Abstract—Testing black-box perceptual-control systems in simulation faces two difficulties. Firstly, perceptual inputs in simulation lack the fidelity of real-world sensor inputs. Secondly, for a reasonably accurate perception system, encountering a rare failure trajectory may require running infeasibly many simulations. This paper combines perception error models—surrogates for a sensor-based detection system—with state-dependent adaptive importance sampling. This allows us to efficiently assess the rare failure probabilities for real-world perceptual control systems within simulation. Our experiments with an autonomous braking system equipped with an RGB obstacle-detector show that our method can calculate accurate failure probabilities with an inexpensive number of simulations. Further, we show how choice of safety metric can influence the process of learning proposal distributions capable of reliably sampling high-probability failures.

I. INTRODUCTION

Perceptual-control systems, such as in automated vehicles (AV), must be rigorously evaluated before deployment. Safety specifications can be defined formally in Signal Temporal Logic (STL), but the black-box nature and operating scenario complexity of such systems often preclude analytical guarantees. Instead, we seek assurances via testing.

While one can aim at the ideal of conducting exhaustive real world tests, this is infeasible [1]. In addition to the cost, these tests represent danger to the public. Alternatively, we can simulate our system, but simulation poses two core problems: First, while modern simulators can replicate real-world physics with high accuracy, perceptual inputs from a simulated sensor are far from their real-world counterparts. Figure 1 shows the photorealism gap between a real and simulated sensor inputs, outputting the expected accuracy of the target system.

A second problem is, for any modestly accurate perception system, safety violations can be extremely rare – especially if a system must fail many times in sequence to cause danger. Consider an automated braking system with an image-based obstacle detector. If the detector achieves 99% accuracy, and needs to miss an obstacle for one second to cause a crash, then at 20 images-per-second, failure probability is $(0.1 - 0.99)^20$. This means we would expect to run $1 \times 10^{40}$ simulations before observing a single crash.

Instead of direct sampling, importance sampling draws from a proposal distribution where failures are likely, then re-weights samples according to their likelihood under the target. The challenge is finding proposals where failures are likely, while producing trajectories common under the original distribution. Existing importance sampling approaches define proposal distributions over full trajectories, often failing to produce proposals capable of generating extremely rare events caused by long sequences of perception failures.

This paper proposes a method for estimating black-box perceptual-control system failures against STL specifications in simulation. To achieve this, we first train a PEM to mimic a perception system trained on real-world images. Then, we exploit this PEM to drive learning of a low-variance sampler using state-dependent adaptive importance sampling.

We apply our method to an automated braking scenario in the CARLA simulator. The perceptual-control system under test uses a YOLO obstacle detector [3] trained on images from the KITTI dataset [4]. Our experiments show that, with a representative set of training images and sufficiently descriptive salient variables, our method accurately estimates how upstream perception accuracy affects downstream safety.

Fig. 1: The difference in fidelity between real traffic images (top) and simulated images rendered in CARLA (bottom).
violations. Since multiple robustness metric choices exist for STL, our experiments also demonstrate how different metrics affect the learning process of adaptive sampling.

II. ESTIMATING THE PROBABILITY OF FAILURE

We first formally describe failure probability estimation for our perceptual-control system: Our environment contains states \( s \) in \( d \)-dimensional space \( \mathcal{S} = \{ (x_1, \ldots, x_d) | x_{1 \ldots d} \in X_1 \ldots \ldots X_d \} \). Our perceptual-control system \( \pi: \mathcal{S} \times \mathcal{A} \to [0, 1] \), will, given state \( s \), output action \( a \in \mathcal{A} \) with probability \( \pi(a|s) \). In the AV domain, variables \( X_1 \ldots X_d \) may include vehicle position/velocity, or pixel values from a camera; action space \( \mathcal{A} \) may consist of whether an obstacle detection system detects a vehicle in frame.

The probability of transitioning to state \( s' \) from \( s \) is \( p(s', a | s) = \pi(a | s) p(s' | s, a) \). This paper makes the simplifying assumption (as in [5]) that the only source of stochasticity is perception system outputs. E.g., once the perception system decides an obstacle is ahead, the decision to brake (and effect of braking on velocity) is deterministic:

\[
p(s', a | s) = \pi(a | s)
\]

Starting from \( s_0 \) we define a \( T \)-step simulation rollout as \( \tau = [s_0, a_1, s_1, a_2 \ldots s_{T-1}] \), where the rollout probability is:

\[
p(\tau) = \prod_{t=1}^{T-1} \pi(a_t | s_{t-1})
\]

We define a safety metric \( r(\varphi, \tau) \in \mathbb{R} \), where \( \varphi \) is a specification, and \( r(\varphi, \tau) < 0 \), denotes violation. Our goal is computing:

\[
\mathbb{E}_\tau [ \mathbb{I} \{ r(\tau, \varphi) \leq \gamma \}] = \int_\tau \mathbb{I} \{ r(\tau, \varphi) \leq \gamma \} p(\tau)
\]

Subsequent sections describe the components needed to estimate\( [3] \) — specifying \( \varphi \) and \( r(\tau, \varphi) \), constructing a PEM, and learning a proposal distribution.

A. Specifying Safety Requirements with STL

Signal Temporal Logic (STL) is a language for expressing properties of timed, continuous signals [6]. Its grammar is:

\[
\varphi ::= \top | \eta | \neg \varphi | \varphi_1 \land \varphi_2 | \Box \varphi | \Diamond \varphi | \varphi_1 U \varphi_2
\]

Where \( \eta \) is any predicate \( \rho(s) - b \leq 0 \) (with \( b \in \mathbb{R} \) and \( \rho : \mathcal{S} \to \mathbb{R} \)). Symbol \( \Box \varphi \) means \( \varphi \) is always true at every time-step within interval \( I \). Symbol \( \Diamond \varphi \) means \( \varphi \) must eventually be true at some time-step in \( I \), and \( \varphi_1 U \varphi_2 \) means \( \varphi_1 \) must remain true within \( I \) until \( \varphi_2 \) becomes true. For example:

\[
\Box_{[0,T]}(\|C_{\text{ego\_pos}} - C_{\text{other\_pos}}\| > 2.0)
\]

Means “From 0 to T, the distance between the ego car and other car must never drop below 2 metres”. STL formulae are paired with a robustness metric \( r(\tau, \varphi) \in \mathbb{R} \), specifying how strongly \( \tau \) satisfied \( \varphi \) \( (r(\tau, \varphi) > 0) \) or violated \( \varphi \) \( (r(\tau, \varphi) < 0) \). Classical robustness focuses on spatial robustness, applying hard minimums across conjunctions [6]. For instance, for evaluating the always operator:

\[
r(\tau, \Box_{[0,T]} \varphi) = \min \{ r(s_0, \varphi), \ldots, r(s_T, \varphi) \}
\]

For [5], this corresponds to the single state in \( \tau \) with the smallest distance between vehicles.

Alternative robustness metrics exist, each offering trade-offs in time/space robustness, soundness, and smoothness: AGM-robustness [7] replaces the min/max functions in always/evereventually with geometric/arithmetic means; Smooth Cumulative Robustness [8], combines smooth-approximations to min/max with cumulative summations.

B. Perception Error Models

In equation [2], \( \tau \) is determined by our perceptual model \( \pi(\mathcal{A} | s) \). For an obstacle detector, this could be a function \( f \) over RGB images, outputting detected or ¬detected:

\[
\pi(\text{detected} | s) = f_\theta(s)
\]

Here \( \theta \) could represent e.g., the weights of a Convolutional Neural Network trained on real traffic images. When testing in simulation however, we cannot use \( f_\theta \) directly, since our simulator does not produce real sensor inputs. We could train an alternative \( f \) based on rendered images, but as discussed in the introduction, such images correspond poorly to the real world. The added cost of including a rendering and image-processing step in the simulation pipeline also increases simulation costs by an order of magnitude.

Here is the key insight of PEMS. For simulated risk assessment to be useful, it is sufficient that, for the same state \( s \), our surrogate \( f \) and function \( f \) share the same error:

\[
\forall s \in \mathcal{S}, f_\theta(s) = \hat{f}_\theta(g(s))
\]

Where \( g \) is a projection of \( \mathcal{S} \) (which could contain RGB image pixels, or other sensor data), down to a set of salient variables — a subset of perceptual features necessary to predict system performance. This may include obstacle distance to camera, vehicle type, or level of occlusion.

We train \( f \) using \( m \) training samples \( D \), consisting of predictions from our perception system \( f_\theta(s) \) paired with the corresponding salient variables:

\[
D = \{ (f(s_0), g(s_0)), \ldots, (f(s_m), g(s_m)) \}
\]

We then find the \( \hat{\theta} \) which minimizes the binary cross entropy (BCE) between predictions \( f_\theta(s) \) and \( \hat{f}_\theta(g(s)) \):

\[
\sum_{i=0}^{m} f(s_i) \log \left[ \hat{f}_\theta(g(s_i)) \right] + (1 - f(s_i)) \log \left[ 1 - \hat{f}_\theta(g(s_i)) \right]
\]

Once \( \hat{f}_\theta \) is learned, we can use it as a drop-in surrogate for \( f \) in our simulator by projecting each \( s \) down to its salient variables as they arrive online.
C. Adaptive Importance Sampling

Given \( N \) simulations, we can estimate via monte-carlo approximation:

\[
\hat{\mu}_{\text{MC}} = \frac{1}{N} \sum_{i=0}^{N} I\{r(\tau, \varphi) \leq \gamma\} \tag{11}
\]

For rare-events, this is highly inefficient. For \( N \) samples, the relative error \( \text{RE}_{\text{MC}} \) [9] is approximately:

\[
\text{RE}_{\text{MC}}(\mu) \approx \sqrt{\frac{1}{N\mu}} \tag{12}
\]

This means to achieve a 1% relative error with event probability \( \mu = 1 \times 10^{-6} \), we need to run \( 1 \times 10^{10} \) simulations. A more efficient approach is importance sampling to draw from a proposal \( q \) where the rare-event is more likely, then re-weight each sample with respect to target distribution \( p \):

\[
\hat{\mu}_{\text{IS}} = \frac{1}{N} \sum_{i=0}^{N} I\{r(\tau, \varphi) \leq \gamma\} \frac{p(\tau_i)}{q(\tau_i)} \tag{13}
\]

Rather than defining distributions over \( \tau \), our modified importance sampler defines proposals over states \( s \in S \):

\[
\frac{1}{N} \sum_{i=0}^{N} I\{r(\tau, \varphi) \leq \gamma\} \prod_{t=0}^{T} \frac{f_{\hat{g}}(g(s_{i,t}))}{q_{\phi}(h(s_{i,t}))} \tag{14}
\]

We use \( s_{i,t} \) to denote state at time \( t \) of the \( i \)-th simulation. Following [2] and [7], we factor \( p(\tau) \) as a product of state-dependent outputs from PEM \( f \). Likewise, \( q(\tau) \) is factored as product of probabilities \( q_{\phi}(h(s)) \). Note, \( h(s) \) does not need to be the same as \( g(s) \). \( h(s) \) might project \( s \) over a small subset of features we suspect influence safety e.g., vehicle acceleration or velocity.

We want a proposal where failure events are common, and trajectories have high likelihood under the target distribution. The perfect proposal \( q^* \) has the form:

\[
q^*(\tau) = \left( \frac{p(\tau)}{\mu} \right) I_{r(\tau, \varphi) < \gamma} \tag{15}
\]

Of course, we do not know \( \mu \), but given example trajectories \( \{\tau_0, \ldots, \tau_N\} \), we can learn good proposal parameters \( \phi \) by minimizing the KL-divergence between the data and [15]:

\[
- \sum_{i=0}^{N} w_i \left( \sum_{t=0}^{T} \log q_{\phi}(h(s_{i,t})) \right) I_{r(\tau, \varphi) < \gamma} \tag{16}
\]

With weights \( w_i \) defined as:

\[
w_i = \prod_{t=0}^{T} \frac{\hat{f}_{\hat{g}}(g(s_{i,t}))}{q_{\phi}(h(s_{i,t}))} \tag{17}
\]

However, we have a problem: If no trajectory in \( \tau_0, \ldots, \tau_N \) passes threshold \( I_{r(\tau, \varphi) < \gamma} \), this halts optimization.

To address this, we apply techniques from the cross-entropy method (CEM) literature [9] — Instead of filtering all simulations by threshold \( \gamma \), we iteratively sample multiple batches of simulations. At each stage \( \kappa \in [0, \ldots, K] \), we run \( N_\kappa \) simulations, then sort \( \{\tau_1, \ldots, \tau_{N_\kappa}\} \) by \( r(\tau, \varphi) \).

In place of \( \gamma \), we set an intermediate threshold \( \gamma_\kappa = \max(\gamma, r(\tau_{\sigma N_\kappa})) \), where \( \sigma \) is a desired quantile within range 0.95 \( \leq \sigma < 1.0 \). This ensures some simulations meet the threshold at every stage, driving incremental proposal learning via [16].

Because we are finding a proposal via numerical optimization of a product of state dependent distributions (rather than e.g., a single exponential distribution admitting a closed-form solution [10], [11], [12]), and because our safety thresholds are characterized by long chains of consecutive failures, we have another problem: The difference between the highest and lowest trajectory weights \( w \) may become so enormous that the less likely trajectories have effectively have zero weight in the learning process due to numerical instability. If those less likely trajectories are necessary to achieve lower safety thresholds, this can result in the optimization stalling at an intermediate threshold \( \gamma_\kappa \).

Multiple or defensive importance sampling techniques are typically used to limit the ratio between the target and proposal likelihoods for a single state [13], but such techniques place no limit on weights \( w_i \) resulting from the product of likelihood sequences. Instead, our approach is to apply a smoothing parameter \( \alpha \in [0, 1] \) to importance weights:

\[
- \sum_{i=0}^{N} w_i \left( \sum_{t=0}^{T} \log q_{\phi}(h(s_{i,t})) \right) I_{r(\tau, \varphi) < \gamma} \tag{18}
\]

This lessens the gap between the highest and lowest weights, allowing our adaptive learning process to progress.

Algorithm 1 Simulated Risk Estimation with PEMs

1: \textbf{function} \texttt{ADAPTIVE-EST}(\( D, \varphi, N_e, N_e, \gamma \))
2: \hspace{1em} \( \theta \leftarrow \text{Train PEM using } D \text{ with } \{10\} \)
3: \hspace{1em} \( \phi \leftarrow \text{Initialize proposal distribution} \)
4: \hspace{1em} \textbf{for} \( \kappa = 1 \) to \( K \) \textbf{do}
5: \hspace{2em} \{\( \tau_0 \ldots \tau_{N_\kappa}\) \} \leftarrow \text{Sample } N_\kappa \text{ rollouts with } q_{\phi} \)
6: \hspace{2em} \text{Sort } \{\( \tau_0 \ldots \tau_{N_\kappa}\) \} \text{ by } r(\tau, \varphi) \)
7: \hspace{2em} \( \gamma_\kappa \leftarrow \max(\gamma, r(\tau_{\sigma N_\kappa})) \)
8: \hspace{2em} \( \phi \leftarrow \text{Min } \{18\} \) with \( \gamma_\kappa, q_{\phi}, \hat{f}_{\hat{g}}, \{\tau_0 \ldots \tau_{N_\kappa}\} \)
9: \hspace{2em} \{\( \tau_0 \ldots \tau_{N_e}\) \} \leftarrow \text{Sample } N_e \text{ rollouts with } q_{\phi} \)
10: \hspace{1em} \textbf{return} \( \hat{\mu}_{\text{IS}} \) using \( \{\tau_0 \ldots \tau_{N_e}\} \) with \{14\}

Algorithm 1 Combines our STL robustness components, PEM, and adaptive sampling to give our full method. After training our PEM to replicate real-world perceptual system behaviour as outlined in section (II-B), we perform \( K \) stages of black-box simulation. At each stage \( \kappa \), we conduct \( N_e \) simulations, set threshold \( \gamma_\kappa \) based on the upper quantile of rollouts, then use these rollouts to optimize parameters \( \phi \) for the next iteration. After the \( K \) iterations, we perform a final \( N_e \) simulations to estimate failure probability \( \mu \).

III. EXPERIMENTS

Our experiments apply algorithm (1) to an automated braking problem in the CARLA vehicle simulator [14].
show our method can estimate the probability of a rare collision event efficiently, yielding a greater number of high-probability failure events (compared with classical monte-carlo sampling, or a naive custom proposal). Further, we show how the choice of STL robustness metric can affect learning of the proposal distribution.

Figure 2 illustrates our scenario: Our car follows another for \( T = 100 \) time steps. At the end of the road, the front vehicle brakes, and our car must brake to avoid crashing. We use \((5)\) as our safety condition — If the distance between vehicles is ever less than 2 metres we consider it a “crash”\(^1\)

Our vehicle’s perceptual-control system comprises two main components: a physical controller and a perception system. The physical controller is a simplification of the BasicAgent from the base CARLA library\[^2\]. A global planner sets road waypoints for the car to follow; A local planner applies PID control to reach max speed if the road is obstacle-free, and applies emergency brakes if there is an obstacle within 10 metres of the vehicle\[^3\]. The perception system is a neural-network-based obstacle detector trained on RGB images of real traffic from the KITTI dataset\[^4\]. We first describe how we train a PEM from this detector, then outline its use within CARLA.

A. Learning a PEM of the KITTI dataset

KITTI is an object detection dataset of 1242 x 375 traffic images, with labels denoting vehicle type and 2D bounding boxes. It contains 7481 training and 7518 test images.

The obstacle detection system uses YOLO-V3\[^3\], trained on KITTI. Figure 3 sketches YOLO’s operation. Images are divided into regions, bounding boxes produced, and detection probabilities \( f(s) \) predicted in one pass through the network.

We next learn a PEM to act as a surrogate \( f_\theta \) for our YOLO perception system. Figure 4 shows our feed-forward PEM architecture. To train, we first pre-process the data from our YOLO-KITTI detector, separating individual obstacles and pairing them to their closest bounding box prediction via hungarian matching\[^15\]. For salient variables \( g(s) \), we use 14 dimensions based on auxiliary labels provided by the KITTI dataset for each image. The 14 dimensions are: A 6-dimensional one-hot encoding of vehicle category, a 3-dimensional one-hot encoding for the amount the vehicle is occluded (None, Partial, Mostly), the \( xyz \) obstacle location, and the rotation of the obstacle on the \( y \)-axis. After removing obstacles with unknown values, we were left with 37498 obstacles to train over. We then minimize loss function (10) using the Adam Optimizer\[^16\] over 300 epochs.

Note, our PEM training has two limits. First, by considering labels on single obstacles, our model is unable to capture false positive detections. This is a common limitation\[^2\], but could be overcome with a more expressive surrogate. Second, we assume the KITTI auxiliary labels are expressive enough to explain variance in detection performance across the domain. This is likely untrue — perception systems are affected by additional factors such as time of day, weather conditions, road surface reflections etc. In an industry-scale system deployment, we would expect to have dedicated teams of annotators to design and record a more widely expressive set of salient variables for each image. These issues lie outside the scope of this paper.

Table 1 and Figure 5 summarize our PEM’s performance (ML-NN), via 5-fold cross-validation. Several baselines are also given: LR fits a logistic regressor; Guess-\(\mu\) returns the mean detection probability. We also fit a complex Bayesian Neural Network (B-NN). Rather than finding the most likely \( \theta \), B-NN learns a distribution over parameters using variational inference\[^17\].

The original YOLO model was a strong obstacle detector, PEM accuracy is uninformative (consider that a PEM which ignored input and guessed “0% false negatives” in all situations would still achieve accuracy \(>90\%)\), so we also report the BCE and ROC for each model. Across baselines, our original model ML-NN performs best.

It is perhaps surprising that B-NN does not perform better. It may be that the variational family used to capture
the detection distribution was not sufficiently expressive. Alternatively, held-out data used in the 5-fold validation may be insufficiently different from the training data to realise the benefit of marginalizing over parameters. While not the focus of this paper, it could be interesting to investigate the performance of such PEMs in cases of distribution shift and out-of-domain images.

B. Adaptive Sampling of Rare Crash Events

We now use our PEM to drive adaptive sampling in algorithm (1). We run $K = 10$ stages, with $N_s = 100$, and $N_c = 100$. Each simulation runs for $T = 100$ time steps, with a delta 0.05. Cars start at 70 km/h, 15 metres apart.

Proposal network $q_\phi$ is a two-layer feed-forward network, with 32 hidden nodes per layer, and ReLU activations. Transformation function $h(s)$ projects $s$ down to a single dimension—the distance (in metres) between vehicles. To ensure our initial proposal produces trajectories similar to the original PEM, we pre-train $\phi$ with 100 simulations from $\hat{f}_\beta$. In adaptation stages, we minimize (18) with $\alpha = 0.1$. We use the Adam optimizer [16] for 500 epochs.

We compare our method (ADAPTIVE-PEM$_{100}$) to several baselines: MC$_{100}$ and MC$_{10000}$ estimate crash probability using 100 and 10000 raw monte-carlo samples respectively. We also compare against a naive custom importance sampler (NAIVE-50$_{100}$ and NAIVE-50$_{10000}$) which ignores state information, instead detecting obstacles with a flat 50% probability. This is likely to produce more frequent crashes, but the trajectories may be unlikely under the target.

For our robustness metrics $r(\tau, \varphi)$ used to evaluate thresholds $\gamma / \kappa$, we consider three variants—Classical Spatial Robustness ($r_c$), AGM Robustness ($r_a$), and Smooth-Cumulative Robustness ($r_s$), with $\gamma = 0$ across all three.

C. Results

Table III shows the results of each sampling strategy. Clearly, raw monte-carlo sampling is insufficient for this problem — After 10,000 simulations, MC$_{10000}$ has produced 0 failures, reporting 0% crash probability. This is unsurprising given (12) and the high accuracy of our target detector (> 90% average detection accuracy). Since conducting enough monte-carlo simulations to extract a ground-truth probability is computationally infeasible, we instead rely on two proxies—Proportion of final trajectories resulting in a crash, and average negative log likelihood (NLL) of failures.

ADAPTIVE-PEM$_{100}$ successfully learns a proposal that produces more than half of its simulations as failures, with an average NLL of 48.03. In contrast, while NAIVE-50 produces a larger failure proportion, the NLL is far lower (162.54). This means trajectories from NAIVE-50 are $1 \times 10^{50}$ times less likely under $\hat{f}_\beta$ than those from ADAPTIVE-PEM. As a result, NAIVE-50 is dominated by a few trajectories with large importance weights, leading to inaccurate estimates of $\mu$. In fact, we can see that as the number of simulations increases from 100 to 10000, NAIVE-50’s estimate gradually moves towards ADAPTIVE-PEM’s estimate.

Figure 6 shows the detection rate of ADAPTIVE-PEM as a function of vehicle distance. At distances where safety is not a concern, the proposal mimics the detection accuracy of the original PEM (between 98–99.9%). As the cars approach distances where a failure to brake would cause a crash, the proposal gradually lowers the detection probability.

For robustness metrics $r(\tau, \varphi)$, Table III shows that the final proposals and $\mu$-estimates after $K = 10$ adaptation stages is roughly similar across all three. However, as figure 7 reveals, the behaviour of the three metrics at intermediate stages varies. Consider the proposal distributions learned

| Metric | $\mu$ failures | NLL |
|--------|----------------|-----|
| $r_c$  | $3.36 \times 10^{-15}$ | 52/100 | 45.51 |
| $r_a$  | $7.12 \times 10^{-16}$ | 41/100 | 45.12 |
| $r_s$  | $1.754 \times 10^{-15}$ | 40/100 | 42.51 |
after $\kappa = 3$. Here, $r_c$ has a sharp dip in probability just below 10 metres, while $r_s$ has a wider smooth decline in probability around this zone. $r_a$ has a flatter curve which deviates less from the initial target. This aligns with our understanding of how each metric ranks trajectory robustness: $r_c$ computes robustness as a strict minimum of distances across all time steps, so will rank trajectories with the single smallest distance the smallest value. In contrast, $r_a$ applies a smooth approximation across time steps, giving smaller values to trajectories which achieve slightly higher minimum distances across a longer time-step span. Similarly, $r_a$ applies a conservative geometric mean across trajectories, leading to similar values for initial trajectories, discouraging aggressive movements from the target in early stages.

The main insight is, given limited adaptation budget, the choice of $r(\tau, \varphi)$ can influence threshold levels $\gamma_\kappa$, trajectory ordering, and potentially learning process behaviour.

IV. RELATED WORK

Many works test systems via black-box falsification or adaptive-stress testing [18], [19], [20], [21], [22]. Such works focus on finding a single failure trajectory, rather than estimating the overall probability of failure.

This paper estimated rare-event failures by augmenting CEM techniques [9]. Other works use CEM for AV risk estimation [23], [23], [12], [10]. These typically learn proposals over trajectories, an approach which can fail when rare events result from a product of probabilities over long-running trajectories. Additionally, these works often limit themselves to exponential-family proposals over a set of initial conditions; they avoid sampling perceptual behaviours of the vehicle itself. Our method instead learns proposals over individual states, using PEMS to handle sensor observations.

One state-dependent approach close to ours frames rare-event sampling as an approximate dynamic programming problem [5]. It requires the user to have prior knowledge of the transition model, and the ability to jump to arbitrary states. For black-box simulators like CARLA, neither requirement holds.

An alternative adaptive sampling technique is multi-level splitting [24], [25]. Here initial simulation trajectories are treated as particles which are perturbed using markov-chain monte carlo (MCMC). This method scales better to long-running sequential problems, but conducting MCMC transitions per trajectory per stage is intractable for simulators with non-trivial run time.

Continuation approaches use earlier perception system training epochs as a proxy for proposal stages [26]. They assume the user has access to earlier training checkpoints, and that failures in those earlier stages are similar to those encountered by the final system.

PEMS are applied in AV as surrogates for image, radar, and lidar sensors [2], [27], [28], [29]. The primary motivation of such applications is usually to bypass expensive rendering, providing fast, low-fidelity predictions. In contrast, our work takes advantage of the probabilistic outputs provided by PEMS to also drive adaptive importance sampling.

We make the core assumption that our PEM is well calibrated [30]. Several issues can challenge this assumption. First, our model may encounter epistemic uncertainty if images in simulation are far from the training distribution [31]. Second, our model may encounter aleatoric uncertainty if the salient variables insufficiently express relevant safety aspects of a scene [32]. In future work, we aim to explore this relationship between epistemic/aleatoric uncertainty in the upstream perceptual system, and the guarantees given on downstream failures.

There have been multiple attempts to encode complex traffic rules using temporal logic [33], [34], [35], [36]. In future work, we aim to scale our method from automated braking to more complex traffic scenarios, investigating the effect of balancing multiple competing rules.

For testing of black-box systems, one issue is assessing trust in your final estimation. Safety certificates help ensure risk is not under or over-estimated [37], [38], [39], [40]. These techniques are complementary to our task, and can be applied as an additional step after sampling.

V. CONCLUSION

This paper proposed a method for testing black box perceptual-control systems in simulation, even when the perception system takes real-world sensors inputs, and failures are rare. We used PEMS to drive state-dependent adaptive importance samplers, with an STL robustness metric to assess failure thresholds. Our experiments in automated braking showed our method efficiently estimates rare failure probabilities, learning distributions that generate a high percentage of high-likelihood failure trajectories.
