Adversarial attack on a deep model of pedestrian detection in CARLA simulator

Ainoa Contreras Rodríguez

Abstract– Driving assistance and autonomous driving use deep perception models to perform tasks such as object detection and classification. Once the models have been trained, it is important to find cases where they can fail, with the aim of including them in subsequent retraining and achieving more robust models. One way to find unforeseen situations is through simulation, thanks to which corner cases can be forced. In this context, this project focuses on pedestrian detection by image using CARLA simulator, based on automatic creation of scenarios in order to validate the performance of the detector. One genetic algorithm automates the search for plausible scenarios designed to cause the detector to fail, thus exposing its weaknesses. This detector validation procedure is considered an adversarial attack method.

Keywords– Deep Learning, autonomous driving, ADAS, genetic algorithm, adversarial attack, scene simulator, pedestrian

Resum– L'assistència a la conducció i la conducció autònoma utilitzen models profunds de percepció per dur a terme tasques com la detecció i classificació d'objectes. Un cop entrenats els models, és important trobar casos en què aquests puguin fallar amb l'objectiu d'incloure'ls en un posterior reentrenament i aconseguir models més robusts. Una manera de trobar situacions no contemplades és a través de la simulació, gràcies a la qual es poden forçar casos extrems. En aquest context, aquest treball s'enfoca en la detecció de vianants per imatge utilitzant el simulador CARLA a partir de la creació automàtica d'escenaris per tal de validar les prestacions del detector. Mitjançant un algorisme genètic s'automatitza la cerca d'escenaris plausibles destinats a fer fallar el detector, mostrant-ne així les debilitats. Aquest procediment de validació del detector es considera un mètode d'atac adversari.

Paraules clau– Deep Learning, conducció autònoma, ADAS, algorisme genètic, atac adversari, simulador d'escenes, vianant

1 INTRODUCTION - MOTIVATION

EVEN though the idea of the autonomous vehicle is not recent, during the last decade autonomous driving and Advanced Driver Assistance Systems (ADAS) [8] have progressed enormously thanks to big advances in *Deep Learning* and in particular, in *neural networks*. ADAS, for instance, can be found in multiple vehicle models on the market and they offer technical support to the driver in tasks such as detecting lane markings, or maintaining a safe distance to the preceding vehicle, among others. However, the inner functioning of the object detectors models that use neural networks developed can be considered as a black box. It may not be not clearly known when they function properly or not, and it is required to ensure consolidation and security mechanisms to guarantee that they operate as expected. It is for this reason that researchers in the sector focus, on the one hand, on innovation, and on the other hand, on the consolidation and strengthening of already developed systems. It is an aspect in which human beings are at risk and the steps must be unyielding.

The use of neural networks allows the automation of processes that previously required a human being, such as object detection or speed control depending on the road or weather conditions. Due to the fact that this automation allows to consider a wide variety of scenarios, manual generation of use cases limits the search.

• E-mail de contacte: ainoa.contreras@autonoma.cat
• Treball tutoritzat per: Antonio López (Department of Computer Science UAB) i João-Vitor Zacchi (Fraunhofer IKS)
• Curs 2022/23

In order to train the neural network models, it is necessary to collect specific data for each task, in the context of autonomous driving and ADAS systems, these are scenarios fin which the main actor is a vehicle traveling. To carry out the data collection there are two options: real data or synthetic data generated in a simulator. Due to the great difficulty of collecting real data, it is essential to use virtual tools for the training and testing phases. The use of a simulator in the early stages allows to reduce the cost and to have critical situations in a simple way, efficiently complementing the conventional tests on the road.

Autonomous vehicles and vehicles that incorporate ADAS use different sensors to interact with the environment where they are located and adapt to situations efficiently. The RGB camera is one of the mandatory sensors and provides a color view of the scene ahead of the vehicle. Thanks to this environment digitalization, the use of model detectors is possible. In this case, we will focus on pedestrian detection. The main task for the system is to provide the required tools for early detection and prevent rather than treat.

In this study, different use cases are explored automatically conditioned to one purpose: failure in detection. The analysis is accomplished through the use of one genetic algorithm that explores different weather parameters and the moment of the day within one use case defined, and creates one scenario in the simulator. When the scenario is generated with the information provided by the algorithm, the objective deep model of pedestrian detection is called for inference. The achieved adversarial attack provides the minimum weather alteration that causes failures on the pedestrian detection model selected and origins a collision between one vehicle and one pedestrian within one use-case previously defined.

The remainder of this paper is organized as follows: Section II and III presents the objectives and planning definition and Section IV provides an overview of related. Section V outlines the details of our proposed diagnosis framework, including the integration of image quality metrics and the analysis of their impact on detection algorithms. Section VI presents the experimental setup and evaluation results. Finally, Section VII concludes the paper by summarizing our contributions and discussing potential avenues for future work in this domain.

2 Objectives

The main objective of this work is to automatically find failure cases of a targeted deep object detection model. We explore a parameterized space of simulated scenarios, searching for adversarial weather conditions to the target model. To this end, we describe the following objectives:

- Parameterization of one use-case.
- Adaptation of one exploratory algorithm, that is objective functions and search space definition.
- Scenario generation within a simulator. The simulator must receive input data from the exploratory algorithm such as the weather parameters and generate a scenario following this information.
- Integration of the target detection model as a pedestrian detector within the simulator.
- Explore realistic weather conditions in order to generate realistic scenarios.
- Compare the results with a benchmark: random generation of parameters.

3 Planning

The organization and the planning of the different tasks such as technical progress or deadlines can be found on Figure 1. The assignments are shown with its intended development time and it follows a linear structure, therefore it is mandatory to complete the first tasks to move forward. Blue color represents the tracking and official deadlines and green color is used for the technical responsibilities.

4 State-of-the-art

In the last few years, rapid advances in deep learning techniques have greatly accelerated the momentum of object detection technology. The object detection algorithms can be divided into one or two-stage models. Within the first group, we can highlight Fast RCNN [5] and Faster RCNN [13] (2015), Mask R-CNN [6] (2017) and G-RCNN [22] (2021), and for one-stage models, the YOLO algorithm has been constantly improving since 2016 (YOLO [10] in 2016, YOLOv3 [11] in 2018, YOLOv4 [1] in 2020, and YOLOv7 [21] and YOLOv8 [12] in 2022), reporting the best accuracy and speed compared to other models found.

Exploratory algorithms [20] have been a frequent resource from ancient ages as well as they are nowadays. Their greatest strength is that they can solve truly diverse problems and adapt the exploration space for the objective functions defined. Some of the state-of-the-art algorithms go from the most simple but effective, such as backtracking, A* or Dijkstra, going through Evolutionary Algorithms (EAs) [20] such as Genetic Algorithm (GA), to more sophisticated methods that have the ability to learn like Reinforcement Learning (RL).

To address problems of high time and infrastructure cost of testing autonomous systems, several simulators have been developed. The most popular are Gazebo [4], CARLA [3], TORCS [23] and AirSim [14]. Modern game engines support creation of realistic urban environments and enable visually realistic simulations, detection of infractions as well as collisions.

As described in the next section, for this study the pedestrian detector model chosen is YOLOv8 [12], the exploratory algorithm is the genetic algorithm and the autonomous driving simulator is CARLA [3].

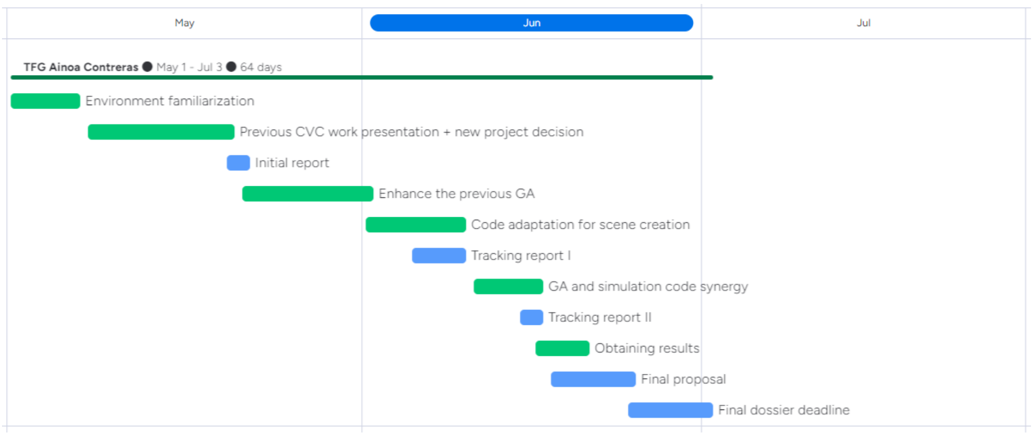


Fig. 1: Gantt diagram of this project.

5 METHODOLOGY

5.1 Adversarial attacks

In the context of computer vision, more specifically object detection, an adversarial attack can be defined as a disturbance that is applied to an objective model in order to obtain an erroneous classification or bounding box. On the one hand, misclassification is considered when the model incorrectly classifies the image when, prior to the disturbance, it did so correctly. On the other hand, the model can also detect the object in a different location of the image, or the object can be partially detected. We can find an example of an adversarial attack adding fog to the input image in Figure 2. After the addition, the model is not able to identify the pedestrian.

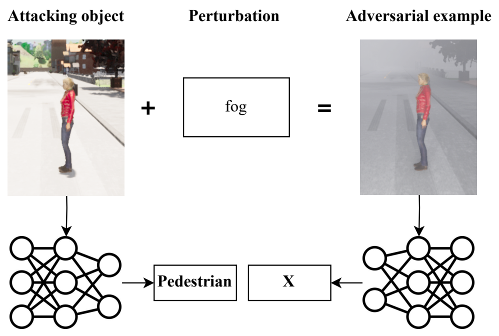


Fig. 2: Example of an adversarial attack where a model is not able to detect the pedestrian after adding fog.

Since the model stages can be divided into training, testing and development, the adversarial attacks can be performed in each of these, following the classification on [9] we can find: attacks in the training, testing and development stages.The training stage adversarial attacks refer to the fact that, in the training stage of the target model, the adversaries carry out attacks by modifying the training dataset, manipulating input features or data labels. Attacks in the testing and development stage refer to change or alter input data with the aim of obtaining different results without modifying the model parameters.

We will focus on the attacks in the testing stage, which can also be divided into two large groups: white-box attacks and black-box attacks. The former have access to the parameters, algorithm and structure of the objective model whereas the latter do not have information about it.

In this project a black-box attack is carried out in the testing stage, and the technique followed is the use of the objective model as an oracle, querying by applying inference in order to obtain the predictions.

5.2 Genetic Algorithms

Genetic algorithms belong to the Evolutionary Algorithms [17] (EAs) group, this means that they use mechanisms inspired by nature and solve problems through processes that emulate the behaviors of living organisms. In a genetic algorithm, a population of candidate solutions (called individuals, organisms, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The symbols that make up the string are called genes. Chromosomes evolve through iterations, called generations. In each generation, chromosomes are evaluated using some measure of fitness, in our case a set of score functions designed for this problem. The following generations (new chromosomes) are generated by applying the genetic operators repeatedly, these being the operators of selection, crossover, mutation and replacement. The genetic algorithm process visualization can be found on Figure 3.

5.2.1 NSGA-II

Within genetic algorithms there are different methods that can be adapted to the characteristics of the problem to be solved. In this case study, the Nondominated Sorting Genetic Algorithm II [2] is the perfect algorithm thanks to the Pareto principle proposed to deal with multi-objectives optimization problems.

The presence of multiple objectives in a problem gives rise to a set of multiple optimal solutions (called Pareto-optimal solutions), instead of a single solution. Neither of these Pareto-optimal solutions are better than

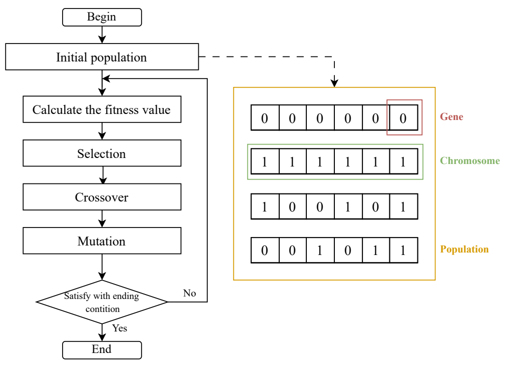


Fig. 3: Flux diagram of the genetic algorithm and gene, chromosome and population visual description.

the other, but each one meets different requirements for each objective function. This fact creates the need to find as many Pareto-optimal solutions as possible.

The NSGA-II algorithm follows the general scheme of a genetic algorithm with a modified selection of mating and survival. The survival method innovates by the fact that the population R_t is the union of parents P_t and childs Q_t (where Q_t is P_t after applying selection, crossover and mutation operators), thus keeping half of the population intact. The mating procedure is as follows:

1. It performs a non-dominated sorting on the current population R_t and classifies by fronts, that is, they are classified according to an ascending level of non-domination. In Figure 4 we can see two different fronts for an optimization problem of 2 functions f_1 and $f2$.

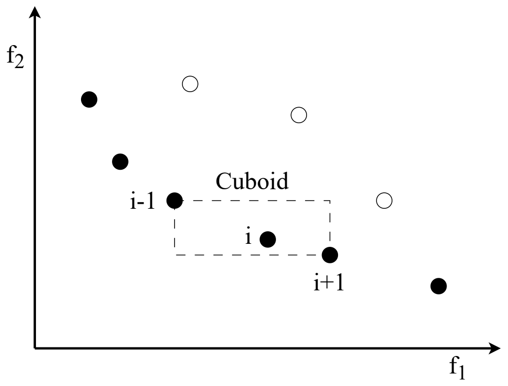


Fig. 4: f_1 and f_2 optimization using NSGA-II. The filled points belong to the first order front F1, and the unfilled points belong to the second order front F2. The dashed box represents the crowding distance cuboid computed for i.

2. Fill in the new population according to the ranking of fronts.

3. If a front is partially used, such as F3 in Figure 5, crowding-sort is performed using the crowding distance that is related to the density of solutions around each solution. The crowding distance of the i-th solution on its front is the average side length of the cuboid (the cuboid is shown as a dashed box on Figure 4) The less dense ones are chosen. Therefore, a representative sample is taken from the set of solutions from the front that we are selecting, trying to choose the samples that are less crowded.

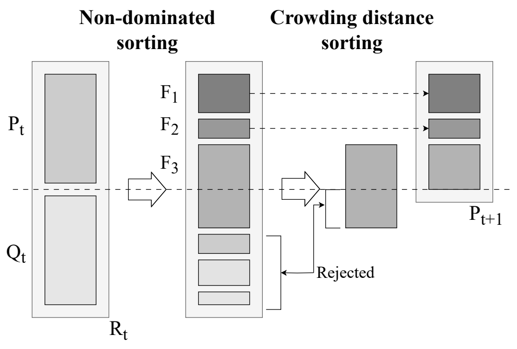


Fig. 5: NSGA II algorithm procedure. The Non-dominated sorting and crowding distance sorting process is shown. The current population R_t consists of P_t and Q_t, both of size N. A group P_{t+1} of size N is chosen based on the non-dominated classification. The ordering is carried out regarding the fronts (F1, F2, F3, ...). In the case of remaining a partially selected front, as is the case of F3, the crowding distance sorting is carried out to select the less crowded individuals. P_{t+1} is given.

4. A new offspring R_{t+1} is created from this new population, on the one hand keeping the entire P_{t+1} individuals, and on the other hand using the crowded tournament selection (it is compared by front rank, if equal, then by crowding distance), crossover and mutation operators on P_{t+1} to lead to Q_{t+1}. The new R_{t+1} is the union of parents P_{t+1} and childs Q_{t+1}, both of size N.

5.3 CARLA simulator

CARLA [3] is an open-source simulator for autonomous driving research. It has been developed to encourage development, training as well as validation of autonomous urban driving systems. In addition to the open-source code and protocols, this simulator provides open digital assets, such as urban layouts, buildings and vehicles.

The simulation platform promotes flexible specification of sensor suites. Carla can be used to study the performance of three approaches to autonomous driving — modular pipeline, an end-to-end model trained via imitation learning, an end-to-end model trained via reinforcement learning (RL). Carla's features include scalability via a server multi-client architecture, autonomous driving sensor suite, flexible API [15], fast simulation for planning and control, maps generation, traffic scenarios simulation, ROS integration, and autonomous driving baselines.

5.4 YOLOv8

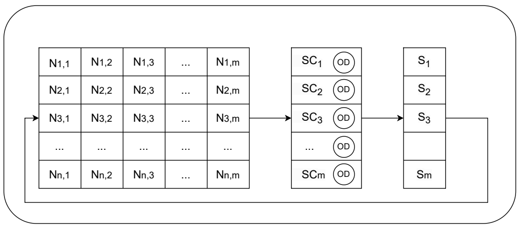
YOLOv8 [12] is the newest state-of-the-art YOLO [10] model that can be used for object detection, image classification, and instance segmentation tasks. The one-stage object detector model is trained with the COCO dataset [7], a 91-labeled dataset including high variability. Thanks to its small model size and easy-to-use CLI implementation, the inference procedure is more intuitive and can be smoothly adapted in the code.

5.5 Adversarial sample generation

Adversarial samples were first named by Szegedy et al. [18], and, as a general rule, are built from generating images - having same features as the used for training - that cause the objective model to classify them incorrectly with high confidence.

In this study, the adversarial samples are generated by the exploration of the weather parameters of the scene in order to find the limitations of the detection model. Following the nomenclature of a genetic algorithm explained in Section 5.2, one chromosome can be defined as one vector that contains a set of parameters that define a scene in the CARLA simulator [3], such as cloudiness or precipitation. Each value is a gene. The dimension *m* of the vector is defined by the number of parameters explored. In each generation, *n* vectors are explored simultaneously, the value known as the population size.

The process starts in the first iteration with the exploration matrix *N* (*Noise*) initialized as the random population. Each row stores the parameter values which enable the scenario *SC* to be generated in the simulator. When the scenario is created, the model inference is called for collecting the detections. The last step is the score *S* calculation for each scenario that will determine the new matrix *N'* according to the GA criteria. We can see in Figure 6 the exploration matrix *N*, the scenario generator vector *SC*, the object detector *OD*, and the score vector *S* as well as the scheme of the exploration process.



$N_{1,1}$ $N_{1,2}$ $N_{1,3}$ ... $N_{1,m}$
$N_{2,1}$ $N_{2,2}$ $N_{2,3}$ ... $N_{2,m}$
$N_{3,1}$ $N_{3,2}$ $N_{3,3}$ ... $N_{3,m}$
... ...
$N_{n,1}$ $N_{n,2}$ $N_{n,3}$ ... $N_{n,m}$
SC_1 OD
SC_2 OD
SC_3 OD
... OD
SC_m OD
S_1
S_2
S_3
S_m

Fig. 6: Diagram of the adversarial samples generation process using the Genetic Algorithm (GA). Each row *N* of the matrix represents a noise vector that allows generating a scenario SC in the simulator that employs the object detector (OD). Subsequently, the output of the scenario generated is applied in order to obtain a score S. The score defines the new matrix *N'* of the next generation following the definition of the GA.

5.5.1 Scenario definition

A single use case is defined: one car is located at one point on the map and drives forward, while a pedestrian is in a static position in the middle of the same road, placed in front of the vehicle.

The vehicle and the pedestrian are the single agents. They are classified as PC ('Player Character') and NPC ('Non-Player Character'), respectively. This means that the main actor is the vehicle and its behaviour is determined within the scenario, different from the control of the pedestrian, that belongs to the scenario definition. The position of these agents in CARLA can be defined as follows.

$$P_{car} = (x, y, z) \quad P_{ped} = (x, y, z)$$

One scenario is defined by a set of parameters such as the map, the initial position of the agents, the weather conditions (rain, clouds, wet ground) or the moment of day. We will divide this set in 3 groups: the basic parameters, the explored parameters and the extra parameters. The basic parameters of the scenario below will be defined by default.

- Map
- Car sensors
- Initial position of the vehicle
- Initial position of the pedestrian
- Initial speed of the vehicle

The parameters that will be explored by means of the genetic algorithm are shown below, delimited with minimum and maximum values.

- Precipitation [0,100]
- Fog density [0,100]
- Wetness [0,100]
- Cloudiness [0,100]
- Sun altitude angle [-90,90]

Precipitation, Fog_density, Wetness and *Cloudiness* refer to the level of rain, the concentration of fog, the intensity of humidity and the cloud level in the sky presented by the CARLA API [15] and include values from 0 to 100, where 0 is the minimum and 100 is the maximum. The variable *Sun_altitude_angle* represents the altitude angle of the sun and includes values from -90 to 90, the first being midnight and the second being noon.

Concerning the selection of the initial position of the agents, we must take into account one restriction: the distance between the vehicle and the pedestrian at the beginning of the simulation must be at least the braking distance, this means that the car has to have enough distance to brake and avoid hitting the passerby.

In order to satisfy the rule described above, the formula for the linear acceleration [19] (Formula (1)) is applied, in which v_1 is the final velocity, v_0 is the initial velocity, a is the acceleration, t is the time taken and s is the distance traveled.

$$v_1^2 = v_0^2 + 2as \quad (1)$$

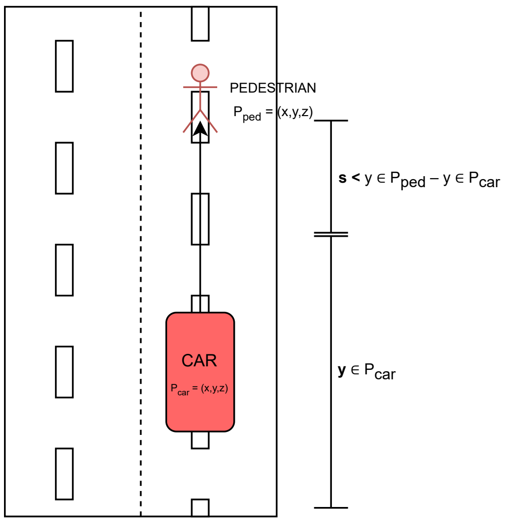


Fig. 7: Diagram of the scenario that represents the study in which we find a 2-lane road. The car is located in the right lane, with coordinates $x, y, z \in P_{car}$ and a straight trajectory in the lane. The variables $x, y, z \in P_{car}$ are immutable. Located in the center of the right lane, in front of the car, the pedestrian stands in a static and immutable position $x, y, z \in P_{ped}$. It must be ensured that, when starting the simulation, the minimum braking distance of the vehicle s between it and the pedestrian is conformed.

The values v_1 and v_0 are easily obtainable, such as a and, based on these, the value s can be found. Therefore, the restriction is defined as follows.

$$s < y \in P_{ped} - y \in P_{car}$$

Once the fixed and variable parameters for the initialization of a scenario in CARLA have been defined, the extra parameters are initialized randomly, since we consider that they are not decisive in the study. This is due to the fact that quotidian scenarios are preferred. The remaining weather parameters are either less common, such as dust storms, or are correlated with the explored parameters, such as precipitation deposits to precipitation.

Regarding the technical specifications, the best configuration for each of the designs is assured. This means that the best sensor configuration, or the best rendering option is guaranteed, among others.

5.5.2 Objective functions

Since it is an exploratory algorithm that minimizes the score, we must define an objective function. In order to generate valuable test scenarios, we must identify scenarios that are more likely to lead to safety violations. In this case, due to the nature of the problem, three objective functions are defined:

- *Journey_distance()* receives the initial and final position of the vehicle and calculates the distance[1] between both points, the objective is to maximize this distance, therefore, to maximize this function.

$$\textit{Journey\_distance} = d(\textit{initial\_Pos}_{car}, \textit{final\_Pos}_{car})$$

- *Total_distance()* receives the number of steps (frames) of the simulation, the position of the vehicle for each step, and the position of the pedestrian for each step. The sum of the distances between them is performed for each step in order to minimize this distance, therefore, the objective is to minimize this function.

$$\textit{Total\_distance} = \sum_{i \in (s,\ldots,steps)} d(Pos_{car}[s], Pos_{ped}[s])$$

- *Scene_alteration()* receives the noise vector and performs the sum of the explored parameters. The amount of alteration of the parameters in one scenario increases by increasing its value. The objective of this function is to minimize the absolute value of these parameters since smaller values of affectation are preferred. Furthermore, day scenarios are preferred over night scenarios.

$$\textit{Scene\_alteration} = \sum_{i \in (1,\ldots,m)} N_i$$

Since we only want to take into account cases where there has been a collision between the vehicle and the pedestrian, the function *Collision()* returns -1000 or 0 depending on whether there has been a collision between the vehicle and the pedestrian or not, respectively, coming from the Collision detector [16] used by CARLA. This value is used to multiply the objective functions described below.

5.5.3 Exploration process

The main part of the project is the exploration of parameters, in Figure 8 we find a flowchart describing the NSGA-II algorithm and the interaction with the CARLA simulator.

First, we initialize the NSGA II evolution parameters such as population size, crossover probability, and stopping condition, among others. The initial random population P is also initialized. Next, for each individual in the population P – it contains a vector with the parameters that condition one scenario with respect to another, such as time, time of the day or the initial position of the vehicle – it is started and launches the simulation in CARLA. Subsequently, we treat the simulation data to obtain one score for each of the 3 objective functions, and we return it to the NSGA II in order to carry out the sorting by fronts, the calculation of the crowding distance and obtaining P.

The first iteration of the algorithm (t=1) starts by applying the selection, crossover and mutation operators on P_t generating Q_t. This is repeated in the CARLA simulator, obtaining a value for each objective function to be minimized by the algorithm. The next step is to apply the non-dominated classification and calculate the crowding distance with the new set R_t (where $R_t = P_t \cup Q_t$) to obtain the new population P_{t+1} and the set of Pareto

[1]Euclidean distance $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

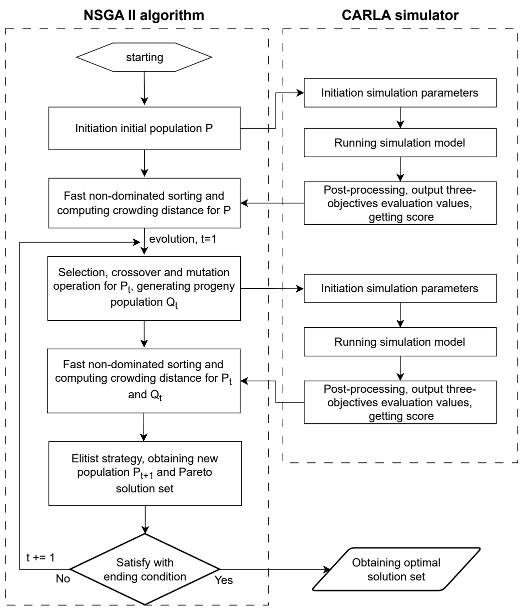

NSGA II algorithm
CARLA simulator
starting
Initiation initial population P
Fast non-dominated sorting and computing crowding distance for P
evolution, t=1
Selection, crossover and mutation operation for P_t, generating progeny population Q_t
Fast non-dominated sorting and computing crowding distance for P_t and Q_t
Elitist strategy, obtaining new population P_{t+1} and Pareto solution set
t += 1
No
Satisfy with ending condition
Yes
Obtaining optimal solution set
Initiation simulation parameters
Running simulation model
Post-processing, output three-objectives evaluation values, getting score

Fig. 8: Flowchart of the NSGA II algorithm combined with the CARLA simulator. The process of the NSGA II algorithm is described until reaching the optimal solution and the interaction that it performs with the simulator for each generation.

solutions.

The last step is to evaluate if the stopping condition is met, in this case the maximum number of iterations. If the evaluation is positive, the set of optimal solutions is obtained, otherwise, the evolution continues.

6 Evaluation

The aim of the adversarial attack is to miss the detection of the pedestrian before reaching the danger zone, this means that the vehicle will have no time to brake before hitting the passerby.

To obtain realistic scenarios, the parameters explored should be common in daily situations. It is clear that in the case of the weather parameters, the alteration level is objective, and the higher the value, the higher the level of alteration. This is not that easy to say regarding the moment of the day, since it can determine the best combination of weather parameters according to different moments of the day.

The first stages of the project did not take into account the scene alteration minimization (*Scene_alteration* function in Section 5.5.2) and, therefore, nights scenarios were preferred. The results shown that light shortage did not help the model to detect the pedestrian and made the final results conditioned by this bias. With the objective of performing a pragmatic parameter exploration and reach unbiased results, day scenarios are preferred. By using the same objective function, weather conditions are minimized.

The number of combinations that one problem with 5 variables (where each one can take integer values between 0

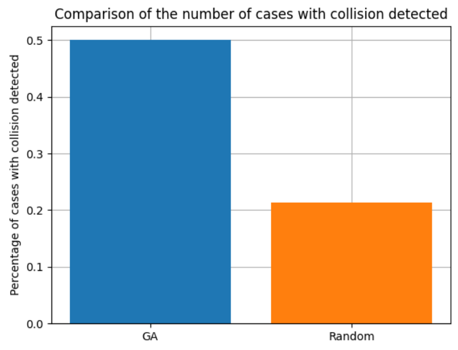


Fig. 9: Bar plot comparing the percentage of total number of collisions detected using GA and random methods.

and 100) is 100^5. This order of magnitude makes the brute-force assessment complex and unreachable.

As a benchmark, the automation of the parameters combination generation is done employing random generators. Applying equal generation hyperparameters - such as population size or number of generations - for both GA and random search, on Figure 9 is shown that the guided search is reaching 50% of cases with collisions facing the 21% obtained running random generation. Following the generation timeline in Figure 10, the GA follows one ascending trend with higher values of collision in almost every generation. The mean number of collisions found per iteration using random values is clearly lower.

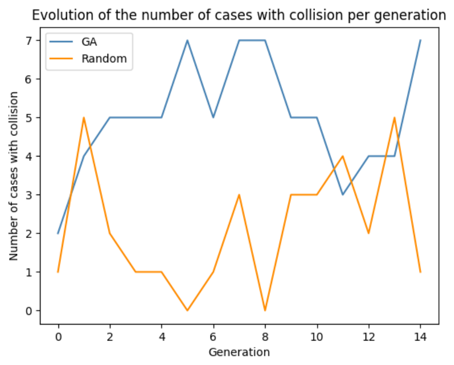


Fig. 10: Line chart describing the evolution of the number of collisions per iteration for each method used.

While the GA benefits from the instructions that provide the objective functions and guide their solutions, the random generation does not have any pattern. This is clearly shown on Figure 11 and Figure 12. If we focus on the results extracted from the guided search in Figure11, some tendencies are found regarding the best results of each iteration. The wetness impact is not significant, it means that higher values of this parameter do not change the final score. Similar situation for the cloudiness, since values fluctuate from 20 to 40 out of 100. This can also be explained because of the fog predominance, since it generates higher altered scenes and specifically modified in the pedestrian location. As said before, darker scenes do not need higher parameter values, as demonstrated in generation 2.

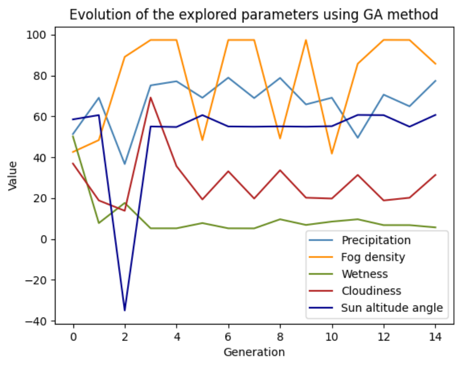


Fig. 11: Evolution of the best combination of parameters regarding the $Scene\_alteration$ function using the GA developed. The evolution is shown through generations.

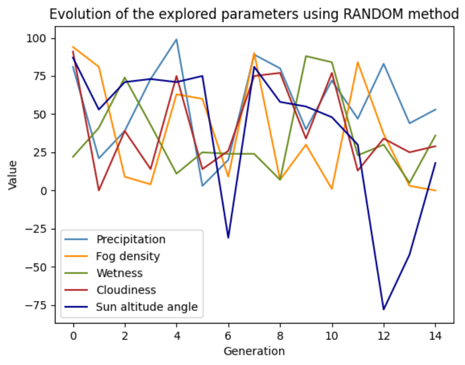


Fig. 12: Evolution of the best combination of parameters regarding the $Scene\_alteration$ function using random generation. The evolution is shown through generations.

The quantity is not as important as the quality and it is also essential to take into account the diversity of the scenarios generated. Due to the fact that the GA keeps the best values and proceeds with a driven search, it makes sense to have less varied samples. Only the combinations of parameters that generated one scenario leading to a collision are selected and the pairwise euclidean distance is calculated among them. We can find on Table 1 that the ranges are 48.9 - 100.6 and 79.1 - 127.5 for the genetic algorithm and random generation techniques respectively. The GA tends to the limit of the constraint, in this case the minimization of the scene alteration, because the collision is a much higher objective. This fact leads to finding fewer but more varied collision cases using the random generation approach. However, by using this technique, the scene alteration minimization and the daily scene boost are not achieved.

On Figure 13, the best configuration regarding the $Scene\_alteration$ function is shown. The best combination of parameters is the following:

- Precipitation = 69.105
- Fog density = 48.378
- Wetness = 7.7714

	GA	Random
Range of distances	48.9 - 100.6	79.1 - 127.5

TABLE 1: Failure diversity, shown as the range in the average pairwise euclidean distance for test cases.

- Cloudiness = 18.843
- Sun altitude angle = 60.6

In this case, the generated scenario is leading to a collision between the pedestrian and the ego vehicle. This is because the detection of the pedestrian starts within the braking zone, so the vehicle has no time to avoid running over the pedestrian.

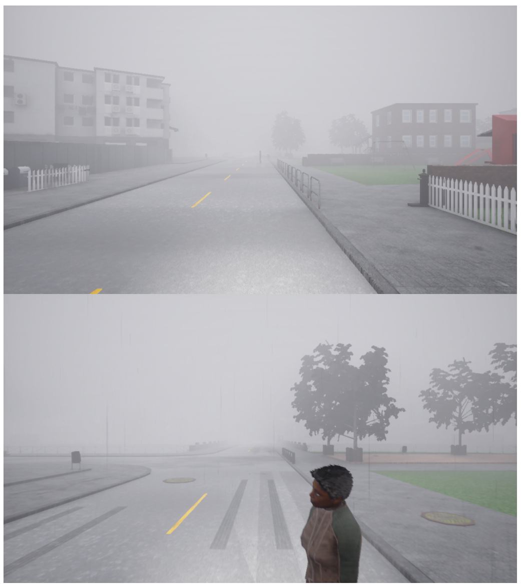


Fig. 13: First and last scene of the best configuration found by the GA regarding the $Scene\_alteration$ function. It shows the collision between the car and pedestrian during a cloudy and foggy day.

6.0.1 Validity

After revealing different cases where the object detector has not been able to detect only by applying one selected configuration of weather parameters and moment of the day, it is demonstrated that there is still a way to improve the state-of-the-art models' performance. On the one hand, in this case the object detector YOLOv8 implements static detection, which involves an inherent lack of information. On the other hand, it is important to consider that the object detector has not been trained with specific data from the CARLA simulator or scenes involving pedestrians. This fact can also explain the results obtained.

Furthermore, identical configurations of the same algorithm may give different results as they are not deterministic. This means that YOLOv8 can provide different detections for the same input.

Another fact to take into account is the random selection of the physical description of the pedestrian for each simulation generated. This can lead to have different detections for same weather parameters and moment of the day. For instance, colorful clothes help the objective model to detect.

7 Conclusions

This work focuses on finding failure cases of onboard object detection task supporting collision avoidance, with emphasis on pedestrians. To carry this out, one use-case has been described, the exploration parameters have been defined as well as the exploration algorithm and the initialization information for the simulator.

The NSGA-II and its interaction with CARLA have been adapted, where the simulator incorporates the YOLOv8 object detector that obtains information from the rgb camera car sensor. Thanks to this configuration, the weather parameters and the moment of the day have been explored in order to find the minimum weather alteration on daily scenes that leads to a collision. This is due to the fact that, the object detector does not detect soon enough the pedestrian and exceeds the braking zone, consequently, even if the vehicle starts the "imminent collision" mode (this means pressing the brake pedal to the maximum) there is no other option than run over the pedestrian.

One realistic set of adversarial samples has been found assuring the problem features described in the objectives (Section 2). For example, on Figure 13, the cloudy and foggy situation is perfectly feasible to find in real life and the system should be aware of this corner case. Even with the high-beam car lights on, the objective model is not able to detect the pedestrian before entering the braking zone. However, this could also be considered as a tricky sample due to the fact that it is also complicated for a human to detect the pedestrian.

Thanks to automatic search algorithms, the driven exploration process is advantageous in relation to brute-force algorithms. The former, due to its searching nature, will follow the same tendency and provide less diverse samples. This fact can also be seen as its strength because of the same reason. It is easy to define one - or multiple - objective functions and achieve the goals by reducing the exploration space.

The next steps shall follow this line of work, that is, enhancing the current state-of-the-art achieved by applying specific modifications concerning the NSGA-II specifications, such as mutation rate, parent selection or crossover methods. Defining a static physical appearance of the pedestrian may also improve the study assessment. In all cases, the final step should be model retraining with the new data generated in order to conclude with one robust model for pedestrian detection.

During this project I have learned to design the solution to a problem from the first to the last stage, as well as define objectives and development deadlines. I have learned to develop a genetic algorithm that has already been implemented, such as NSGA-II, and adapt it to the problem described, furthermore I have developed myself from scratch with an autonomous driving simulator such as CARLA.

Acknowledgement

This project has been supported by the Computer Vision Center on the early stages and Fraunhofer IKS on the last stages, which have provided the necessary resources and data. I would like to thank to Joao-Vitor Zacchi and Núria Mata as well as Antonio López for all the support and knowledge provided week after week.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[3] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[4] O.S.R Foundation. Vehicle simulation in gazebo. https://classic.gazebosim.org/blog/vehicle%20simulation, accessed on June 26, 2023.

[5] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[8] Meng Lu, Kees Wevers, and Rob Van Der Heijden. Technical feasibility of advanced driver assistance systems (adas) for road traffic safety. *Transportation Planning and Technology*, 28(3):167–187, 2005.

[9] Shilin Qiu, Qihe Liu, Shijie Zhou, and Chunjiang Wu. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, 9(5), 2019.

[10] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[11] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[12] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8, 2023.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[14] Dey D. Lovett C. Kapoor A. Shah, S. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and Service Robotics (2017), https://arxiv.org/abs/1705.05065.

[15] CARLA Simulator. Carla python api. carla.readthedocs.io/en/0.9.3/, accessed on June 18, 2023.

[16] CARLA Simulator. Carla python api. sensors reference: Collision detector. carla.readthedocs.io/en/latest/ref_sensors/#collision-detector, accessed on June 18, 2023.

[17] Andrew N. Sloss and Steven Gustafson. 2019 evolutionary algorithms review, 2019.

[18] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 2014.

[19] Physics Forulas toppr. Linear acceleration formula. www.toppr.com/guides/physics-formulas/linear-acceleration-formula/, accessed on June 18, 2023.

[20] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3), jul 2013.

[21] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022.

[22] Jianfeng Wang and Xiaolin Hu. Convolutional neural networks with gated recurrent connections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[23] Espié E. Guionneau C. Dimitrakakis C. Coulom R. Sumner A. Wymann, B. Torcs, the open racing car simulator. https://torcs.sourceforge.net/, accessed on June 26, 2023.