Towards End-to-End Chase in Urban Autonomous Driving using Reinforcement Learning

Michał Kołomański[1], Mustafa Sakhai[1], Jakub Nowak[1], and Maciej Wielgosz[1,2]

[1] AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, al. Adama Mickiewicza 30, 30-059 Cracow, Poland

[2] ACC Cyfronet AGH, Nawojki 11, 30-950 Cracow, Poland

michalk@agh.edu.pl, msakhai@agh.edu.pl, jakubn@student.agh.edu.pl, wielgosz@agh.edu.pl

Abstract. This paper addresses the challenging task of developing an autonomous chase protocol. First, training of an autonomous vehicle capable of driving autonomously from point A to B was developed to proceed with a chase protocol as a second step. A dedicated driving setup, based on a discrete action space and a single RGB camera, was developed through a series of experiments. A dedicated curriculum learning agenda allowed to train the model capable of performing all fundamental road maneuvers. Several reward functions were proposed, which enabled effective training of the agent. In the subsequent experiments, we selected the reward function and model that produced the most significant outcome, guaranteeing that the chasing car was within 25 meters of a runaway car for 63% of the episode duration. To the best of our knowledge, this work is the first one that addressed the task of the chase in urban driving using the Reinforcement Learning approach.

Keywords: autonomous vehicle; reinforcement learning; autonomous chase

1 Introduction

In 2018, the percentage of the population living in urban areas was over 55%, and it is projected that by 2030 it will exceed 60% [1, Table III.1, p. 55]. Sustainable urbanization requires that, among other essential factors, cities provide the necessary infrastructure for transportation and preserve a healthy environment within the city and surrounding areas [1, p. 2]. Additionally, the policies to manage the urban growth need to consider and adjust to the needs of various groups such as youth, elderly people, and individuals with disabilities [1, p. 2].

Autonomous vehicles (AVs) are seen as a significant disruptor in the transportation industry, with the ability to alter travel habits and change the environment significantly [2]. Vehicle security, traffic congestion, and travel behavior may all be affected by this new technology. Overall, significant social AV impacts in the form of crash savings, travel time reduction, fuel efficiency, and lowering the number of parking spaces needed are projected to be in the range of $2000 to $4000 a year per AV, with extensive crash costs accounting for nearly $4000 [3]. Moreover, AVs can enhance the quality and productivity of time spent in vehicles, improve the transportation system's safety and reliability, and transform transportation into a service open to everyone at any time [4].

The rest of the paper is structured as follows. Section 2.1 section provides a brief review of the prior works related AV and chase. The next section 2.2 presents an architecture of the proposed model along with all the steps which were taken in order to train it. The results of the conducted experiements are presented in section 3.1 Finally, the paper short discussion and conclusion are given in sections 4.

1.1 Autonomous Vehicle

We use the following definition of an autonomous car in this research: *An autonomous car is a self-driving vehicle capable of perceiving unpredictable, uncertain, and open traffic environment and navigate autonomously through this environment without human intervention* [5].

However, the above definition represents the pinnacle of autonomy and, therefore, the perfect future end-state of AVs. Given the current absence of genuinely autonomous vehicles, classifying the different degrees of autonomy is necessary.

The Society of Automotive Engineers (SAE) defines six Levels of Driving Automation (LoDA)[6]. The United States Department of Transportation has already adopted these standards. [7, 8]. Instead of autonomous, the SAE uses the word automatic. One explanation is that the term "autonomy" has broader connotations than those associated with electromechanical systems. Furthermore, the terms self-driving and autonomous are often used interchangeably. However, while a self-driving car can operate autonomously in specific, if not all, circumstances, a human driver must still be present and prepared to take control. The authors of [6], described the SAE's levels of driving automation (LoDA); starting from level '0' No Driving Automation, level '2' Partial Driving Automation, to level '5' where Full Automation is applied for the vehicle to perform all driving tasks under all conditions.

Developing a fully autonomous, economically viable vehicle is a complex undertaking, as an AV may encounter an infinite number of situations in the real world. An AV must demonstrate that its system can successfully navigate typical scenarios and edge cases – rare occurrences in which operational parameters reach extreme values or never seen circumstances occur. Regrettably, proper behavior in such scenarios is extremely difficult, and today's autonomous vehicles are not always capable of dealing with them. Exposing a model to many comparable scenarios with the same context is one method to improve its performance in edge case circumstances. However, getting similar edge case scenarios in the real world is very difficult and will never exhaust all possible scenarios. As a result, having a massive amount of data from both the real world and simulation, which allows for the simulation of a significant number of road occurrences, is critical. AVs would need to be driven billions of kilometers in real-world and simulated scenarios to establish safety for all road users.

1.2 CARLA

CARLA (Car Learning to Act) is an open-source research simulator for autonomous driving. Essentially, it simulates metropolitan traffic, including vehicle traffic. This environment was created from scratch to provide a modular and adaptable API for developing, prototyping, training, and validating autonomous urban driving systems, including perception and control [9, 10]. It uses the OpenDRIVE standard (version 1.4) to define roads and the urban environment. Control of the simulation is provided by a Python and C++ API [10]. Many city layouts, a range of car models, buildings, pedestrians, sensors, traffic signs, and traffic lights are provided by CARLA environment [9].

CARLA simulator is built on a client-server architecture. The server is in charge of the simulation, including physics calculation, sensor rendering, updating actors, and world state. The client, identified by an IP address and a port number, is used for requesting information or changes in the simulation using CARLA API. Multiple clients can be active at the same time [10, 11].

2 Materials and Methods

2.1 Related work

During the literature review, only one scientific publication was found that deals with a very similar problem (chase done by autonomous vehicle). Other relevant works involve utilizing Reinforcement Learning (RL) to learn the driving task in general. A car chase is when one vehicle follows another vehicle from point A to point B as quickly as possible, usually using a similar but not identical route. Chase is associated with high speeds. As a result, it involves complex maneuvers and puts an autonomous driving system's capabilities to the limits. In the autonomous car chasing scenario, the pursued vehicle can be operated by a human or controlled by an automated system. In comparison, the pursuing car is entirely operated by an autonomous system.

Although there are some papers about car chases that mostly described by the authors of [12], there are numerous ones about car following. Models of car-following explain the mechanisms by which cars follow one another in a traffic flow [13]. A car-following model regulates the vehicle's actions concerning the vehicle ahead in the same lane. A vehicle is classified as following when its actions and movement are constrained by a preceding vehicle, and attempting to drive at the desired speed will result in a crash. The actions of the follower are primarily specified through the followed car's acceleration, speed, or the distance between two vehicles.

Some car-following models define a vehicle's actions only when directly following another vehicle. In contrast, others are more comprehensive and describe the vehicle's behavior under all circumstances. Finally, a car-following model should be capable of determining the regime or state in which a vehicle is, and the appropriate actions to take in each state [14].

While car following and car chasing have a lot in common, they also have a number of characteristics that set them apart. The primary distinction between these two notions is that during car following, the vehicle in the front does not attempt to flee. This implies that the vehicle in front generally does not make unexpected manoeuvres while adhering to the road's laws. The another distinction between a car chase and a car following scenario is speed, which is normally much higher in a vehicle chase scenario. Furthermore, the car chase scenario assumes the possibility of breaking traffic laws while assuming no significant negative consequences for other road users. The scenario of chasing another car can be considered as a major extension of the scenario of following another car. Moreover, chase can be perceived as following with a strong engagement imperative. We assume that chase starts when vehicles are close enough and the chasing one is determined to maintain the requested distance.

This paper focuses on working on a chase scenario using reinforcement learning. The ultimate goal is to come up with a pair of agents (chasing and escaping) which can operate in extreme conditions (at very high speeds) which a proxy to real-life situations in which current AV technology has the biggest challenges. In other words, the most demanding situations in driving occur quite rarely but usually develop in very short time.

Autonomous Chase The authors of [12] provided a partial solution to the problem of developing an autonomous car capable of pursuing another vehicle using just images from a single RGB camera. According to their research, there was no scientific activity in that domain until 2020, making them the forerunners of this topic. Their solution is based on a dual-task convolutional neural network that concurrently identifies the pursued vehicle's 2D bounding box and predicts coarse semantic segmentation that detects a drivable surface [12]. The authors divided the challenge of estimating the distance to a chased car into two scenarios. The first is when the pursuing vehicle is evident in the image, in which case a PnP (Perspective-n-Point) problem solver was used [15, 12].

The authors first gathered a dataset of 20 individual and manual travels throughout the CARLA's city by a fleeing car to conduct the experiments. This publicly available dataset has ten simple vehicle chase scenarios and ten harder ones, with rides lasting approximately one minute on average. Several versions of the dataset were prepared, which allowed scaling experiments from easy to hard ones depending on the complexity of a dataset.

On a simple dataset, all three algorithms proposed in [12] worked well. With a complex dataset, we can see that the full version of the algorithm (with semantic segmentation and extrapolation of a distance and an angle) performed far better than others. However, it was still only 63% accurate in terms of average completion rate [12]. Semantic segmentation was found to be crucial in increasingly complex scenarios, particularly those including turns. Detection of the road, buildings, and sidewalks helped the pursuing car chase for an extended period. Authors of [12] were the first to address the issue of an autonomous car chase. **The authors did not employ reinforcement learning techniques; instead, they took a more algorithmic approach.** The obtained results were satisfactory, particularly in cases involving simple car chases. However, results from more complex scenarios indicate that there is considerable space for improvement.

CARLA: An Open Urban Driving Simulator In [9], the authors investigated three approaches to autonomous driving utilizing CARLA to demonstrate the potential of the offered environment. The initial approach used a modular pipeline (MP) with distinct visual perception (semantic segmentation), planning (a rule-based state machine for enforcing simple predetermined police in metropolitan settings), and a control subsystem (a PID controller that controls the steering throttle and braking). The second and third strategies, respectively, were based on a deep network that had been trained end-to-end utilizing conditional imitation learning (IL; the use of high-level commands) and reinforcement learning (the asynchronous advantage actor-critic (A3C) algorithm).

The dataset used for the model training was acquired during random rides. It is composed of photos captured by a forward-facing camera. The authors added noise into the data collection process to boost the robustness of the learned policies. The dataset is utilized for training a deep network capable of predicting the expert's action a given an observation o and a control command c. Around 14 hours of driving data were gathered for training purposes. Four progressively more tough driving tasks were utilized in six distinct weather conditions. The agent is permitted to disregard speed restrictions and traffic lights in these experiments. Town 1 is utilized for training purposes, while Town 2 is used for testing purposes.

Testing was conducted over a 25-episode period for each combination of a task, a town, and a weather set. Each episode's objective was to reach a specified destination. An episode was considered successful if the agent accomplished the objective within the time limit. None of the approaches performs flawlessly, even when doing the most straightforward driving in a straight line. Moreover, RL performed significantly worse than IL even though RL was trained on a more considerable amount of data (12 days) compared to 14 hours by IL. However, the authors admitted that no data augmentation or regularization, such as dropout, was used to train the RL model.

CIRL: Controllable Imitative Reinforcement Learning for Vision-based Self-driving The authors of [16] reported an upgraded version of the deep reinforcement learning pipeline for vision-based autonomous driving. Their approach outperforms prior Modular Pipeline (MP), Imitation Learning (IL), and Reinforcement Learning (RL) methodologies on various driving tasks using the CARLA benchmark from [9]. The authors concluded that it is impossible to obtain ideal results with classical RL or IL on complex real-world problems.

RL is prone to be very time-consuming and easy to fall into local optimum after many episodes. IL cannot be extrapolated to unknown scenarios, and the human driving data coverage strongly constrains their performance. To address these issues, the authors introduced a unique Controllable Imitative Reinforcement Learning (CIRL) technique that enables continuous controllable deep reinforcement learning by leveraging knowledge gained from human expert demonstrations. This strategy is founded on the concept of the Deep Deterministic Policy Gradient (DDPG) that is an off-policy replay-memory-based actor-critic algorithm [16]. Due to an excessive number of failed explorations for an ample action space, the standard DDPG frequently falls into the local optimal. The proposed CIRL addresses this issue by offering improved exploration seeds for searching the actor networks' action space.

Since the core algorithms used in our work on this paper employ Reinforcement Learning, we decided to include a section which settles this work in a context of the broadly-known framework formalism.

Reinforcement Learning The field of machine learning, known as Reinforcement Learning (RL) , is concerned with sequential decision-making. The concept of RL is based on the idea of an agent performing actions in an unpredictable environment to maximize cumulative rewards. Unlike the dynamic programming approach, the RL agent does not require complete knowledge nor control of the environment. It just needs to be capable of interacting with its environment and gathering data [17]. The learner is not instructed which actions to take; instead, it must experiment to determine which ones produce the greatest reward. Additionally, actions have the potential to influence not just the immediate reward but also all future rewards. The two main defining characteristics of reinforcement learning are trial-and-error experience and delayed reward [18].

In Figure 1, an agent and environment interaction are shown. It can be described in the terms of the Markov Decision Process (MDP).

An MDP is a discrete-time stochastic control process which indicates that the system is controlled and its states are fully observed. An MDP is a 5-tuple (S, A, T, R, γ) where [17]:

$$S \;\text{ is the state space,} \tag{1}$$

$$A \;\text{ is the action space,} \tag{2}$$

$$T: \; S \times A \times S \rightarrow [0,1] \;\; is\ a\ state\ transition\ probability\ function, \tag{3}$$

$$R: \; S \times A \times S \rightarrow \mathbb{R} \text{ is the reward function,} \tag{4}$$

$$\gamma \in [0,1] \text{ is the discount factor.} \tag{5}$$

In the timestamp t, the environment is described by the state S_t. The agent reads the current state S_t and uses its policy $\pi(a_t|s_t)$ to determine which action perform based on the state S_t. This is referred to as A_t. An agent operates in accordance with its policy. A policy is a function $\pi : S \rightarrow A$ which takes a state s as an input and returns an action a which should be performed by a model

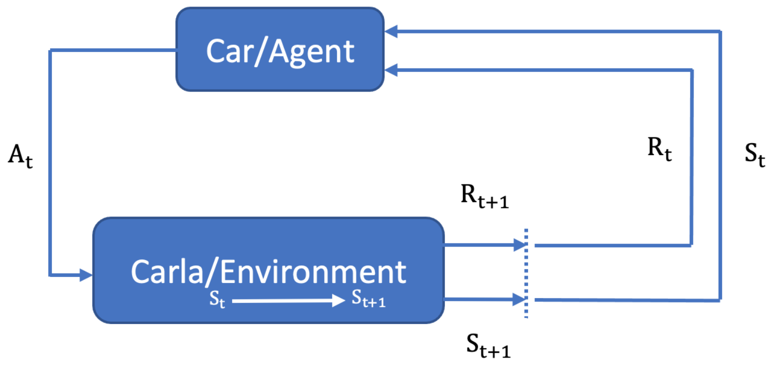


Fig. 1: Agent-environment interaction in RL.

in state s. The probability of moving to S_{t+1} is given by the state transition function $T(s_t, a_t, s_{t+1})$. The action gets transmitted to the environment, which results in obtaining the subsequent timestamp $t+1$ and receiving a reward R_{t+1} from a reward function [17].

$$R(s_t, a_t, s_{t+1}) \in \mathbb{R} \tag{6}$$

2.2 Experimental setup

The CARLA in version 0.9.10 has been successfully configured in such a way that efficient autonomous vehicle training was possible. To present and validate the methodology, we ran two setups. Initially, we developed a model capable of autonomously traveling from point A to point B using separate convolutional neural networks (CNNs). Next, we developed a model capable of autonomous car chasing by establishing a conducive environment for car chase learning. The used code is publicly available[3]. The selected model was controlled by the well-known Advantage Actor Critic (A2C) algorithm in all performed experiments. A2C is a policy gradient algorithm that is a member of the family of on-policy algorithms [19].

Action space A vehicle (Tesla Model 3) may be operated by transmitting desired steer, throttle, brake, handbrake, and reverse (gear) signals [20].

We used the possible car control commands described in [20] to calculate an approximation of the size of the action space. Such action space is called **continuous**. The manual_gear_shift parameter should be omitted since an AV must automatically change gears without human intervention. Furthermore, throttle, steer, and brake values are defined using a single-precision floating-point. For

[3]https://github.com/Michal-Kolomanski/Autonomous-driving-in-Carla

throttle and brake parameters, there are $126 * 2^{23}$ distinct unique values that lie between 0 and 1. There are twice as many possible unique values for the steering parameter since it ranges between -1 and 1. Since a single control message contains a set of values for each command, the possible number of different actions is equal to their product. It is approximately equal to [20]:

$$\underbrace{(252 * 2^{23})}_{\text{Steering}} * \underbrace{(126 * 2^{23})}_{\text{Throttle}} * \underbrace{(126 * 2^{23})}_{\text{Brake}} * \underbrace{2}_{\text{Hand\_brake}} * \underbrace{2}_{\text{Reverse}} \approx 5.63055788 * 10^{20} \quad (7)$$

This action space is vast even for a deep learning agent. Fortunately, its size can be reduced to some extent. In this work parameters: hand_brake and reverse were always set to false to reduce the action space significantly. Because those actions are seldom performed in the real world, the experiments were designed so that those actions were unnecessary. As a consequence, an autonomous vehicle was controlled by the throttle, steer, and brake commands. Moreover, drivers do not accelerate and brake simultaneously, so the throttle and brake parameters can be expressed using only one parameter. The values between 0 and 1 represent braking, as well as acceleration. However, this operation does not reduce the action space. Regarding the steering parameter, values between -1 and 0 indicate left turn, whereas values between 0 and 1 indicate a right turn. As a result, three parameters are used to control the movement of the vehicle in CARLA.

The discrete action space was also introduced to reduce the size of the action space even more. Discretizing the action space is done by reducing the possible values of the control commands. The most commonly used values of control commands should be chosen that allow reaching any place. At first, action space proposed by [20] was used but their training process was not effective. Therefore, we decided to modify the action space and leave only those actions which were essential to successfully complete the experiments, to get faster satisfactory results. Also, steering was changed from 1 to 0.5 in all turns. This resulted in the action space shown in Table 1.

Table 1: Discrete actions used in this work.

Action index	**Action description**	**Action array value [acceleration, brake, steer]**
0	Forward	[1, 0, 0]
1	Forward and left	[1, 0, -0.5]
2	Forward and right	[1, 0, 0.5]
3	Brake	[0, 1, 0]
4	Brake and left	[0, 1, -0.5]
5	Brake and right	[0, 1, 0.5]

While the presented actions ensure that each point may be reached, it may not be exact. Furthermore, the car will drive abruptly and unpredictably, which instills little confidence in humans. Nevertheless, the discrete space significantly reduces the size of the action set. As a result, finding the optimal solution to problems that do not require perfect precision can be faster than utilizing continuous action space.

In the initial experiments, the agent took actions determined by the network in each simulation frame. Given the simulation's 30-60 frames per second frame rate, the agent performed 30-60 actions per second. It was discovered that by performing so many actions per second, the agent could not obtain accurate feedback from the environment, as each action takes time to manifest its effects. The answer to this issue was to lower the number of actions performed per second to five, which proved to be a critical breakthrough in our research. The agent was able to learn well as a result of this adjustment. It is also worth emphasizing that, with five actions per second, the agent has a considerable measure of flexibility when it comes to adjusting the steering angle, acceleration, and braking values. Although the values for the turns, acceleration and braking were predefined, the agent controlling algorithm was capable of adjusting these values by performing various actions in the proper order. The best way to illustrate this is through an example. Let us assume that the algorithm controlling our model needed to perform a right turn with a lesser angle than predefined in right turn action; the algorithm might then perform a certain number of right turns and fewer left turns. As a result, the algorithm was able to control the steering angle of a turn. Learning how the algorithm should exploit this mechanism appropriately was a key component of the training and it proved to be an efficient method of vehicle control.

Model architecture Separate convolutional neural networks were used for the actor and critic. The images of size 80×80 were utilized as an input for the networks. The networks were identical in each experiment. For continuous action space, a Gaussian distribution-based policy representation was utilized. The actor's CNN generated the mean and sigma. A categorical distribution was used in discrete action space. In tables 2 and 3, the architecture of CNN, the actor and critic are shown respectively. Table 2 contains a section of the architecture that may be considered a backbone, whereas Table 3 covers a head of the model. The head architecture is presented for both discrete and continuous action spaces.

We determined that the gamma and learning rate values producing the best performance are $gamma = 0.9$ and $lr = 1e - 4$. Furthermore, it was determined to apply entropy regularization to encourage exploration and avoid being stuck in a local optimum.

Since there were no additional road objects in the city during the experiments, the networks did not need to be more sophisticated to produce satisfactory results. Additionally, given the available computing power, these networks enabled relatively rapid model learning.

Table 2: The shared architecture of CNN (backbone) which was utilized for actor and critic.

Layer	**Parameters**
Conv2d	in_channels=3, out_channels=32, kernel_size=8
Conv2d	in_channels=32, out_channels=64, kernel_size=3
Conv2d	in_channels=64, out_channels=64, kernel_size=3
Linear	in_features=64 * 7 * 7, out_features=512

Table 3: Differences between the actor and critic CNNs. Different head architectures

		Layer Parameters
Actor	Discrete action space	Linear in_features=512, out_features=9
	Continuous action space	Linear in_features=512, out_features=2
Critic		Linear in_features=512, out_features=1

Routes for autonomous driving from point A to point B Even in a simulator, the challenge of developing an autonomous vehicle model capable of autonomously chasing is significant. As a consequence, prior to experimenting with autonomous chase, we decided to conduct some experiments on a slightly simpler problem - autonomous driving from point A to point B It was chosen to base the training of the AV model on the concept of curriculum learning. The idea of curriculum learning comes from teaching people. It is an effective way to learn from elementary concepts to more complex challenges progressively. It simplifies complicated knowledge by providing a succession of learning steps with increasing complexity [21].

As a consequence, a few critical driving scenarios with growing difficulty were picked to train the model. Each scenario has a unique route between a spawn point and a terminal point. Both the route (drawn as blue line) and the terminal point are visible, which is crucial from an actor and critic's standpoint. Furthermore, in some scenarios, the distance between the spawn and terminal points was too large to produce practical model training, necessitating the introduction of middle points, which are also visible on the route. The middle points are positioned at critical locations on the route to aid the model's navigation throughout the town:

– At the start and end of turns,
– Along the route, according to the user-specified parameter *mp_density*, which specifies how dense the middle points should be.

Ultimately, five distinct scenarios were used to train the models, including two distinct right and left turns and a straight road segment.

Reward for autonomous driving from point A to point B A reward is a critical component of any reinforcement learning algorithm. The reward in this paper is composed of several components:

– **Route distance**: The distance between the vehicle and the route. If this distance is less than 2, the reward is equal to 2; otherwise, the reward is:

$$The\ reward\ = -3 * (route\_distance - 2). \tag{8}$$

– **Speed**: Vehicle's speed. The reward is:

$$The\ reward\ = \frac{speed}{10-2}. \tag{9}$$

– **Terminal point reward**: The static reward for reaching the terminal point is equal to 500.
– **Middle point reward**: The static reward for reaching the middle point is equal to 100.
– **Collision**: The static reward from colliding with other environmental objects is equivalent to -500.
– **Lane invasion**: The static reward from lane invasion is equal to -50

The reward is calculated by summing the elements mentioned above. Individual element values were empirically determined in such a way that the model improved over time. The reward formula was identical in all experiments.

Autonomous Car Chase To prepare for the chase task, we first manually controlled the runaway car. Each manual ride resulted in a file containing the precise coordinates of the car's location and rotation in each simulation frame. Thus, the ride can be recreated by appropriately loading the file. Figure 2c illustrates the scenarios used to train the models. The scenarios were dispersed over the map to make the model's task more challenging and enforce the generalization of the model's knowledge. Curriculum learning was utilized in such a way as to teach the model basic road maneuvers such as driving straight, turning left and right during a pursuit. The speed of the escaping car at the starting point was always zero.

Depending on the scenario, the escaping car accelerated to approximately 30km/h. The runaway car's average speed differed throughout the entire route due to maneuvers performed in each scenario. The chasing car, controlled by the learning network, always started from a point about 5 meters in a straight line behind the escaping car, also at zero speed. Figure 2 presents the right turn scenario in detail. The route taken by the escaping car is indicated on the map by a red line visible from the pursuing car's perspective. The escaping car is also red, which should help the model perform better by linking red with the possibility of getting a larger reward. The camera settings are chosen so that the runaway vehicle is visible in 80×80 images used as input for a learning network.

It was decided to train the models in a variety of scenarios distributed around the city, but with predefined spawn points for the runaway and chasing cars.

In this way, we enabled the generalization of the model without making the model's task prohibitively difficult in terms of computational power and training time. It's worth mentioning that the vehicle was equipped with only a single camera for both the autonomous driving from A to B and the autonomous chase. Naturally, this is the the most affordable choice in reality, which translates directly into the comparatively low cost of our solution. However, it does have implications. Due to the fact that just one camera is used for navigation, the algorithm operating the vehicle does not have complete knowledge about its surroundings, particularly the environment that is not facing. As a result, our system may have difficulty detecting the pursuing vehicle which is not facing frontally.

Furthermore, it should be highlighted that our simplified version of the pursuit is the most prevalent one in reality. According to the real-life scenarios (e.g. police videos) we studied on police chases, we concluded that police pursuits typically begin when the police are roughly behind the fleeing vehicle. When the police are not immediately next to the fleeing vehicle (e.g., in another part of the city), they have to first reach a certain location before pursuing the fleeing car. We believe that this problem should be seen as an autonomous drive from A to B, rather than as part of an autonomous chase. As a result, we believe that our simplification of the pursuit may support different applications and effectively address the challenges posed by reality.

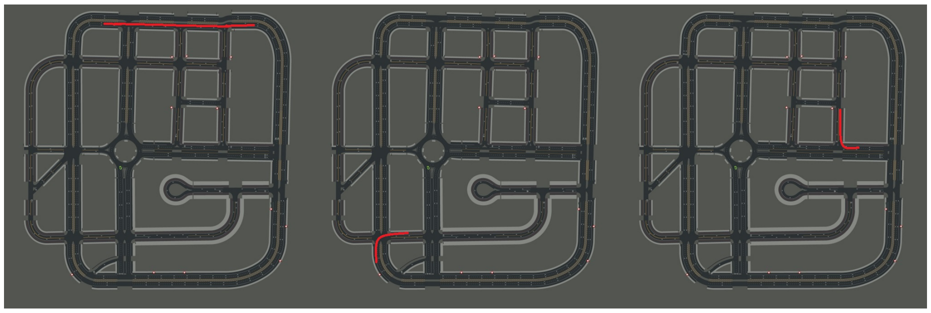
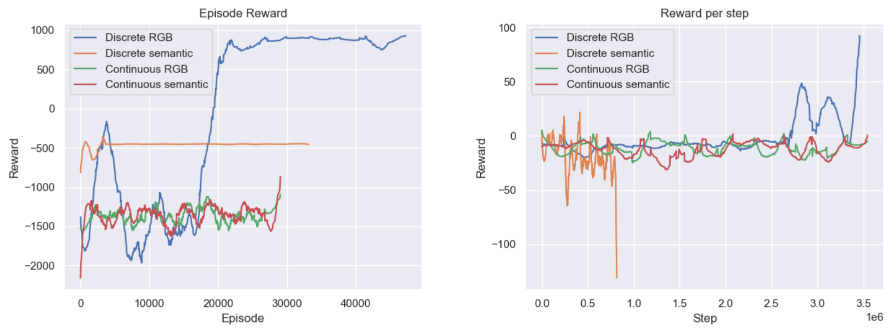

(a) (b) (c)

Fig. 2: Some of the scenarios on which the models were trained. **(a)** Straight line scenario. **(b)** Left turn scenario. **(c)** Right turn scenario.

3 Results

3.1 Autonomous Driving From Point A to B

Many trials have been conducted to determine which combination of parameters produces the most favorable outcomes. The tests were designed to determine which action space (discrete or continuous) and what type of camera sensor (RGB or semantic segmentation) produces superior results relatively quickly. Each experiment consisted of at least ten thousand episodes. Following that, only the experiment with the most beneficial results was continued. Figure 3 depicts the outcomes of the experiments.



(a) (b)

Fig. 3: The results of the experiments. **(a)** Episode reward. **(b)** Reward per step.

The experiments yielded unambiguous findings. A discrete action space combined with an RGB camera outperformed the alternatives. Each time, this model reached its designated location on the map, resolving the scenario's dilemma. However, it is worth considering why this agent's performance dropped drastically after about 5,000 thousand episodes. The agent initially attempted to perform similar maneuvers in each scenario, which resulted in the agent receiving a favorable reward on occasion and a significantly negative reward frequently. This policy led to an episode reward of roughly zero on average. Nevertheless, after 5,000 episodes, the agent started expanding its exploration of the action space in each scenario, resulting in a sudden and temporary decline in performance. However, it was beneficial from the long-term perspective because the agent learned to perform the appropriate maneuvers in each scenario after about 20,000 episodes.

Each experiment lasted a minimum of around 30,000 episodes. The point of around 5000 episodes was critical. At that time, it became clear how much the action space affects the agents' performance. While continuous action space agents performed at a somewhat consistent low level, the discrete RGB agent initially performed impressively before collapsing in performance.

It was also a decisive time for the agent utilizing a combination of discrete action space and semantic camera. It became stuck in a local maximum after approximately 4000 episodes and ceased exploring the action space. As a result, its network consistently returned the same action, resulting in straight driving. In this particular situation, it was discovered that the network used was not well-suited to the combination of discrete action space and semantic camera and contained an excessive number of layers for that purpose. We postulate that a simpler neural network may provide superior performance for this combination since the input class space in semantic segmentation is much smaller. However, it may have generalization difficulties in more complex problems in future scenarios. Furthermore, a segmentation model employed in the pipeline may have a strong impact on the quality of the overall performance. In a chase scenario an agent needs to keep track of the same instance of the object, in this case an escaping car. This however requires instance segmentation and such a flow may be sensitive to occlusions which lead to distraction and potentially unlocking from the tracking.

It is worth mentioning that continuous action space models were more computationally intensive and took significantly longer to simulate the same amount of episodes as other models.

After determining the optimal parameters combination, the model was trained in a considerably more difficult scenario involving two right turns and three straight road sections. Figure 4 depicts the following training scenario for the model that utilized discrete action space and an RGB camera.

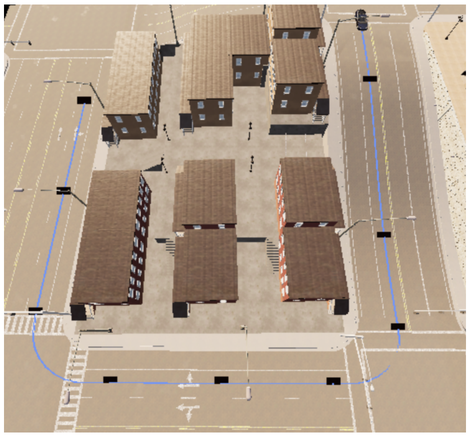


Fig. 4: The scenario on which discrete, RGB model was trained.

The results of the experiments are presented in 5 figure.

After around 15,000 episodes, the model was almost always able to arrive at the scenario's end destination. However, the model continued to explore the action space because it could still beat the scenario faster, resulting in a higher reward. Attempts to improve the solution were unsuccessful, and the model occasionally turned incorrectly or collided with a building due to excessive speed. Nevertheless, the model managed to improve the initial solution and follow the scenario faster in the end. While the trained model did not guarantee that it

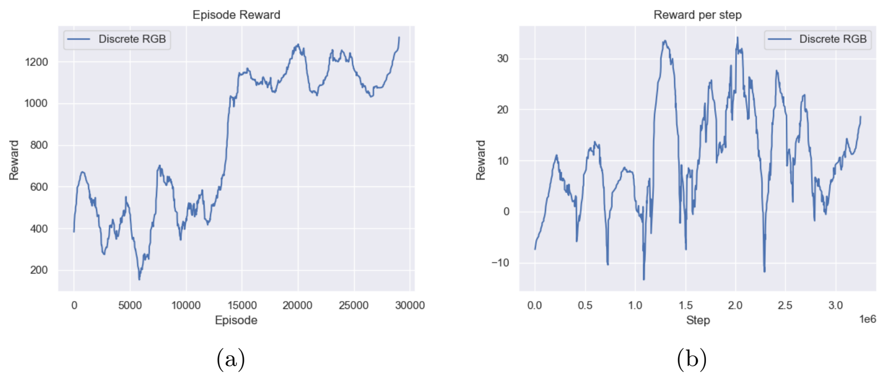


Fig. 5: The results of the experiment. **(a)** Episode reward. **(b)** Reward per step.

would always follow the scenario perfectly, it did demonstrate the potential to develop into such a model with more training.

3.2 Autonomous Car Chase

After establishing an environment conducive to autonomous chase training, it was time to conduct experiments. The constant parameters that were utilized in all experiments are; Discrete Action Space, RGB for Camera, frequency of the agent's actions 5 per second, $80x80$ as input image size, 0.9 Gamma, $1e-4$ for learning rate, entropy regularization set to True, 0.9.10 is the version used in Carla, and Carla quality level set to Low.

The experiments in this section utilized the network models introduced in section 2.2. Initially, the goal was to determine whether the developed model could outperform models trained from scratch in 'chasing' performance. A new reward function was created, and a test was conducted in which one model learned to chase the car from scratch while the other model was starting from the checkpoint obtained from a model trained in the setup described in 2.2. Equations (10 - 15) the reward function was straightforward.

$$\textbf{if } 5 <= \text{distance } <= 25 \rightarrow \text{Distance between vehicles} \tag{10}$$

$$\mathrm{r}_{\text{distance}} = 1 \rightarrow \text{Distance reward} \tag{11}$$

else:

$$\text{distance reward } = 0 \tag{12}$$

$$\textbf{if } \text{len(collision history list)} \mathrel{!}= 0 :\rightarrow \text{If there was a collision} \tag{13}$$

$$\text{col\_reward} = -1 \rightarrow \textit{Collision reward} \tag{14}$$

else:

$$\text{col\_reward} = 0 \tag{15}$$

This reward was largely determined by the distance between the pursuing and escaping vehicles. It is worth mentioning that because the locations of the vehicles were specified relative to their midsections, the distance was calculated as the distance between both cars' centers. As such, 5 meters is one of the shortest distances that cars can reach without causing a collision. The reward was distributed during each step of the episode. The episode was interrupted when the chase finished, the distance between vehicles exceeded 60 meters, or a pursuing car collided with any object. The reward was a sum of *distance_reward* and *col_reward*. Figure 6 presents the results of the experiment.

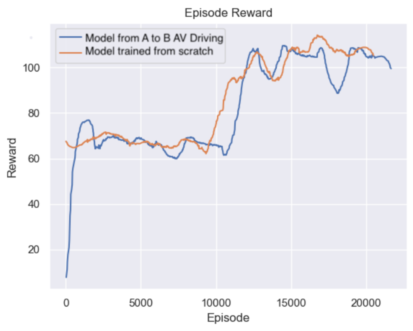


Fig. 6: The comparison of the obtained episode rewards by the models.

The models trained from scratch were able to pursue the fleeing car efficiently for around 57% of the duration of the episodes. The pursuing car was able to stay between 5 and 25 meters from a runaway car through effective chasing.

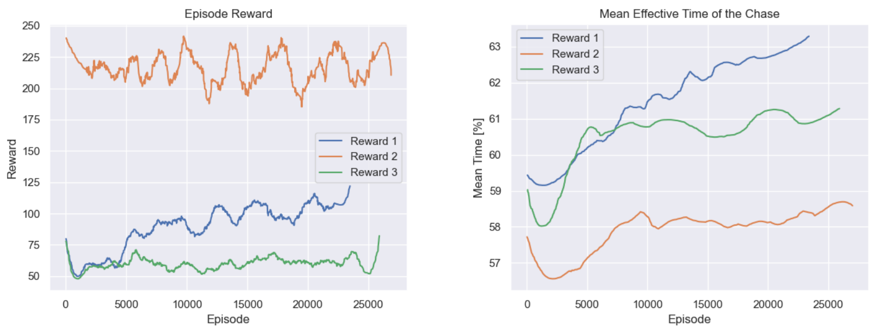
Another experiment that was carried out was to compare several created reward functions and select the one that provided the best agent performance along with the progressive learning process. The created reward functions were inspired by the reward function created in the section 2.2.

In the initial setup, when the distance between the chasing car and the escaping car was between 25m and 59m, it was very rarely profitable for the chasing car to take steps to reduce the distance and extend the time of the episode. It was because then the chasing car received negative rewards for a longer period. Since we intended for the model to try to catch up with the runaway car, the episode's duration was included as a factor that positively influences the final reward. Moreover, in chasing scenarios that are several minutes long and very dynamic, effective chasing time should be one of the main factors considered when comparing models. Table 4 presents reward functions that have been created.

Table 4: The reward functions.

	Reward components		
	Distance	**Duration**	**Collision**
Reward 1	`if 5 <= distance <= 25:` `distance_reward = 1` `else:` `distance_reward = 0`	$1/14 *$ `time`	-1
Reward 2	`distance_reward` $= -1/6 *$ `distance` $+ 4$	$1/3 *$ `time`	-10
Reward 3	`distance_reward` $= -19/500 *$ `distance` $+ 1$	$1/12 *$ `time`	-1

The reward at each step was the sum of all components. Figure 7 depicts the experiment's outcome.



(a) (b)

Fig. 7: The reward functions comparison. **(a)** Episode reward. **(b)** Mean, effective time of the chase.

The first reward function produced the best results, accounting for over 63% of the effective chase of the fleeing car. When comparing this result to the experiment utilizing only distance reward, there is a 6% improvement in agent performance at approximately the same training time.

It is worth noting that while the first reward function was by far the simplest among the ones in Table 4, it has consistently outperformed the more complicated reward functions. The second reward function was found to produce the poorest results throughout the training period. For a time, the third reward function produced promising agent performance; however, the episode reward started to oscillate, eventually limiting the agent's performance.

4 Conclusions

To the best of our knowledge this is the first work which addresses the Chase task with RL framework. Even in a simulator, developing an autonomous vehicle that does a particular task flawlessly is extremely tough. The process of autonomous training cars involves the selection of numerous parameters, including the cameras utilized, the training algorithm, the reward function, the number of actions per second, the learning rate, and the discount factor. All of this expands the realm of possibility for experimentation on this subject. As a result, it is essential to run numerous experiments and fine-tune parameter values to improve the agent's final performance. Additionally, the method is computationally intensive. As a consequence, achieving satisfactory outcomes may take months.

Despite obtaining satisfactory results, the project needs further development. Models training should include scenarios that provide random routes and more road objects throughout the city. Developing a system for recognizing the pursued vehicle among other cars on the road is also necessary for robust autonomous chase models. Furthermore, we intend to improve our vision system by employing more cameras (e.g., eight) around the vehicle and training a model capable of merging pictures from all cameras and delivering complete knowledge about the environment around the car. This system will directly enable the prospect of effective autonomous chase training, independent of the position of the pursuing or fleeing vehicle. It is also worth noting that the challenge of the chase can be formulated as an adversarial setup together with escape in a similar fashion as presented in [22]. The concept of establishing two networks, one of which controls a fleeing car and the other of which controls a chasing vehicle, both of which must maximize the expected sum of their future rewards, appears extremely promising and fascinating, given the inextricably linked difficulties of fleeing and chasing. Therefore, another future initiative will be to conduct an experiment leveraging adversarial machine learning methods.

5 Acknowledgement

We would like to thank the AGH Institute of Electronics which financially supported publishing of this paper with subvention 16.16.230.434

Bibliography

[1] United Nations, Department of Economic and Social Affairs, Population Division (2019): World Urbanization Prospects: The 2018 Revision (ST/ESA/SER.A/420). United Nations (2019), https://population.un.org/wup/Publications/Files/WUP2018-Report.pdf

[2] Rojas-Rueda, D., Nieuwenhuijsen, M.J., Khreis, H., Frumkin, H.: Autonomous vehicles and public health. Annual review of public health 41, 329–345 (2020)

[3] Fagnant, D.J., Kockelman, K.: Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. Transportation Research Part A: Policy and Practice 77, 167–181 (2015)

[4] Schwarting, W., Alonso-Mora, J., Rus, D.: Planning and decision-making for autonomous vehicles. Annual Review of Control, Robotics, and Autonomous Systems (2018)

[5] Jo, K., Kim, J., Kim, D., Jang, C., Sunwoo, M.: Development of autonomous car—part i: Distributed system architecture and development process. IEEE Transactions on Industrial Electronics 61(12), 7131–7140 (2014)

[6] SAE International: J3016 apr2021. taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles (Apr 2021), https://www.sae.org/standards/content/j3016_202104/

[7] Inagaki, T., Sheridan, T.B.: A critique of the sae conditional driving automation definition, and analyses of options for improvement. Cognition, technology & work 21(4), 569–578 (2019)

[8] Synopsys, Inc.: The 6 levels of vehicle autonomy explained, https://www.synopsys.com/automotive/autonomous-driving-levels.html

[9] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: An open urban driving simulator. In: Conference on robot learning. pp. 1–16. PMLR (2017)

[10] Introduction - carla simulator, https://carla.readthedocs.io/en/0.9.11/start_introduction/

[11] Core concepts - carla simulator, https://carla.readthedocs.io/en/0.9.11/core_concepts/

[12] Jahoda, P., Cech, J., Matas, J.: Autonomous car chasing. In: European Conference on Computer Vision. pp. 337–352. Springer (2020)

[13] Brackstone, M., McDonald, M.: Car-following: a historical review. Transportation Research Part F: Traffic Psychology and Behaviour 2(4), 181–196 (1999)

[14] Olstam, J.J., Tapani, A.: Comparison of Car-following models, vol. 960. Swedish National Road and Transport Research Institute Linköping (2004)

[15] Moreno-Noguer, F., Lepetit, V., Fua, P.: Accurate non-iterative o(n) solution to the pnp problem. In: 2007 IEEE 11th International Conference on Computer Vision. pp. 1–8. IEEE (2007)

[16] Liang, X., Wang, T., Yang, L., Xing, E.: Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 584–599 (2018)
[17] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning (2018)
[18] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
[19] Chu, T., Wang, J., Codecà, L., Li, Z.: Multi-agent deep reinforcement learning for large-scale traffic signal control. IEEE Transactions on Intelligent Transportation Systems 21(3), 1086–1095 (2019)
[20] Palanisamy, P.: Hands-On Intelligent Agents with OpenAI Gym: Your Guide to Developing AI Agents Using Deep Reinforcement Learning. Packt Publishing (2018)
[21] Weng, L.: Curriculum for reinforcement learning (Jan 2020), https://lilianweng.github.io/lil-log/2020/01/29/curriculum-for-reinforcement-learning.html
[22] Wu, Q., Ruan, T., Zhou, F., Huang, Y., Xu, F., Zhao, S., Liu, Y., Huang, X.: A unified cognitive learning framework for adapting to dynamic environment and tasks (2021)