-driven verification and compositional reasoning for automotive systems," in *CAV*, 2017.

[12] J. Feinberg and H. P. Langtangen, "Truncation scheme - chaospy," https://chaospy.readthedocs.io/en/master/user_guide/polynomial/truncation_scheme.html.

[13] ——, "Chaospy: An open source tool for designing methods of uncertainty quantification," *Journal of Computational Science*, vol. 11, 2015.

[14] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: A language for scenario specification and scene generation," in *PLDI*, 2019.

[15] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *ISSTA*, 2019.

[16] S. Ghosh, Y. V. Pant, H. Ravanbakhsh, and S. A. Seshia, "Counterexample-guided synthesis of perception models and control," in *ACC*, 2021.

[17] F. U. Haq, D. Shin, and L. Briand, "Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization," in *ICSE*, 2022.

[18] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Comparing offline and online testing of deep neural networks: An autonomous car case study," in *ICST*, 2020.

[19] C. Hsieh, Y. Koh, Y. Li, and S. Mitra, "Assuring safety of vision-based swarm formation control," in *ACC*, 2024.

[20] C. Hsieh, Y. Li, D. Sun, K. Joshi, S. Misailovic, and S. Mitra, "Verifying controllers with vision-based perception using safe approximate abstractions," in *EMSOFT*, 2022.

[21] K. D. Julian and M. J. Kochenderfer, "Guaranteeing safety for neural network-based aircraft collision avoidance systems," in *DASC*, 2019.

[22] G. Kewlani, J. Crawford, and K. Iagnemma, "A polynomial chaos approach to the analysis of vehicle dynamics under uncertainty," *Vehicle System Dynamics*, 2012.

[23] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Workshop on Intelligent Robots and Systems*, 2004.

[24] K. Konakli and B. Sudret, "Uncertainty quantification in high dimensional spaces with low-rank tensor approximations," in *Uncertainty Quantification in Computational Sciences and Engineering*, 2015.

[25] Y. Li, H. Zhu, K. Braught, K. Shen, and S. Mitra, "Verse: A python library for reasoning about multi-agent hybrid system scenarios," in *CAV*, 2023.

[26] J. Lin, H. Li, Y. Huang, Z. Huang, and Z. Luo, "Adaptive artificial neural network surrogate model of nonlinear hydraulic adjustable damper for automotive semi-active suspension system," *IEEE Access*, vol. 8, 2020.

[27] C. Lu, H. Zhang, T. Yue, and S. Ali, "Search-based selection and prioritization of test scenarios for autonomous driving systems," in *Search-Based Soft. Engg.*, 2021.

[28] MaybeShewill-CV, "Unofficial implemention of the lanenet model for real time lane detection," https://github.com/MaybeShewill-CV/lanenet-lane-detection.

[29] C. Menghi, S. Nejati, L. Briand, and Y. I. Parache, "Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification," in *ICSE*, 2020.

[30] M. M. Morando, Q. Tian, L. T. Truong, and H. L. Vu, "Studying the safety impact of autonomous vehicles using simulation-based surrogate safety measures," *Journal of Advanced Transportation*, vol. 2018, 2018.

[31] R. Pan and H. Rajan, "On decomposing a deep neural network into modules," in *ESEC/FSE*, 2020.

[32] C. S. Păsăreanu, D. Gopinath, and H. Yu, "Compositional verification for autonomous systems with deep learning components," in *Safe, Autonomous and Intelligent Vehicles*, 2019.

[33] C. P. Robert and G. Casella, *Monte Carlo statistical methods*. Springer, 2004, vol. 2.

[34] C. Robert, T. Sotiropoulos, H. Waeselynck, J. Guiochet, and S. Vernhes, "The virtual lands of Oz: testing an agribot in simulation," *Empir. Softw. Eng.*, vol. 25, 2020.

[35] A. Sivakumar, S. Modi, M. Gasparino, C. Ellis, A. Velasquez, G. Chowdhary, and S. Gupta, "Learned visual navigation for under-canopy agricultural robots," 2021.

[36] I. Sobol, "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates," *Mathematics and Computers in Simulation*, vol. 55, 2001.

[37] D. Sun, B. Yang, and S. Mitra, "Learning-based inverse perception contracts and applications," in *ICRA*, 2024.

[38] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues, "Crashing simulated planes is cheap: Can simulation detect robotics bugs early?" in *ICST*, 2018.

[39] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, "Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles," in *ITSC*, 2016.

[40] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton Press, 2010.

[41] R. Yang, J. Laurel, S. Misailovic, and G. Singh, "Provable defense against geometric transformations," in *ICLR, 2023*.

[42] Y. Yang, R. Kaur, S. Dutta, and I. Lee, "Interpretable detection of distribution shifts in learning enabled cyber-physical systems," in *Cyber-Physical Systems*, 2022.

[43] H. Zhang, J. Sun, and Y. Tian, "Accelerated safety testing for highly automated vehicles: Application and capability comparison of surrogate models," *Transactions on Intelligent Vehicles*, vol. PP, 2023.

[44] Y. Zhao, H. Sharif, P. Pao-Huang, V. Shah, A. N. Sivakumar, M. Valverde Gasparino, A. Mahmoud, N. Zhao, S. Adve, G. Chowdhary, S. Misailovic, and V. Adve, "Approxcaliper: A programmable framework for application-aware neural network optimization," in *MLSYS*, 2023.