Identification of Test Cases for Automated Driving Systems Using Bayesian Optimization*

Briti Gangopadhyay[1] Siddartha Khastgir[2] Sumanta Dey[1] Pallab Dasgupta[1] Giovanni Montana [2] Paul Jennings[2]

***Abstract*— With the growth in technology, the automotive industry is experiencing a paradigm shift from assisted driving to highly automated driving. However, autonomous driving systems are highly safety critical in nature and need to be thoroughly tested for a diverse set of conditions before being commercially deployed. Due to the huge complexities involved with Advanced Driver Assistance Systems (ADAS) /and Automated Driving Systems (ADS), traditional software testing methods have well-known limitations. They also fail to cover the infinite number of adverse conditions that can occur due to a slight change in the interactions between the environment and the system. Hence, it is important to identify test conditions that push the vehicle under test to reach or fail its safe boundary. Hazard Based Testing (HBT) methods inspired by Systems-Theoretic Process Analysis (STPA) identify such test conditions (with parametrization) that can lead to system failure. However, these techniques fall short in discovering the exact parameter values that lead to the failures. The presented paper proposes a test case identification technique using Bayesian Optimization. The proposed method identifies parameter values by learning from the system's output, for a given test scenario. The identified values create test cases that drive the system to violate its safe boundaries. STPA inspired outputs (parameters and pass/fail criteria) are used as inputs to the Bayesian Optimization model. The proposed method was applied to an SAE Level 4 autonomous Low Speed Automated Driving (LSAD) system which was modelled in a driving simulator.**

I. INTRODUCTION

Testing and certification in safety-critical domains such as avionics, medical devices, and railway systems have a long history and are well developed [1],[2]. However, the complexities involved in the automotive domain have some unique features which distinguish this domain from the ones mentioned above. Current automotive systems have over 100 million lines of code executing on 70 to 100 microprocessor-based electronic control units [3]. The deployment cycle for automotive software is typically (2-4) years whereas for avionics it is approximately 20 years. These constraints force the industry to adopt different processes and standards for proper verification and validation. ISO 26262, which is an industry recognized state of the art standard providing guidance on testing, does not define a quantifiable and rigorous process for testing Advanced Driver Assistance Systems (ADAS) and Automated Driving Systems (ADS) [4]. Crash testing strategies developed by the National Highway Traffic Safety Administration (NHTSA) and The Insurance Institute for Highway Safety (IIHS) are also not exhaustive in nature as it is not manually possible to generate all the crash cases that an autonomous vehicle would face in its lifetime [5].This complexity increase with the system's interaction with its subsystems and different dynamic environment parameters. In addition, non-deterministic and statistical algorithms are an inherent part of building intelligent driving systems. The behaviour of these algorithms is often a black box to the testing team. From a statistical perspective, it is suggested that Autonomous Vehicles (AVs) will need to be driven for more than 11 billion miles to have a 95% confidence that AV's are 20% safer than their human counterparts [6]. Even if this was possible, it would not cover the scenarios that we are not aware of (unknown unknowns) or the black swan scenarios [7] and hence would not guarantee safety. The recent crashes of Uber and Tesla [8] systems exhibit the requirement for intelligent testing to uncover the black swan scenarios. Such testing can be done in simulation by testing the vehicle in different hazardous scenarios which are parameterized. However, simulations are often expensive and intelligent identification of these parameterized scenarios is required. Since, parameter value identification is done in the context of test scenarios and test cases, the following section highlights the relationship between them.

A. Use Case, Test Scenario and Test Case

Testing of any system starts with respect to a *use case* where *use case* is a set of actions that take the system under test from one state to another linking the result to a particular actor [9]. A *use case* can have multiple *test scenarios*, which provide quantitative description of the subject vehicle, its activities and/or goals, its geo-spatially stationary environment (scenery), and its dynamic environment [10]. Each *test scenario* can be further associated with multiple *test cases*, which are defined as the set of preconditions and inputs parameters related to the system under test and agents interacting with it including actions that drive the system under test to meet a test objective. Essentially *use case*, *test scenario* and *test case* follow a pyramidal structure as defined in [9]. For example, a subject vehicle reducing its speed when a pedestrian is crossing the road and the pedestrian is occluded would be a *use case*. Dynamic element like pedestrian behaviour describes one test scenarios related to this use case. Concrete values of throttle, steering commands issued by the subject vehicle comprises of the test case.

The major contributions of this paper are:

*This work was done in collaboration between Indian Institute of Technology Kharagpur and WMG, University of Warwick

[1]Indian Institute of Technology Kharagpur, India
[2]WMG, University of Warwick, UK

1) A methodology for using Systems Theoretic Process Analysis (STPA) for setting the context for a test scenario and using Bayesian Optimization to find meaningful combinations of test parameters that drive the system under test to extremal behaviours.
2) The first approach finds one test case of extremal nature, but in reality, many combinations of parameter values can lead to failure cases. We extend the Bayesian Optimization algorithm to identify multiple combinations of parameter values for a non-convex function leading to the failure condition.

II. Related Work & Background Study

Traditionally, the safety of a system is either assured via software testing methods such as black box and white box testing or use of formal verification techniques that mathematically prove properties known as specifications for cyber-physical systems [11]. The introduction of Responsibility Sensitive Safety (RSS) [12] helps in formulating a mathematical model for devising clear rules for faults based on the common sense of human judgment. If the rules are predetermined, then investigation of safety properties can be done via formal verification of the system, and safety can be determined conclusively. However, RSS in its current form suffers from scalability as it is difficult to have a predetermined knowledge of all the actors in the environment to derive mathematical formulation of every unsafe situation. The following section discusses relevant white box and black box approaches present for identification of corner cases that can lead to failure.

A. White Box Approaches

Ontologies or knowledge graphs that are abstract simplified representations of the world are used for situation assessment and behaviour planning. Ontology-based processes can be extended for test scenario creation [13]. However, complete functional and operational knowledge gathering, required for such modeling, is not always possible since operational rules vary from region to region. Goal guided techniques such as AI Planning can also be used for test case identification by extending classical AI planning with environment and control actions. The action set and the domain propositions are identified from the requirement specification of the system. The planning path generated by the planner from the initial state to goal state produces the test cases that would cause the system to fail. Such a planning formulation has been studied on adaptive cruise control systems [14]. The major drawback of such white box approaches are i) curse of dimensionality, i.e. even if the depth of the search tree is bounded by a simulation time k the breadth would increase exponentially with the addition of each element in the scenario, ii) Computational complexity, such computations are sometimes impossible considering unbounded numeric state variables and continuous change (planning with hybrid systems) [15], iii) Having prior knowledge of the system which captures all the uncertainties generated from the interaction between environment and ADS/ADAS.

B. Black Box Approaches

Adversarial attacks have been considered as a method of test case generation for testing of closed loop safety-critical systems. Methods explained in [16] [17] generate perturbed or synthetic images as a test case for the underlying perception algorithm of the ADS/ADAS. However such local perturbations or static image generation do not cover the entire test space for dynamic systems. Minimization of robustness cost functions such as time to collision using standard optimization engines such as S-TaLiRo [18] can also be used for test case identification. These techniques require the system boundaries to be predefined and do not account for changing environmental conditions. An often-simpler problem to solve, than searching the entire test space, is the falsification of a property, which should hold in all simulations using an optimization algorithm. Any optimization technique relies on the development of search algorithms which intelligently sample the uncertainty space in order to reduce query to the simulation engine. Several heuristics based sequential search algorithms such as Simulated Annealing [19], Tabu search [20], and CMA-ES [21] have been explored in this domain. However, information gathered during previous simulations is not taken into account by these algorithms. One of the techniques that has been gaining a lot of attention for optimization of black-box systems is Bayesian Optimization (BO) [22], a technique that finds the global maxima/minima based on stochastic evaluations of an unknown function. This technique has been actively used in hyper-parameter optimization for neural networks, combinatorial optimization, optimization of parameters in robotics and even generation of adversarial counterexamples for complex controllers [22],[23],[24].

The presented study uses test scenarios having a number of uncertain bounded parameters and a pass criterion to identify test cases using Bayesian Optimization.

C. Identification of test scenario using STPA

STPA is a hazardous scenario identification method which is ground with systems theory and control theory [25],[26], designed to analyse safety in a socio-technical system with diverse interacting elements [26]. With foundations in systems based approach, STPA identifies broader range hazards, which may occur due to a variety of reasons including component failures, component interactions, human-error, human-automation interaction, software issues, incorrect requirements and even socio-technical and organisational factors.

- **Step 1**: The first step is to define the purpose of the analysis. This system involves defining the system (at a higher level) that is to be analysed. It also involves identifying high level losses or accidents for the system which need to be avoided along with potential hazardous states
- **Step 2**: This step involves creating a hierarchical control structure model of the system, capturing the functional

interactions by creating a set of nested feedback control loops between the sub-systems.

- **Step 3**: Once the control actions (CA) in the control structure have been identified, each CA is analysed to understand how a CA would manifest into an Unsafe Control Action (UCA). As per STPA, this can happen in four general conditions: i) Not providing a control action ii) Providing a control action iii) Providing a control action too late, too early or out of sequence iv) Control action stopped too soon or applied too long.
- **Step 4**: This step identifies the causal factor and control flaws. The process model and its variables are studied to understand how each UCA could occur. The Pass criteria are obtained from the negation of process model belief and the reason for the belief causing the UCA.
- **Step 5**: This is an extension [27] of STPA and deals with scenario parametrization . The inputs to the proposed test generation framework are obtained in this step. Parameterization is done by providing context to the unsafe control action, where context can be either dynamic in nature (elements that change their state at a continuous rate or abruptly.) or scenery elements (elements that do not change their state over a given span of time.).

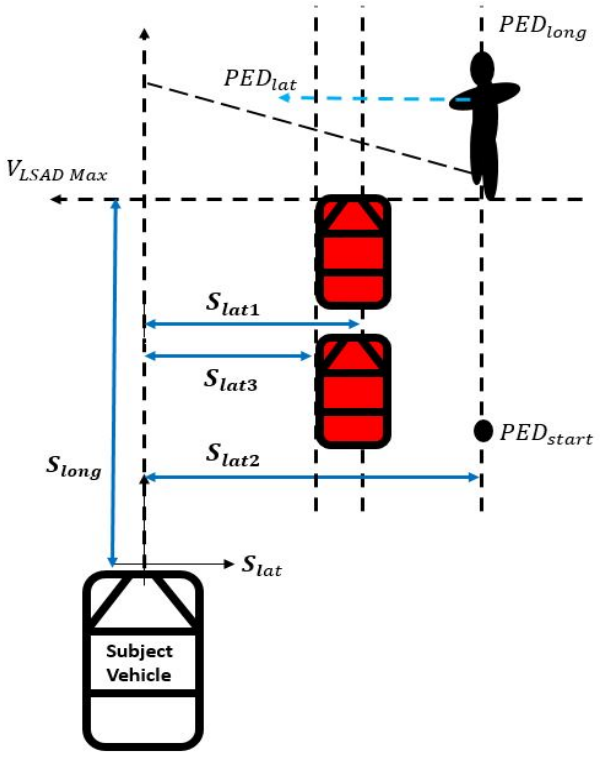

Fig. 1: Scenario generated from STPA. Subject vehicle is the vehicle under test. The pedestrian starts at Ped_{start} position and is occluded from the field of view by red cars. $Slat_1$ is set to be 3m. $Slat_3$ depends on the choice of the second occlusion object.[1]

1.https://www.euroncap.com/en/vehicle-safety/the-ratings-explained/vulnerable-road-user-vru-protection/aeb-pedestrian/

For example, the scenario depicted in Figure 1 is generated from the STPA analysis of a LSAD system. The subject vehicle (SV) is treated as a black box. Context to the pedestrian is provided by varying the speed and the angle of the pedestrian. The Unsafe Control Action is to provide a deceleration rate lower than required to avoid the collision once the pedestrian is detected at a particular speed. The pass criterion is to avoid collision between the pedestrian and the subject vehicle.

D. Pass Criteria as an optimization objective

The formal methods community uses axiomic notations and predicate calculus to specify complex properties that a system under test should satisfy during all simulations of a specific test scenario. These properties can be complex and temporal in nature. Since the test generation framework samples parameters that cause violations of system's safe boundaries, the violation of pass criteria obtained from step 4 of STPA is treated as the optimization objective following [24]. W is a set of bonded parameter values from which exact parameter values w need to be identified. We use ρ(w) to denote the objective function, namely the manifestation of the pass criteria in terms of the exact parameter values w. The objective pushes the optimization algorithm to sample parameters such that ρ(w)<0. ρ is a logical combination of multiple individual constraints, called predicates. These predicates are combined using a grammar of logical operations: $\rho := \mu \mid \neg\mu \mid \mu\wedge\mu \mid \mu\vee\mu$ where μ is a predicate and is assumed to be a smooth and continuous function of a trajectory ξ. The constraint $\mu < 0$ forms the basis of violation of the overall system specification ρ. A predicate is falsified if μ(w) over ξ is less than 0 or satisfied otherwise. Since μ is a real valued function, it can be converted into an equivalent equation with continuous output,

$$\begin{aligned} \neg\mu(w) &:= -\mu(w), \\ (\mu_i \wedge \mu_j)(w) &:= min(\mu_i(w), \mu_j(w)), \\ (\mu_i \vee \mu_j)(w) &:= max(\mu_i(w), \mu_j(w)). \end{aligned}$$

For example : If the pass criterion for a scenario is given as speed shall be less than 10kmph when fuel is less than 20% then the pass criterion in formal specification would be given by (fuel<0.2 $\rightarrow$ speed<10). Let us assume fuel is bounded from 0 to 100% and speed from 0 to 50km/hr. Here W is {fuel,speed}. Hence, ρ(speed,fuel) = $\neg$(fuel<0.2) $\vee$ (speed<10). Following the definition ρ can be represented in terms of two predicates:

1) μ_1(fuel) = fuel - 0.2
2) μ_2(speed) = 10 - speed

Rewriting ρ in terms of μ:
ρ(speed,fuel) = μ_1(fuel) $\vee$ μ_2(speed) = max(fuel - 0.2,10 - speed)
The objective is to find min($\rho(w)$) which will ensure both μ_1 & $\mu_2 \leq 0$ and violate the pass criteria. The worst violation will happen when both the predicates simultaneously reach minimum value, i.e. when fuel is 0 and speed is 50km/hr.

Similarly, for Figure 1 we must have ρ(vehicle speed, pedestrian speed, pedestrian angle, deceleration rate)≤ 0 for collision to happen.

E. Bayesian Optimization

Bayesian optimization is a class of machine-learning based optimization methods focused on solving the problem

$$\max_{x \in A} \quad f_0(x)$$

It is an approach for optimizing objective functions that take a long time (minutes or hours) to evaluate. In the heart of this algorithm lies Gaussian Process (GP) models that are used to derive a prior over the black box function that is being optimized. Gaussian processes are nonparametric regression methods that assume the function value of any $\mu_i(w)$ to be random variables such that any finite number of them can be modelled by a joint Gaussian distribution. The mean of the distribution $m(w)$ is initially assumed to be at 0 and the covariance $\kappa(w_i, w_j)$ is given by a squared exponential kernel $e^{-(|w_i - w_j|)^2}$. From such a joint distribution for every unobserved value w_* the mean m and variance σ^2 can be estimated [22].

$$m(w_*|w) = m(w_*) + K_*^T K^{-1}(y - m(w))$$
$$\sigma^2(w_*|w) = K_{**} - K_*^T K^{-1} K_*$$
$$K_* = \kappa(w, w_*), K = \kappa(w, w), K_{**} = \kappa(w_*, w_*)$$

This surrogate function along with assumed mean and variance of unobserved points is sampled using an acquisition function to derive a posterior distribution. The acquisition function chosen in the presented study is Expected Improvement which accounts for the size of the improvement while exploring and exploiting the function to find a global minimum. If μ_i'(w) is the minimal value of $\mu_i(w)$ observed so far. Then the reward utility function is given by

$$u(w) = \max(0, (\mu_i(w) - \mu_i{}'(w)))$$

And the expected improvement utility function is.

$$\alpha_{EI}(w) = E(u(w)|w, D)$$

III. Methodology

Given a black box system, a constrained environment, and a set of bounded parameters, W, the test case generator needs to find the limiting parameter values w $\in \{\mathrm{W}\}$, such that the continuous trajectory ξ, starting from initial condition vector, I, under the choice, w, refutes the pass criteria ρ, that is, $\xi \not\models \rho$. Since the underlying function being estimated maybe nonconvex in nature all the test cases causing failure need to be identified. Algorithm 1 uses background theory discussed in previous sections for test case generation. The inputs are identified from the STPA analysis and passed through Bayesian Optimization module for identification of exact parameter values that leads to failure. After each iteration, the samples are sent to the simulator to obtain a stochastic evaluation and the process continues until the specification is violated or time budget is exhausted.

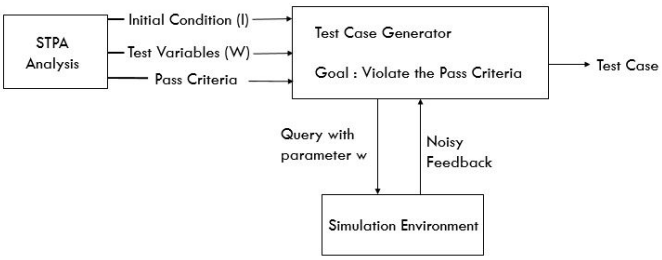

Fig. 2: High level diagram of the test case generation framework.

Algorithm 1 Sampling Using Bayesian Optimization

Input: Bounded Parameter vector W from STPA 3
Pass criteria from STPA step 2
Initial Condition vector : I

procedure DERIVEPARSETREE(PassCriteria)
  $\rho \leftarrow PassCriteria$
  for <each predicate μ in ρ> **do**
    BayesianOptimization(μ, W, I)
  end for
end procedure

procedure BAYESIANOPTIMIZATION(μ, W, I)
  for <n = 1,2,....> **do**
    Select new $w_{n+1} \in W$ by minimizing acquisition function α
    $w_{n+1} \leftarrow \arg\min_{w \in W} \ \alpha(w)$
    $D_{n+1} \leftarrow D_n(w_{n+1}, y_{n+1})$
  end for
  return w_i for which we get the minimum y_i
end procedure

Comments:
α : Expected Improvement (acquisition function).
D_n : Observed (input,output) pair from sampling.
y_i : Output of the system for i^{th} input.

We consider Figure 1 as a case study and identify parameter values of dynamic elements like pedestrian speed, angle, subject vehicle speed, deceleration rate using Bayesian Optimization. Typically different combinations of parameter values for the same test scenario can lead to different manifestations of the same failure condition. Bayesian Optimization, which is at the heart of our methodology is designed to find a global minima, which is one of many test cases leading to the failure condition. We propose a methodology to direct the search to find other minima of the continuous function $f : \mathrm{W} \rightarrow \mathbb{R}$, where W is a compact subset of $\mathbb{R}^d$.

A. Identifying Multiple Minima Regions

The key to identifying multiple minima is to eliminate the minima already explored from the search space. We do this

by leveraging the fact that the minima are (by definition) of negative sign (see Figure 4 for a pictorial view). Therefore, if we square the function, then all the minima rise above zero. The points originally at zero remain at zero.

Algorithm 2 Extension of Algorithm 1 for identifying multiple minima

Input: λ
Global Variable :
$regionsStack \leftarrow [[W_{1LB}, W_{1UB}], ..., [W_{nLB}, W_{nUB}]]$

1: **procedure** FINDMULTIPLEMINIMA(λ)
2: **while** true **do**
3: **if** $regionsStack$ is empty **then**
4: `return` $minimaList$
5: **end if**
6: $region \leftarrow pop(regionsStack)$
7: $minPoint \leftarrow BaysOptMinPoint(region)$
8: **for** `<i = 0, 1 ... dimension>` **do**
9: $cutoffRegL \leftarrow minPoint[i] - \lambda$
10: $lowB_i \leftarrow findZeroPnt(cutoffRegL, i, 0)$
11: $cutoffRegU \leftarrow minPoint[i] + \lambda$
12: $upB_i \leftarrow findZeroPnt(cutoffRegU, i, 1)$
13: $zeroPnts.add([lowB_i, upB_i])$
14: **end for**
15: $omittedRegions.add(zeroPnts)$
16: **for** `<i = 0, 1 ... dimension>` **do**
17: $newReg1.add(zeroPnts[0..(i-1)])$
18: $newReg2.add(zeroPnts[0..(i-1)])$
19: $newReg1.add([regLBound, zeroPnts[i][0]])$
20: $newReg2.add([zeroPnts[i][1], regUBound])$
21: $newReg1.add(region[(i+1)..dimension])$
22: $newReg2.add(region[(i+1)..dimension])$
23: **end for**
24: **if** $newReg(1,2)$ not in $omittedRegions$ **then**
25: $regionsStack.push(newReg1, newReg2)$
26: **end if**
27: **end while**
28: **end procedure**
29: **procedure** FINDZEROPNT($cutoffRegion, i, lowUpFlag$)
30: **while** not nearest 0 **do**
31: $zeroPnt \leftarrow BaysOptZeroPnt(cutoffRegion)$
32: $bound_i \leftarrow zeroPnt[i]$
33: **if** $lowUpFlag$ equals 0 **then**
34: $cutoffRegion[i][0] \leftarrow bound_i$
35: **else**
36: $cutoffRegion[i][1] \leftarrow bound_i$
37: **end if**
38: **end while**
39: `return` $bound_i$
40: **end procedure**

We find the zeros surrounding the minima in each parametric direction and eliminate the intermediate points (namely, the valley containing the minima) from further consideration in the search. Thereafter, a new search begins in each parametric direction, and this continues in a recursive manner. This section elaborates this methodology. We use $[W_{kLB}, W_{kUB}]$ to denote the domain of parameter $W_k \in W$. Essentially the search happens over the n-dimensional hyperspace defined by the parameters, $W_1, \ldots, W_n$. Our algorithm uses a stack called $regionsStack$ to store the set of regions to be explored. The algorithm starts with the entire region. After discovering each minimum, the region in which the minimum was discovered is split in a way to eliminate the valley containing the minimum. For simplicity, we consider hyper-rectangular abstractions around the minima for elimination. The algorithm terminates when $regionStack$ is empty.

We use a step size, denoted by λ, to demarcate the region surrounding the discovered minimum within which we look for the zero (in each parameteric direction). These zeros then define the rectangular constraints fencing the region to be eliminated from further consideration. For every dimension, two exclusive new regions are constructed as upper and lower search regions. The coordinates of the new search regions for each i dimension is defined by zero points of previous $i-1$ dimension, zero point of i and lower, upper bounds of $i+1$ dimensions. If in some direction, no zero is found within a distance λ from the minimum, then the search resumes with a larger λ. The choice of λ is kept as a hyper-parameter and is set depending on domain knowledge.

The steps of Algorithm 2 is illustrated with a one-dimensional function. $f(W) = \sin(2 * \pi * W/F_s)$ where a F_s = 1600Hz and W ranges from [0,5000]. $f(W)$ has three minimum points : $f(1200)$, $f(2800)$, $f(4400)$ = -1. The first minimum value is identified at w_m = 1200 by procedure BaysOptMinPoint(Algorithm 1). A step length λ = 800 (period/2) is chosen. Procedure findZeroPnt searches nearest points where function value is 0 each for $w_m - 800$ to w_m and w_m to $w_m + 800$. The observation value used by

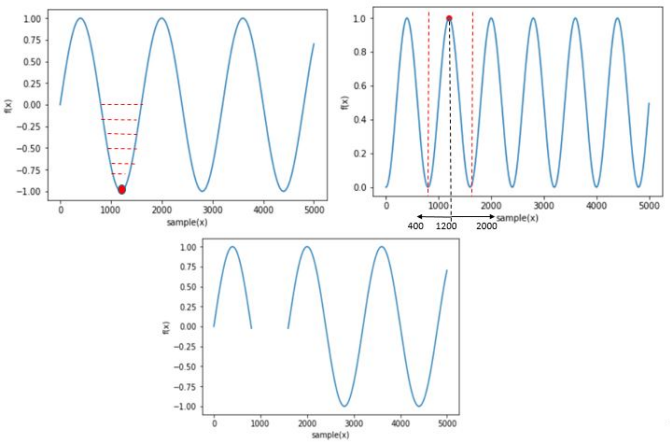

Fig. 3: From top left to bottom right a)The red region specifies the region to be eliminated, the red point is the first identified minimum point b) The squared function where all the minima lie above 0 the red lines denote nearest zeroes c) The two new search regions after elimination

BaysOptZeroPoint method is a square of the actual observed function value and the rest of its functionality is same as Algorithm 1. Observing squared value makes the underlying Gaussian Process believe it is estimating the square of the actual function without hampering the continuity of the function. Once the lower zero point w_l and upper zero point w_u are identified the next search for minima is done from ($w_{lowerbound}$,w_l) and (w_u,$w_{upperbound}$). The region between (w_l,w_u) is eliminated from search space. This process continues until all the regions containing a minima are identified. The algorithm is tested on different non convex functions as discussed in the results section. The convergence proof of the algorithm is discussed in the appendix.

IV. Results

This section discusses the results of experiments on non-convex functions and test scenarios.

A. Identifying multiple minima regions for nonconvex functions

The first function is the Holdertable function defined by $f(x,y) = -|sin(x) * cos(y) * e^{|(1-\sqrt{(x^2+y^2)}/\pi)|}$ where four global minima occur at (x,y) = (8.05502, 9.66459), (8.05502, -9.66459), (-8.05502, 9.66459), (-8.05502, -9.66459) and for all these f(x,y) = -19.2805. The second function is the Eggholder function defined by f(x,y) = $-(y+47) * \sin\sqrt{|y+x/2+47|} - x * \sin\sqrt{|x-(y+47)|}$. This function has one global minima at f(512,404.2319) = -959.6407. Both the functions have multiple local minima.

TABLE I: Performance of Algorithm 2 on different non convex functions

Function	Domain	λ	Identified Minima Regions
1	x:[-10,10],y:[-10,10]	2	56
2	x:[0,512],y:[0,512]	100	275
3	Each x_i:[-50,50]	50	512

The third function is of the form $f(x_1, x_2....x_n) = x_1 * x_2 * .... * x_n$ where the number of minima increase with increase with dimension as 2^{d-1}. The algorithm is tested upto 10th dimension.

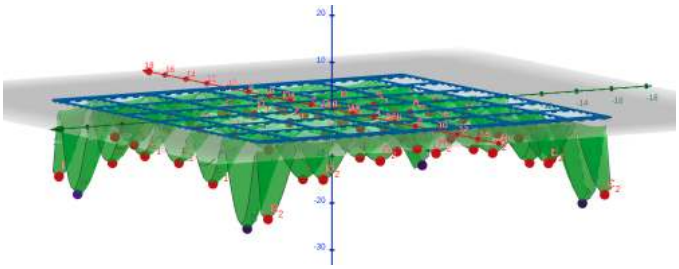

Fig. 4: Visualization of Holdertable function after running Algorithm 2. The purple dots are the identified global minimas and the red dots are local minimas. The blue rectangles denote the regions eliminated.

B. Testing the pedestrian scenario generated from STPA

The scenario described in Figure 1 is modelled in IPG Carmaker[1]. The domain of uncertainty comprises of the subject vehicle velocity [40km/hr,60km/hr], the pedestrian speed [5km/hr,20km/hr], the pedestrian crossing angle [0^o,10^o], the deceleration rate after stop command is executed [3m/s^2,6m/s^2]. As objective function the Euclidean distance between the latitude and longitudinal distance of the car and pedestrian is minimized i.e. collision occurs when $\sqrt{(car.y - ped.y)^2 + (car.x - ped.x)^2} \leq 0$. A safety envelop of 5m is considered longitudinally, i.e. the function value would be -5 when the collision occurs with the car and 0 at the beginning of the safety region.

Fig. 5: Collision detected in simulation at parameter values subject vehicle velocity = 58.06km/hr, pedestrian speed = 17.82km/hr, Crossing angle = 4.57^o, deceleration rate = 3m/s^2. This test case gives a function value of -5

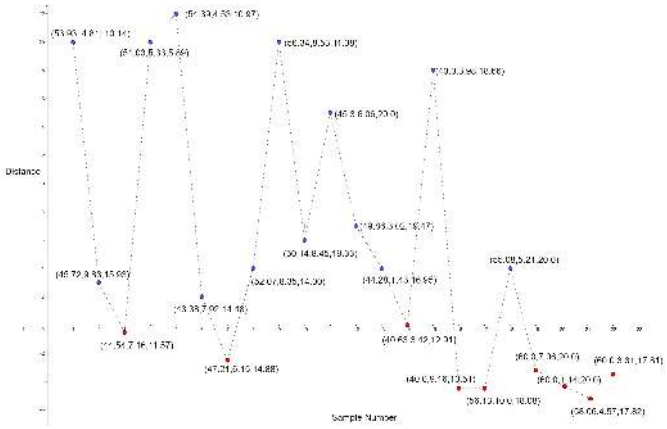

Fig. 6: Plot illustrating function value of each sampled point up to 22 samples. Deceleration for the plot is kept fixed at 3m/s^2. Each point represents (subject vehicle speed, Pedestrian angle, Pedestrian speed). One of the left zero for the identified minima is found at [58.05,4.57^o,15.69] and right zero at [58.05,4.57^o,19.83].

Certain parameter values of the scenario are predetermined

[1]http://ipg.de/de/simulationsolutions/carmaker/

for example the radar sensor range is set at 15m and the distance between first occlusion object and subject vehicle at 3m. The dynamics of the subject vehicle is a black box. The search space for this problem is $20*15*10*3*(10^2)^3$ (upto 2 decimal places of each parameter is except deceleration rate considered). Each point in the search space can form a test case. The step size λ is chosen to be (domain range)/2 for each domain. This can be further tuned for faster convergence. Multiple test cases are identified like [58.06,4.57^o,17.82,3],[47.21,6.15^o,14.88,3], [60,0^o,20,4], [60,10^o,18.59,3], [42.36,7.19^o,11.64,4] etc. One such test case as is illustrated in Figure 5. No violations are observed over a deceleration rate of 5m/s^2. Hence, a safe driving strategy for this scenario would be to have a deceleration rate of more than 5m/s^2.

V. Conclusion & Future Scope

The presented study discusses a methodology to identify test cases for complex black boxes like autonomous systems with fast convergence. This type of analysis can be further use to define safe strategies for autonomous driving systems. The algorithms can be further optimized by tuning hyper-parameters such as λ and searching new regions in parallel. In future, the authors intend to extend this work to identify parameters values for temporally changing functions and testing the methodology over more complex scenarios.

Appendix

Assumimg the function $f \in$ reproducing kernel Hilbert Space (RKHS) $\mathcal{H}$ described by the Gaussian Process prior π. Algorithm 2 is shown to converge for n minima in f considering finite size of domains and fixed λ with the help of induction.

Basis : for n = 1 continuous function $f \in \mathcal{H}$ and has only one global minima. when π is a fixed Gaussian process prior of finite smoothness, expected improvement converges on the minimum of any $f \in \mathcal{H}$, and almost surely for f drawn from π. [28].

Hypothesis: Assuming convergence for n = k

Inductive Step: for n = k+1 For each k minima points a continuous subpart of the original function is subtracted from the original function using a region described by the d dimensional hyper rectangle. Let each subpart be denoted by $g_i(x)$. Then, the function after k iteration is given by $f'(x) = f(x) - \sum_{i=1}^{k} g_i(x)$ $f'(x) \in \mathcal{H}$ and is continuous containing only one minima which is same as the base case. Hence, the method converges for n minimum points.

References

[1] Rushby J., "New Challenges In Certification For Aircraft Software,"in ACM International Conference On Embedded Software (EMSOFT),2011.

[2] Mller J., Drewes J., May J. and Trog C., "The formal representation of the safety case processes described in the EN 5012x norms," in European Safety and Reliability Conference, 2009.

[3] R. Charette, "This Car Runs on Code", IEEE Spectrum: Technology, Engineering, and Science News, 2018. [Online]. Available: https://spectrum.ieee.org/transportation/systems/this-car-runs-on-code. [Accessed: 15- Aug- 2018].

[4] Yu, H., Lin, C., and Kim, B., "Automotive Software Certification: Current Status and Challenges," SAE Int. J. Passeng. Cars Electron. Electr. Syst. 9(1):74-80, 2016, https://doi.org/10.4271/2016-01-0050.

[5] Transport Systems Catapult, Taxonomy of Scenarios for Automated Driving, 2017.

[6] Kalra, N. and Paddock, S.M., Driving to Safety: How Many Miles of Driving would it Take to Demonstrate Autonomous Vehicle Reliability?, Transp. Res. Part A Policy Pract. 94:182-193, Dec 2016, doi:10.1016/j.tra.2016.09.010.

[7] Khastgir, S., Birrell, S., Dhadyalla, G., and Jennings, P., The Science of Testing: An Automotive Perspective, SAE Technical Paper 2018-01-1070, 2018, doi:10.4271/2018-01-1070.

[8] Williams, E. (2018). Fatalities vs False Positives: The Lessons from the Tesla and Uber Crashes. [online] Hackaday. Available at: https://hackaday.com/2018/06/18/fatalities-vs-false-positives-the-lessons-from-the-tesla-and-uber-crashes/ [Accessed 1 Dec. 2018].

[9] Khastgir, S., Dhadyalla, G., Birrell, S., Redmond, S., Addinall, R., & Jennings, P. (2017). Test Scenario Generation for Driving Simulators Using Constrained Randomization Technique. WCX 17: SAE World Congress Experience. https://doi.org/10.4271/2017-01-1672.Copyright

[10] Ulbrich, S., Menzel, T., Reschka, A., Schuldt, F. & Maurer, M. Defining and Substantiating the Terms Scene , Situation, and Scenario for Automated Driving. (2015). doi:10.1109/ITSC.2015.164

[11] V. Raman, A. Donze, M. Maasoumy, R. M. Murray, A. SangiovanniVincentelli, and S. A. Seshia, Model predictive control with signal temporal logic specifications, in Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on. IEEE, 2014, pp. 8187.

[12] Shalev-Shwartz, S., Shammah, S., & Shashua, A. (2017). On a Formal Model of Safe and Scalable Self-driving Cars, 133.https://doi.org/1708.06374v2

[13] Bagschik, G., Menzel, T. and Maurer, M. (2017). Ontology based Scene Creation for the Development of Automated Vehicles.

[14] Kamlesh Gosh, Automated Planning Based Methods for Early Verification of Reactive Control Systems, Ph. D. dissertation, IIT Kharagpur

[15] Jussi Rintanen. Complexity of planning with partial observability. In ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling, pages 345354.

[16] DeepXplore: Automated Whitebox Testing of Deep Learning Systems Kexin Pei, Yinzhi Cao, Junfeng Yang, Suman Jana

[17] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, DeepRoad:GAN-based Metamorphic Autonomous Driving System Testing, 2018.

[18] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles, IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC, no. i, pp. 14701475, 2016.

[19] Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, Staliro: A tool for temporal logic falsification for hybrid systems. in TACAS, vol. 6605. Springer, 2011, pp. 254257.

[20] J. Deshmukh, X. Jin, J. Kapinski, and O. Maler, Stochastic local search for falsification of hybrid systems, in ATVA. Springer, 2015.

[21] N. Hansen, The CMA evolution strategy: A tutorial, arXivpreprint arXiv:1604.00772, 2016.

[22] B. Shahriari, K. Swersky, Z. Wang, R. Adams and N. de Freitas, Taking the Human Out of the Loop: A Review of Bayesian Optimization, Proceedings of the IEEE, vol. 104, no. 1, pp. 148-175, 2016.

[23] J. V. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu, Testing cyber-physical systems through bayesian optimization, Transactions on Embedded Computing Systems, 2017

[24] S. Ghosh, F. Berkenkamp, G. Ranade, S. Qadeer and A. Kapoor, "Verifying Controllers Against Adversarial Examples with Bayesian Optimization," 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, 2018, pp. 7306-7313. doi: 10.1109/ICRA.2018.8460635

[25] Leveson, Nancy. (2004). A New Accident Model for Engineering Safer Systems. Safety Science. 42. 237-270. 10.1016/S0925-7535(03)00047-X.

[26] Engineering a safer world (2012), NG Leveson, The MIT Press.

[27] Siddartha Khastgir (2018). Systems Approach to Creating Interesting Test Scenarios for Autonomous Vehicles Autonomous Vehicle Test & Development Symposium 2018 Stuttgart, Germany 5 June 2018.

[28] Grunewalder S, Audibert J, Opper M, and Shawe-Taylor J. Regret bounds for Gaussian process bandit problems. In Proc. 13th International Conference on Artificial Intelligence and Statistics (AISTATS 10), pages 273280, Sardinia, Italy, 2010.