Safe At Any Speed: A Simulation-Based Test Harness for Autonomous Vehicles

Houssam Abbas, Matthew O'Kelly, Alena Rodionova, and Rahul Mangharam

University of Pennsylvania, Philadelphia, PA 19104, USA,
{habbas, mokelly, alena.rodionova, rahulm}@seas.upenn.edu

Abstract. The testing of Autonomous Vehicles (AVs) requires driving the AV billions of miles under varied scenarios in order to find bugs, accidents and otherwise inappropriate behavior. Because driving a real AV that many miles is too slow and costly, this motivates the use of sophisticated 'world simulators', which present the AV's perception pipeline with realistic input scenes, and present the AV's control stack with realistic traffic and physics to which to react. Thus the simulator is a crucial piece of any CAD toolchain for AV testing. In this work, we build a test harness for driving an arbitrary AV's code in a simulated world. We demonstrate this harness by using the game Grand Theft Auto V (GTA) as world simulator for AV testing. Namely, our AV code, for both perception and control, interacts in real-time with the game engine to drive our AV in the GTA world, and we search for weather conditions and AV operating conditions that lead to dangerous situations. This goes beyond the current state-of-the-art where AVs are tested under ideal weather conditions, and lays the ground work for a more comprehensive testing effort. We also propose and demonstrate necessary analyses to validate the simulation results relative to the real world. The results of such analyses allow the designers and verification engineers to weigh the results of simulation-based testing.

1 Introduction: Testing AVs in Simulated Worlds

The development of Autonomous Vehicles (AVs) has seen a remarkable acceleration in the last decade, as technological advances like Deep Learning have allowed breakthroughs in processing visual information, and regulators have come to appreciate the potential of AVs to reduce accidents. While the first wave of AV development focused on improving the performance of individual components, like the Computer Vision (CV) pipeline or the behavioral controller, there is a growing need for whole-AV testing. This is testing of the *integrated AV as a whole*, where perception, control and environment conditions interact in unforeseen and complicated ways. This is an essential step towards building technical, regulatory, and public confidence in AVs as the solution to some of our transportation problems.

Testing the real AV on real roads is a necessary part of this effort, but is woefully insufficient: a recent statistical study by the RAND Corporation (a U.S.-based policy think tank) found that AVs would have to drive "hundreds of

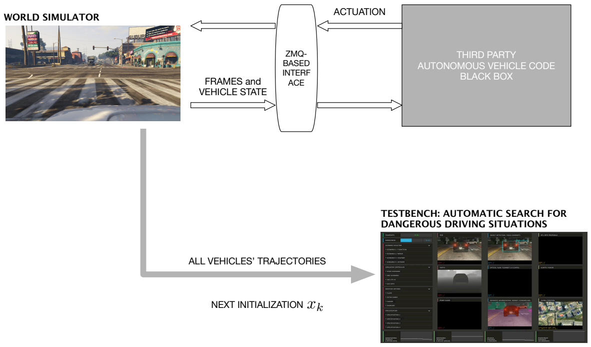

WORLD SIMULATOR
ACTUATION
FRAMES and VEHICLE STATE
ZMQ-BASED INTERF ACE
THIRD PARTY AUTONOMOUS VEHICLE CODE BLACK BOX
TESTBENCH: AUTOMATIC SEARCH FOR DANGEROUS DRIVING SITUATIONS
ALL VEHICLES' TRAJECTORIES
NEXT INITIALIZATION x_k

Fig. 1. The test harness.

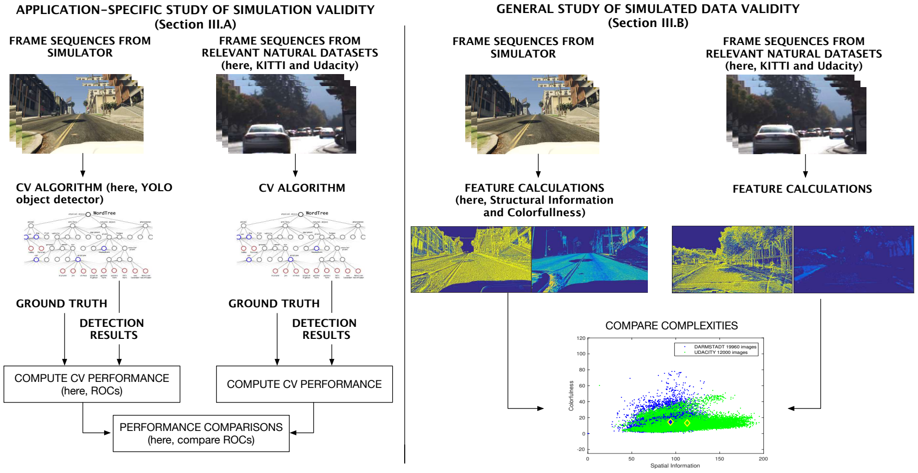

APPLICATION–SPECIFIC STUDY OF SIMULATION VALIDITY (Section III.A)
FRAME SEQUENCES FROM SIMULATOR
FRAME SEQUENCES FROM RELEVANT NATURAL DATASETS (here, KITTI and Udacity)
CV ALGORITHM (here, YOLO object detector)
CV ALGORITHM
GROUND TRUTH
DETECTION RESULTS
GROUND TRUTH
DETECTION RESULTS
COMPUTE CV PERFORMANCE (here, ROCs)
COMPUTE CV PERFORMANCE
PERFORMANCE COMPARISONS (here, compare ROCs)
GENERAL STUDY OF SIMULATED DATA VALIDITY (Section III.B)
FRAME SEQUENCES FROM SIMULATOR
FRAME SEQUENCES FROM RELEVANT NATURAL DATASETS (here, KITTI and Udacity)
FEATURE CALCULATIONS (here, Structural Information and Colorfullness)
FEATURE CALCULATIONS
COMPARE COMPLEXITIES
DARMSTADT 19960 images
UDACITY 12000 images
Colorfulness
Spatial Information
120
100
80
60
40
20
0
-20
0
50
100
150
200

Fig. 2. Analyzes of test results on synthetic data.

millions of miles and, under some scenarios, hundreds of billions of miles to create enough data to clearly demonstrate their safety". According to the same report, "it would take existing fleets of autonomous vehicles tens and even hundreds of years to log sufficient miles" to demonstrate their safety when compared to human-driven vehicles. This constitutes a definitive argument for building *world simulators*, that the AV can be driven in. A world simulator provides the AV with perceptual inputs (like video and range data), traffic conditions (like other cars and pedestrians), varied weather conditions, and moves the AV in the simulated world in response to the AV's computed actuation commands. Simulation is many orders of magnitude cheaper and faster than real-world testing.

This paper fills a gap in this regard: it demonstrates a *simulation-based test harness* for AVs, illustrates it use to automatically find dangerous situations, and clarifies the questions that must be answered when using simulation-based results for debugging real AV code. A number of companies and startups have released open-source AV code and platforms, such as Baidu's Apollo [1] and Autoware [2], to cite a few. However, the open-source community still lacks a simulator in which to test *the whole AV* in a *wide range of driving scenarios*, and a corresponding automatic testing tool that can search for dangerous scenarios. To illustrate our test harness, we use the GTA V game engine as world simulator. Recent work in the deep learning community uses synthetic scenes from GTA to *train* a neural network to perform a CV task like object detection [3] or image segmentation [4] or depth estimation [5]. By contrast, in this paper, we explore the use of synthetic scenes to *test* a *given pre-trained* algorithm (like an object detector), as part of an overall AV testing effort. Research that tries to find the most dangerous instances of *human driving* by analyzing millions of *human-driven* miles [6] is complementary to what can be accomplished using our test harness. That research highlights what miles *must* be driven by the AV to make sure it doesn't repeat human errors; our harness allows the driving of *any* kinds of miles at little cost, and we search *specifically autonomous miles* for dangerous behavior.

Fig. 2 gives an overview of the test harness and the questions we answer in this paper, and which are detailed in the following sections. Briefly, the test harness consists of a real-time communication architecture that allows connecting an arbitary AV code to a world simulator, like GTA. The simulator feeds the AV information about the state of the simulation. The AV processes this information and computes the next control inputs, which are sent back to the simulator to advance the simulation by one clock tick. The harness also includes a *testbench*: the latter computes a measure of how dangerous was the last simulation. E.g., the simplest measure of danger, which we implement, is the minimum distance between the AV and other traffic participants. Based on this value, the testbench decides on how to *initialize* the next simulation, including at what time of day it should take place. The ability to control the time of day, and thus the lighting conditions, in a simulator is a very powerful feature, since it allows us to stress the perception algorithms and the speed of reaction of the AV. Section 2 describes sample testing results that we obtain.

Using a simulator to test the AV raises questions on the applicability of the results to the real world. This can be broken down into two questions: the relation between the perception algorithms' performance in the simulated vs. the real world, and the validity of the simulated AV dynamical model vs. the real dynamics of the vehicle. In Section 3 of this paper, we explain the types of analyzes that are needed to answer the first of these two questions, and which are implemented by the test harness, as shown in Fig. 2. Briefly, they are statistical analysis of the performance of specific algorithms (Section 3.1) and a more general study of the visual complexity of scenes in the simulator vs. in the real world (Section 3.2). Section 4 concludes the paper.

2 Searching the world for non-robust behavior

A world simulator for testing AVs must have at least the following features:

- it must provide sufficiently realistic graphics so the perception pipeline is adequately tested.
- it must provide sufficiently realistic dynamics for the AV and other cars so that the AV controller's commands are implemented appropriately, and the other cars' reactions are realistic.
- it must create a variety of short-term traffic conditions for the AV to navigate (e.g., traffic at a T-junction).
- it must vary the weather conditions, since it is known that the environmental conditions can affect perception algorithms like objet detectors and image segmentors.

The test harness, which connects the AV to this simulated world, must have the following features:

- it must allow us to plug-in third-party AV code (both the perception and control components), so the AV can drive in this simulated world, and to collect all relevant data from an execution, like distance to obstacles and time-to-collision.
- it must support the plugging of general-purpose optimization algorithms, that can then be used to search for dangerous driving situations.
- it must support real-time simulation or faster.
- it must support replay of particular driving scenarios so the designers can debug the dangerous scenarios and improve the design.

We have developed a test harness that allows us to drive an AV in a simulated world, and which possesses the features described above. The harness, and the analyses it provides, are illustrated in Fig. 2, and described in the following sections. As a particular example of using this harness, we use GTA as a world simulator. We should stress that the harness can be used with any specialized simulator that supports the required, generic interface described in the next section.

2.1 Game-in-the-loop test harness

Fig. 2 shows the architecture of the software used to test our AV code inside GTA. Most AVs use machine learning algorithms in their perception pipeline, typically some form of deep neural network which performs inference on images obtained by the vehicle's cameras. In the AV domain timing is critical: for many perception tasks, an algorithm that takes more than 100 ms to execute is practically useless because both the AV's and the environment's state may change significantly in that time. As a result, most machine learning frameworks utilize GPUs in order to perform perception tasks quickly enough; the majority of such frameworks are compiled for UNIX machines. Thus, it is important that the AV code must run on a Linux machine even if the game engine does not. Moreover, separating the computational hosts enables modularity: an updated AV software stack or improved simulation engine can be swapped during the design cycle. This enables continuous comparison of software releases without changing the internal workings of the simulation engine.

A typical simulation runs as follows: The *test harness* selects an initial state of the AV (e.g., initial position, velocity, jitter, etc). It also selects initial environment conditions: number of cars, their initial positions and velocities, and time of day. The time of day is a way to control the lighting conditions: from bright and clear skies in the morning, to dark and cloudy skies later in the day. As explained in the Introduction, this is particularly important for stressing the perception pipeline. This initialization is then sent to the simulator. The testbench samples the simulation once every second: every second, the simulator sends back to the testbench the current states of all traffic participants, including the AV, and the current "video" frame. The testbench stores the state for later computation of performance objectives. The AV's perception pipeline processes the frame (e.g., to detect objects' positions and velocities), and the controller then computes the next actuation (steering angle and acceleration). The control commands are passed back to the simulator, and this loop continues until the end of simulation. In the experiments the perception pipeline consists of the YOLO object detector, but there is no inherent restriction on using other sensors' models.

Our test harness allows this to run in real-time (so 10s of simulated time require about 10s of wallclock time). Given that we visualize the simulation *as it runs*, faster than real-time is not possible. Another simulator, that can run without the graphics, could run faster than real-time. The bottleneck of the current setup is the GTA simulator, not the testbench.

2.2 Search algorithm

The test harness can be used to test the AV code as follows. First, pick a location in GTA's map. Next, define the AV state vector $x \in \mathcal{R}^5$, consisting of AV 2D position, 2D velocity and longitudinal jitter (second derivative of longitudinal velocity). The AV state can be initialized, in a given simlation, to any value in a pre-determined set X, e.g. $X = [-1,1]^2 \times [5,15]^2 \times [-5,5]$. We also define a time-of-day variable *tod*, measured in minutes, and which can be initialized to $D = \{0, 1, \ldots, 60 \times 24\}$. E.g. $tod = 0$ is midnight and $tod = 60 \times 8$ is 8 a.m. Finally, we define an environment vector $y \in \mathcal{R}^{4N}$, consisting of the positions and velocities of N other traffic participants. This can also be initialized to a set Y. Collectively, we refer to $z = (x, tod, y)$ as the *world state*, and it can be initialized to $Z = X \times D \times Y$. If the testbench initializes the test harness with a given $z \in Z$, the harness will simulate the resulting driving situation as explained in Section 2.1. The objective of the search is to find a value of z in Z such that the resulting simulation exposes dangerous driving situations, be they due to the AV's errors of control or perception, or because of unfortunate circumstances that might not have occurred to the AV designers. Indeed, even accidents that are not due to the AV's fault are informative, as they might cause the designers to equip the AV with better sensors or make it more conservative.

For illustration purposes in this paper, we define a 'dangerous driving situation' to be a state where the minimum distance between the AV and other cars or pedestrians is smaller than a nominal value. Therefore, we can now run an *optimization*: the objective function is $f : Z \to [0, \infty)$ where $f(z)$ is the minimum distance between the AV and other cars or pedestrians, in the simulation initialized at z. Our goal is to minimize f over Z: find the most dangerous situation, where the AV gets closest to moving obstacles. Of course, if $f(z_*) = 0$, then z_* actually witnesses an accident. All dangerous situations are then returned to designers to examine: did the object detector miss the obstacle? Did it detect it but too late? Did the obstacle come from behind a blind corner, if so, do we need to annotate the AV map with blind intersections? Or was the controller tuned too aggressively and an accident followed?

The AV code and simulator are treated as black boxes both due to their complexity and in order to provide a methodology which works to examine proprietary software without jeopardizing trade secrets or IP. Therefore we need to use a gradient-free optimization heuristic. While a simple uniform random sampling is always possible, in the experiments we use Simulated Annealing [7], a popular optimization algorithm that offers asymptotic guarantees (namely, as the number of simulations goes to infinity, the probability of being more than ϵ away from the global minimum goes to 0.[1]). Simulated Annealing and its variations have been successfully used in a very wide array of applications in the last 60 years. Other heuristics can be used, of course. The next section presents some illustrative results obtained by this test harness.

2.3 Optimization results

We selected a T-junction in Los Santos, the fictional city that is the setting for GTA. The objective of the AV is to make a safe right turn, and obey the Stop Sign. The simulation continues until either the objective is achieved, or a timeout (set to 20s) occurs.

The search automatically found an accident between the AV and another car in under 100 simulations. (The search space X was described in the previous section.) We can examine the exact conditions that led to the accident to understand what happened. First, let's describe the accident: the AV approaches the T-junction, and starts the right turn. Another car approaches from the left. Neither car is able to stop on time, even though they both eventually 'saw' each other. In this case, two factors contributed to the accident: first, the scenario takes place at twilight. While the YOLO object detector correctly classified the stop sign there is some delay. This delay was nevertheless enough to allow the AV to edge further into the intersection before stopping. Secondly, the other car was traveling at a speed similar to the AV's. Any faster, and it would've passed the AV before it started the right turn. Any slower, and the ego-vehicle would've been able to stop on time. In addition, the other vehicle is initially occluded and subsequently missed in several frames just as the ego vehicle makes a decision to turn.

This is an example of a non-trivial accident, where just the right conditions of timing and vechicle behavior must be present to cause the accident.

The automatic search enabled by our harness thus found environment conditions (lighting) and traffic conditions (speed of one other car) that produced

[1] Since a countable infinite set always has measure 0 in a continuous search space, uniform random sampling cannot offer such a guarantee. That said, currently known bounds on the convergence rate of Simulated Annealing are too loose in our context.

an accident. Another accident, captured from 3 different camera angles, can be found at this anonymous Dropbox link: http://bit.ly/2fe2tZq

3 Fake world, real news: on the validity of using synthetic environments for testing AVs

The described test harness allows us to test orders of magnitude more scenarios than we could in the real world, and dangerous situations (so-called 'counter-examples') that are exposed in simulation can help improve the design and flush out bugs. The natural question we need to answer is: do accidents discovered in simulation tell us something about real-world accidents? Without actually running the AV in the real world and correlating real-world results to the simulated results, it is impossible to obtain a direct empirical answer to this question. However, there are two 'big' questions that one can answer instead, and which go a long way toward establishing confidence in the simulated results. They are:

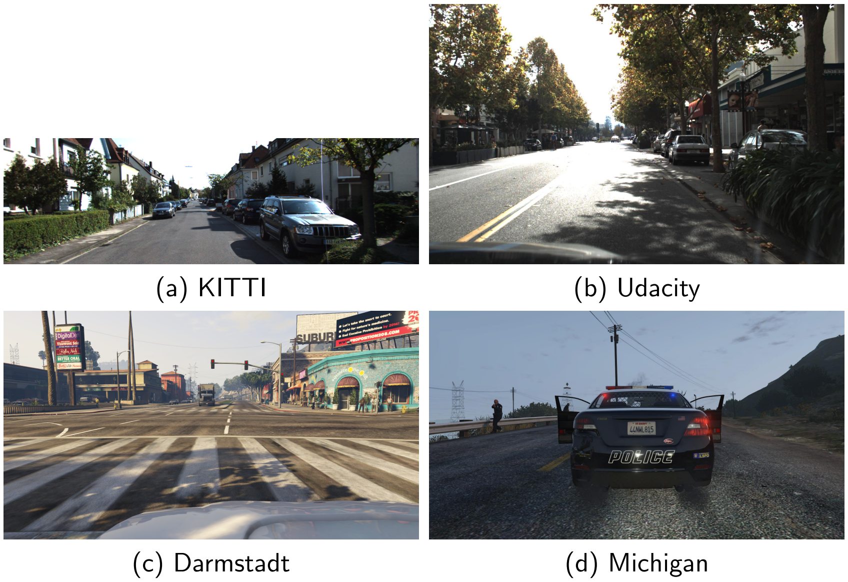
1. What is the relation between the *perception algorithms' performance* on synthetic driving scenes rendered by the graphics engine and their performance on natural ('real') driving scenes?
2. What is the relation between *the effect of AV controller's actions* in the simulator and their effect in the real world?

If we have confidence in our answers to these two questions, then we have more confidence that the whole-AV test enabled by our test harness is useful.

In this paper, we study the first question above. For the second question, it suffices to note that any dynamical model used in the automotive industry will be thouroughly validated by the automotive engineers, and the test harness we propose can accommodate any world simulator as explained earlier.

We answer the first question on two levels: first, in Section 3.1, we do a direct comparison between the performance of a perception algorithm (e.g., object detection) on synthetic and natural scenes. This gives an application-specific evaluation of the suitability of synthetic scenes for our purposes. The same study can be done on any CV algorithm. Secondly, in Section 3.2, we study the *visual complexity* of synthetic and natural scenes. Such a study is *application-independent*, and gives us a broader understanding of the differences and similarities between synthetic and natural scenes. While such a broader understanding is, at first, harder to apply than an application-specific comparison, it has a benefit: by understanding the ways in which synthetic scenes (as an ensemble) differ from natural scenes, we can better weigh the results of simulation-based testing and their relevance to real-world testing, *accross a range of peception algorithms.* E.g., visual complexity plays an important role in many computer vision algorithms, like edge detection and motion from structure. If the complexity of synthetic scenes is, say, poorer than that of natural scenes, we know that these algorithms will perform better on them, which allows us to weigh the evidence obtained from simulations.

Note that these questions are not only relevant for the case where the world simulator uses synthetic scenes, such as GTA. They apply equally to the case


(a) KITTI

(b) Udacity

(c) Darmstadt

(d) Michigan

Fig. 3. Examples of images from the datasets.

where natural scenes are used (e.g., when driving through Google Street View): as we show, the dataset of images encountered by the AV does have an effect on its performance, and any simulation-based testing must first evaluate the validity of test environment using multiple measures.

The datasets. We use the KITTI [8] and Udacity datasets [9] as sources of natural scenes. **KITTI** is extensively used in the Computer Vision and Image Processing communities to test their algorithms. We use its test set of 7481 images of urban and rural driving in and around a mid-size German city (Karlsruhe). **Udacity** is made of 15,000 images obtained by driving over Highway 92 in California during daylight conditions. For synthetic scenes, we use two sets of frames obtained from GTA: the **Darmstadt** [10] set of 25,000 frames and the **Michigan** set of 15,000 frames [3]. They were collected from the game using two plugins, Script Hook V and Script Hook V.NET [11]. The images in the datasets are highly variable in their content and layout. A range of different times of day and weather typees are captured: day, night, morning and dusk, and sun, fog, rain and haze. The Michigan dataset is annotated with the true bounding boxes for the objects in it and so can be used for profiling object detection algorithms. See Fig. 3 for example frames from these 4 datasets.

3.1 Object detection on synthetic and natural scenes

An object detection algorithm takes in an image and returns a set of bounding boxes, one box around each object it has detected in the image. See Fig. 4. It also returns the type of each detected object, e.g., ‘car’, ‘person’, or ‘stop sign’. In order to evaluate the performance of a given object detection algorithm, we

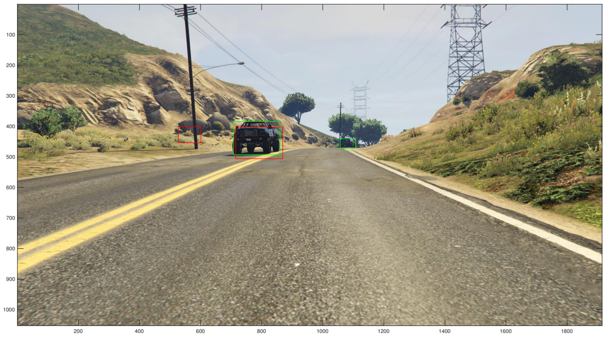
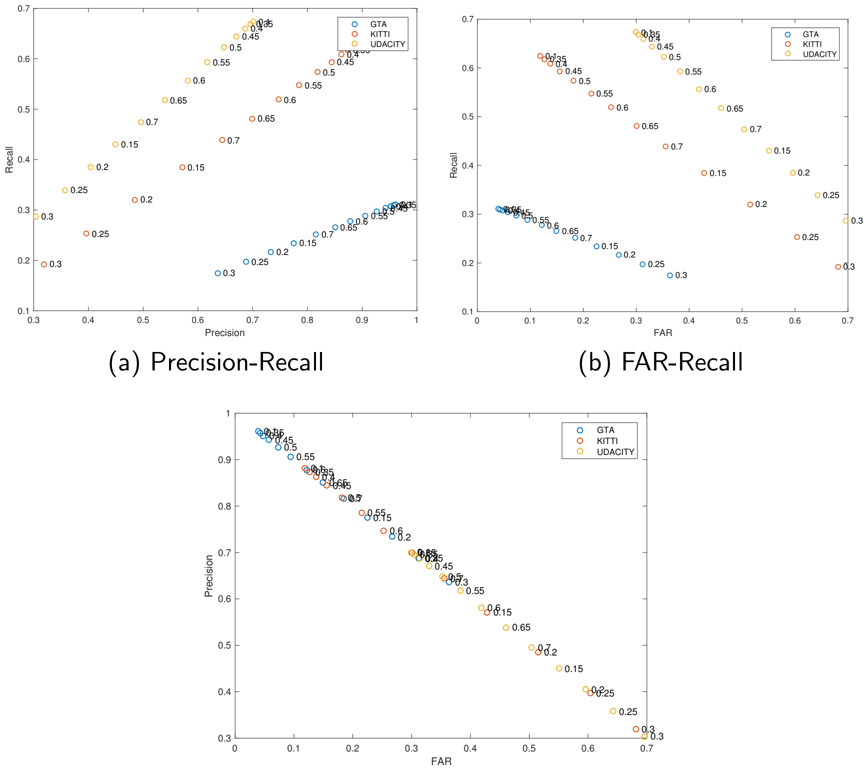


Fig. 4. GTA frame with red bounding boxes around cars detected by YOLO, and green boxes around true cars. Note the red box around the bush on the left, indicating a YOLO false positive, and the lack of red box around the faraway car on the right, indicating a YOLO false negative. The red box in the middle is a true positive.

use the three following standard metrics [12]: precision, recall and false alarm rate (FAR). These three measures belong to the interval [0, 1] and are calculated using the number $\#TP$ of true positives, number $\#FN$ of false negatives FN, and the number $\#FP$ of false positives over the given data set. A *true positive* (TP) is a detected object that is indeed in the image. A *false positive* (FP) is a detected object that isn't in the image, i.e. a mis-detection. A *false negative* (FN) is an object in the image that was not detected by the algorithm. A threshold $\alpha \in (0, 1)$ is used to compute $\#TP, \#FP$ and $\#FN$. Roughly, if a detected object's bounding box overlaps with the bounding box of a true object (of the same object type) by more than α, then this is considered to be a TP, otherwise, it's a FP. α is chosen during the network's validation phase and is fixed when running the network, therefore it is fixed in the following experiments.

We can now define the detection performance metrics: *precision* measures the fraction of detected objects that are correct: Precision $:= \#TP/(\#TP + \#FP)$. A higher precision is better. *Recall* measures the fraction of true objects that were correctly detected: Recall $:= \ \#TP/(\#TP + \#FN)$. Higher recall is better. The *False Alarm Rate* (FAR) measures the fraction of all detected objects that are not correct: FAR $:= \#FP/(\#FP + \#TP)$. *Lower* FAR is better. Note that Precision, Recall and FAR are all in the range $[0, 1]$.

Results. We measured the performance of YOLO9000 [13, 14], a popular real-time object detection algorithm, on the KITTI, Udacity and Michigan datasets, for which we have ground truth data, i.e., the bounding boxes of true objects. (We don't have ground truth for Darmstadt). Because the values of Precision, Recall and FAR depend on the threshold α, the appropriate way to compare YOLO's performance on different datasets is to vary the threshold and plot Receiver Operating Curves (ROCs). To avoid bias due to the content of the images ('content bias'), we performed this analysis on 50 randomly selected subsets of the data, each subset containing 80% of the images in the dataset. The ROCs and conclusions presented below hold accross the random selections.



(a) Precision-Recall

(b) FAR-Recall

(c) FAR-Precision

Fig. 5. YOLO ROCs for three performance measures, plotted pair-wise, for 3 datasets. Every ROC contains 13 points, one per value of the overlap threshold α. The α value is indicated next to the point.

Fig. 5 shows the results. The three performance measures are plotted against each other, two at a time. It can be seen that there is a measurable difference between YOLO on synthetic scenes (Michigan dataset) and real scenes. Indeed, there is a measurable difference *between natural datasets*. Both of these are confirmed by 2-sample Kolmogorov-Smirnov tests, which confirm that the performance numbers of different datasets come from different distributions. Thus, even if the world simulator used only natural scenes (e.g. if Google Street View is used to provide the visual input), *the question of how applicable testing results are must still be answered*.

The way to interpret and make use of these ROCs is as follows: suppose we will enforce a Precision of 0.7 during real AV operation (by selecting the right α). The Precision-Recall curve (Fig. 5a) tells us that at Precision = 0.7, GTA YOLO performance is a *lower bound* on YOLO performance in the real world (i.e., on natural images). That is, the KITTI and Udacity Recall values, for a Precision of 0.7, are both higher than the Michigan Recall value. *Thus,* *simulation results cannot mislead us*, since they are conservative. Similarly, if we enforce a FAR of 0.3 in the real world, then again GTA YOLO Recall results are a conservative lower bound on real-scenes Recall values (Fig. 5b). We will have more to say on this in the next section. Finally, the FAR-Precision ROC (Fig. 5c) reveals the noteworthy fact that the performance of YOLO on synthetic and natural scenes are nearly identical. Thus if Precision and FAR are the more important aspects of YOLO performance, simulations give a very good idea of real-world performance. *Thus a complex picture emerges, where the relation between performance in the simulator and in the real world depends on multiple factors, including on the trade-offs that the AV designers are willing to make between different performance measures.* The test harness we are presenting in this paper serves to analyze these trade-offs.

3.2 The complexity of synthetic and natural scenes

The results of the previous section might be surprising at first: after all, synthetic scenes are generally thought to be somewhat simpler, informally speaking, than natural scenes, because the latter have a greater variety of detail, texture, lighting changes, distortion and compression effects, etc. This is indirectly confirmed by studies such as [3] where, for the purposes of *training* an objet detection neural network, many more synthetic images are needed than natural images. (We are not aware of studies on using synthetic scenes in *testing* perception algorithms.) Thus it might be expected that an object detector would perform *better* on synthetic scenes than on natural scenes. However, let us first note that we used YOLO that was trained on natural scenes - which is what the real AV would use in the real world. This should temper the surprise, since YOLO is performing better on those scenes that are more 'similar' to the ones it was trained on. Secondly, in this section, we make a more rigorous study of the difference in complexity between natural and synthetic scenes.

An important property of a scene is its *visual complexity*, in terms of the density and distribution of edges, textures, colorfullness and contrast variations accross the image. A more complex scene, a priori, presents a greater challenge to any Computer Vision (CV) algorithm, because it makes it harder to extract features. E.g., a texture-rich image presents serious difficulties to an edge detector since textures can be confused for edges. In this section, we characterize the complexity of the datasets using the two complexity measures proposed in [15]: Spatial information (SI) and Colorfullness (CF). These are established measures of complexity in the Image Processing community (e.g., see their use in [16]) and they are simple to compute. Due to lack of space, we refer the reader to [15] for their mathematical definitions. Here, we mention that SI measures the strength and amount of edges in an image; edges are a crucial element of information for many image processing algorithms. CF measures the variation and intensity of colors in the image.

Results. To avoid content bias, we measured SI and CF on 50 randomly selected subsets of the four datasets, each selection containing 80% of the images. The results and conclusions presented below hold accross the random selections. Fig. 6 shows the scatterplots of complexities from one such selection.

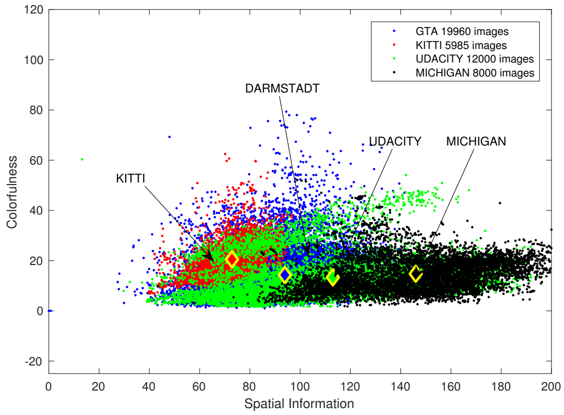


Fig. 6. Complexity of datasets. Diamonds show the centroids of the clusters. Red: KITTI, Black: Udacity (both natural) Blue: Darmstadt, Green: Michigan (both synthetic from GTA). Colors in digital version of paper.

There are clear differences between synthetic and real, but also *between synthetic datasets, and between real datasets*. The first, striking feature is that the Darmstadt (GTA) dataset complexity lies *between* the complexities of Kitti and Udacity (both real). Thus saying that 'synthetic is less complex' is too simplistic. The second feature we note is that there is a large degree of overlap between Udacity and Michigan datasets. Both have a wide range of SI, and comparatively small range of CF, which is the opposite of KITTI and Darmstadt. The complexity results suggest that using a simulated world is a reasonable means to test an AV, given the intermediate complexity of synthetic scenes, and the overlap between synthetic and some real scenes. Computer vision algorithms that are affected by the complexity of the images, like object tracking, can thus be tested in this simulated world with relevance to the real world.

4 Conclusion

The test harness we have presented allows automatic testing of AV code in simulated worlds, and implements necessary analyses for understanding the similarities and differences between the simulated data and representative real-world data. The next step is to implement a more advanced notion of AV safety, which takes into account the driving context, and to automate the debugging process for the accidents we find.

References

1. Baidu, "Apollo platform," September 2017. [Online]. Available: apollo.auto
2. Kato, Shinpei, "Autoware," September 2017. [Online]. Available: https://github.com/CPFL/Autoware

3. M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, “Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?” in *ICRA*, May 2017.
4. S. R. Richter, V. Vineet, S. Roth, and V. Koltun, “Playing for data: Ground truth from computer games,” in *European Conference on Computer Vision (ECCV)*, ser. LNCS, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9906. Springer International Publishing, 2016, pp. 102–118.
5. C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *International Conference on Computer Vision*, 2015.
6. Zhao, Ding and Peng, Huei, “From the lab to the street,” May 2017, m-City White Paper.
7. S. Chib and E. Greenberg, “Understanding the Metropolis-Hastings algorithm,” *The American Statistician*, vol. 49, no. 4, pp. 327–335, Nov 1995.
8. A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.
9. A. R. Udacity, “Udacity self-driving car dataset 2-2,” 2017. [Online]. Available: http://bit.ly/udacity-annotations-autti
10. S. R. Richter, V. Vineet, S. Roth, and V. Koltun, “Playing for data: Ground truth from computer games,” in *European Conference on Computer Vision*. Springer, 2016, pp. 102–118.
11. A. S. D. Alexander Blade, “Script hook v .net,” Sept 2017. [Online]. Available: https://github.com/crosire/scripthookvdotnet
12. Afzal A. Godil et al., “Performance metrics for evaluating object and human detection and tracking systems,” July 2014, nIST Interagency/Internal Report (NISTIR) - 7972.
13. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Conference on Computer Vision and Pattern Recognition*, 2016.
14. J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger.” [Online]. Available: https://arxiv.org/abs/1612.08242
15. S. Winkler, “Analysis of public image and video databases for quality assessment,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 6, no. 6, pp. 616–625, Oct 2012.
16. Kundu, Debarati, “Subjective and objective quality evaluation of synthetic and high dynamic range images,” May 2016, Ph.D. Dissertation.