

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

LEARNING TO DRIVE IN A SIMULATED ENVIRONMENT
USING DEEP REINFORCEMENT LEARNING

MUSTAFA CEMİL GÜNEY
A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

GEBZE

2022

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**LEARNING TO DRIVE IN A SIMULATED
ENVIRONMENT USING DEEP
REINFORCEMENT LEARNING**

MUSTAFA CEMİL GÜNEY
**A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING**

THESIS SUPERVISOR
ASSIST. PROF. DR. YAKUP GENÇ

GEBZE
2022

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERİN PEKİŞTİRMELİ ÖĞRENME
KULLANARAK SİMÜLE EDİLMİŞ BİR
ORTAMDA ARAÇ KULLANMAYI ÖĞRENMEK

MUSTAFA CEMİL GÜNEY
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
DR. ÖĞR. ÜYESİ YAKUP GENÇ

GEBZE
2022

GEBZE TEKNİK ÜNİVERSİTESİ	YÜKSEK LİSANS JÜRİ ONAY FORMU
----------------------------------	--------------------------------------

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 07/02/2022 tarih ve 2022/08 sayılı kararıyla oluşturulan jüri tarafından 19/02/2022 tarihinde tez savunma sınavı yapılan Mustafa Cemil Güney'in tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE

(TEZ DANIŞMANI) : Dr. Öğr. Üyesi Yakup GENÇ

ÜYE

: Prof. Dr. Yusuf Sinan AKGÜL

ÜYE

: Prof. Dr. Behçet Uğur TÖREYİN

ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun
...../...../..... tarih ve/..... sayılı kararı.

İMZA/MÜHÜR

SUMMARY

Self-driving vehicles can be a disruptive technology and learning to drive with Deep Reinforcement Learning is still challenging and unsolved. End-to-end reinforcement learning methods recently have yielded successful models in video games and specific self-driving tasks. These methods primarily work on simulation environments, and they heavily rely on well-defined reward functions for capturing intended driving policy and mainly use heuristic optimization methods (such as stochastic gradient descent).

This thesis builds a 2D driving environment appropriate to Reinforcement Learning algorithms while providing observations, applying actions, and giving rewards with its defined reward function. A generally used Deep Q-Networks algorithm trained on it successfully to learn a driving policy. However, it cannot move without collision in all scenarios, and it is also moving erratically. Although simple reward functions work very well for static environments, wrongly designed reward functions can lead to unsafe and unwanted driving policies. The optimization methods require a lot of data and time to converge.

In this thesis, the research is enhanced by extending human-in-the-loop reinforcement learning to learn a driving policy. Instead of designing a reward function, one can capture it from actual drivers. As an extension, the smooth feedback mechanism is proposed to capture scenarios such as, instead of making a sharp turn, going for a gradual one. Furthermore, scenes repeated until the agent gained positive rewards, allowing the system to overcome the problematic states. Finally, explanation-based learning is applied by letting the user drive initially, and demonstrations used to be positive and negative feedback. The resulting method applied to the driving simulation environment yields successfully trained models. The method converged faster, and learned driving policy lasts longer without errors with smoother and safer movements than prior Reinforcement Learning and human-in-the-loop works.

Keywords: Reinforcement Learning, Autonomous Driving, Convolutional Neural Networks, Human Reinforcement Learning.

ÖZET

Kendi kendini sürebilen araçlar yıkıcı bir teknoloji olabilir ve Derin Pekiştirmeli Öğrenme ile araç kullanmayı öğrenmek hala zorlu ve çözülmemiş bir problemdir. Uçtan uca pekiştirmeli öğrenme yöntemleri, son zamanlarda video oyunlarında ve belirli kendi kendine sürüş görevlerinde başarılı olduğu gösterilmiştir. Bu yöntemler çoğunlukla simülasyon ortamlarında veya oyunlarda çalışır ve amaçlanan sürüş politikasını yakalamak için iyi tanımlanmış ödül işlevlerine ve ayrıca çoğunlukla kullanılan sezgisel optimizasyon yöntemlerine (rasgele gradyan inişi gibi) büyük ölçüde güvenirlir.

Bu tezde, daha karmaşık bir 3 boyutlu sürüş simülasyonunun ara yüzü olan 2 boyutlu bir sürüş ortamı inşa edildi. Bu sürüş ortamı pekiştirmeli öğrenme algoritmalarının ihtiyaç duyduğu ödül fonksiyonuna sahiptir ve gözlemler sunar. Sıkça kullanılan Derin Q-Ağları algoritması bu ortamda sürüş politikası öğrenmek için başarılı bir şekilde eğitildi. Ancak, elde edilen sonuçlara göre öğrenilen sürüş politikası için çok sayıda deney gerekirken, tüm senaryoları çarpmadan bitirememekte ve düz bir şekilde hareket edememektedir.

Pekiştirmeli öğrenmede optimal bir politikaya yakınsamak için çok fazla veri ve zaman gerekir. Bu yöntemlerin ihtiyaç duyduğu basit ödül fonksiyonları güvensiz ve istenmeyen sürüş politikalarına yol açabilir. Pekiştirmeli öğrenmede sonuçlar her zaman kolayca tekrarlanabilir değildir. Kapsayıcı bir ödül fonksiyonu tasarlamak yerine, gerçek sürücülerin yargılarından ödül fonksiyonu öğrenmeye çalışılabilir. Bu tezde, literatürdeki ödül fonksiyonu yakınsaması yöntemleri bir sürüş simülasyonunda çalışabilecek şekilde genişletildi. Bu yöntemler insanlardan toplanan geribildirimlerin yumuşatılması, insan sürüş örneklerinin kullanılması ve bozuk senaryoların tekrar edilmesi gibi özetlenebilir. Sonuç olarak elde edilen sürüş politikası diğer yöntemlere göre daha hızlı yakınsamıştır. Daha düzgün ve güvenli hareketlerle daha uzun süreler çarpmadan sürüş yapabilmektedir.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Otonom Sürüş, Evrimsel Sinir Ağları (ESA), İnsan Pekiştirmeli Öğrenme.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my primary supervisor, Assist. Prof. Yakup Genç, for his support, guidance, and encouragement throughout this thesis. I am grateful to my friends and family for their love and support. I would also like to thank my friend Emrah Korkmaz for his undying friendship.



TABLE of CONTENTS

	<u>Page</u>
SUMMARY	v
ÖZET	vi
ACKNOWLEDGEMENTS	vii
TABLE of CONTENTS	viii
LIST of ABBREVIATIONS and ACRONYMS	ix
LIST of FIGURES	x
LIST of TABLES	xi
1. INTRODUCTION	12
2. BACKGROUND	16
2.1. Driving Simulations and Datasets	16
2.2. Learning to Drive & Reinforcement Learning	18
2.3. Human-in-the-Loop Reinforcement Learning	20
3. DRIVING SIMULATION ENVIRONMENT	22
4. LEARNING to DRIVE USING REINFORCEMENT LEARNING	26
4.1. Method	26
4.2. Training Details	27
4.3. Results	28
4.4. Conclusion	29
5. CAPTURING REWARD FUNCTIONS FROM HUMANS	31
5.1. Preliminaries	33
5.2. Method	34
5.3. Technical Details	39
5.4. Results and Discussions	41
5.5. Summary	45
6. CONCLUSION AND FUTURE WORK	46
REFERENCES	48
BIOGRAPHY	54
APPENDICES	55

LIST of ABBREVIATIONS and ACRONYMS

<u>Abbreviations and Acronyms</u>	<u>Explanations</u>
2D	: 2 Dimensional
3D	: 3 Dimensional
AI	: Artificial Intelligence
API	: Application Programming Interface
CNN	: Convolutional Neural Network
DL	: Deep Learning
DRL	: Deep Reinforcement Learning
DQN	: Deep Q Network
GPS	: Global Positioning System
LIDAR	: Light Detection and Ranging
LSTM	: Long Short-Term Memory
MAE	: Mean Absolute Error
MSE	: Mean Squared Error
ML	: Machine Learning
ReLU	: Rectified Linear Unit
RL	: Reinforcement Learning
RNN	: Recurrent Neural Network
RGB	: Red, Green, Blue
SGD	: Stochastic Gradient Descent

LIST of FIGURES

<u>Figure No:</u>	<u>Page</u>
3.1: The default map of the environment chosen from the internal maps of CARLA.	22
3.2: Birds eye view of the car with RGB camera (left) and it's semantic segmentation (right).	23
3.3: Important areas of the roads.	24
4.1: The flow diagram of the DQN training loop.	28
4.2: The average episode duration in evaluation phase of the agent (left) and the training loss per episode graph (right).	29
5.1: The general flow of the capturing reward function method.	32
5.2: The Human Feedback Predictor model.	35
5.3: The Autoencoder architecture extract features from high dimensional semantic segmented images.	35
5.4: The algorithm of policy updating and environment loop.	36
5.5: The soft feedback collection algorithm.	37
5.6: The comparison between direct feedback and soft feedback mechanism for a sample trajectory.	38
5.7: The complete design of the human-in-the-loop driving policy learning system.	40
5.8: The human feedback collection user interface.	41
5.9: The average steps size per episode of the agents trained with different policy methods.	42
5.10: Total feedback counts in training time.	43
5.11: The average steps size per episode of the agents trained with different policy methods in different scenarios.	44

LIST of TABLES

<u>Table No:</u>	<u>Page</u>
5.1: The comparison of the average Euclidean distance from the center of the road and the smoothness score of the models and humans.	44



1. INTRODUCTION

Self-driving cars or autonomous cars are Artificial Intelligence (AI) agent vehicles that use cameras, radars, and other complex systems to establish driving strategies for traveling between destinations. For a self-driving vehicle to be fully autonomous, it must be capable of navigation without a human driver. There is a rapid progress in self-driving vehicle technology, mainly backed by advances in deep learning. According to prestigious committees, autonomous Driving is classified into five levels, from no automation to full automation [1].

- Level One where the driver is assisted with warnings and cruise-control. Driving does not differ from current non-autonomous systems, but devices can take control in minor ways if needed.
- Level Two, system can take full control, but still needs the driver is in charge.
- Level Three is where the car is driving autonomously, but the driver is ready to take control when needed. The system can identify such emergency cases and alert the driver.
- Level Four is defined as high automation; car drives in a fully automatic way, but the driver can intervene whenever he wants.
- Level Five is the fully autonomous level. A driver is not needed, and no human has any role in the driving task.

Autonomous cars are decision-making systems that process streams of observations coming from different sensors such as cameras, radars, LIDAR, ultrasonic sensors, and GPS units. The autonomous system uses these observations to determine its actions. LIDAR, the acronym for Light Detection and Ranging (sometimes Light Imaging, Detection, And Ranging), measures distance by disturbing a target with a laser and was created as a combination of light and radar. Radar is an object detection system that uses radio waves to determine various objects' range, angle, or velocity. Ultrasonic sensors are also object detection systems that emit ultrasonic sound waves and detect their return to determine distance. Finally, visual systems (often cameras) use sophisticated object detection algorithms (deep learning) to understand what is visible.

Learning to drive is still a challenging and under-explored task, fed with high space inputs. Driving requires the capacity to solve complex problems such as lane tracking, localization, drivable path recognition, and steering control, among others. Building this kind of system requires enormous amounts of data and experimentation. Simulations can also close the data and experimentation gap. Deep reinforcement learning methods have been applied to various autonomous driving tasks to teach machines to control cars in simulated environments.

This thesis involves comprehensive research on building a system that learns to drive in a simulated environment in an end-to-end manner using Deep Reinforcement Learning techniques. We examined ways of involving humans in the learning loop to improve the driving system and built our own environment on top of the frequently used driving simulation CARLA [2]. We kept our newly built environment simple, focusing on the algorithms instead of the problems unrelated to driving. The environment is two-dimensional, though the simulation and dynamics are realistic. It could be said that we built an environment with a 2D interface but a 3D system. The environment has been implemented in GYM [3] like API, meaning universal Deep Reinforcement Learning algorithms can be applied with minimal effort. The environment has its own rules to manage collisions or crashes, and it has a reward function for Reinforcement Learning based algorithms to train on. It can also provide RGB images of the state and semantically segmented versions.

The reinforcement learning algorithm applied to this task is the often-used method Deep Q-Networks (DQN) [4]. It has been proven that policies learned with DQN can outperform humans in specific ATARI games tasks. Because the simplified environment is like a simple game and works with discrete actions, we preferred to work with DQN. The algorithm assumes that the neural networks are nonlinear function approximators, and therefore, learning parameters of Q function would be possible. The Q function is the discounted sum of rewards of an agent at a time. For this purpose, we built a CNN architecture that takes semantically segmented images as inputs and maps them into Q values for each action.

It has been shown that a neural network that ends up with an acceptable driving policy over a sufficient period of time can be successfully trained. It can go straight and turn left and right without collisions, or with minimal collisions and without any crashes. However, it cannot successfully turn junctions. While it can go straight, the agent exhibits slalom behavior instead of going forward. Moreover, it requires

extensive data and long experiments. Since safety is paramount in self-driving, it cannot be said that it is safe, as it exhibits non-smooth movements, and its reward function cannot identify all safety cases. Hence, the scope of this thesis was expanded, with additional research into ways of including humans in the RL to overcome these problems.

Humans are experts in driving systems because everything has been built to simplify the action, including the control of the car. They can drive safely without experiencing crashes, learning from other humans for the most part. Even an individual without a driving license can understand if someone is driving safely or not. So, we based our study on the assumption that humans have a hidden reward function that judges driving. We used humans to capture the reward function, which we accomplished by collecting feedback with this assumption in mind. There are different ways of collecting feedback, such as natural language directions or EEG signals. Feedback can also be collected by demonstrations, comparing behaviors, giving directions, or giving binary feedback. We prefer to collect feedback in binary, as positive and negative. However, we also use demonstrations and collective approaches, building a feedback mechanism that can be used efficiently.

In this thesis, the work related to capturing reward functions can be considered an extension of previous studies, including Deep TAMER [5]. As in previous studies, we used a CNN autoencoder to capture information of high dimensional input, and we had a replay memory that kept feedback and observations in a buffer to fit human feedback function, sampling from it. The human feedback was in binary form (positive and negative) and continuously collected. The policy to drive was added on top of the estimated human reward function. We improved this with human evidence, adding human behavior directly into the loop to give the model an advantage, pretraining it. In addition, we used a soft feedback mechanism to reduce delays in feedback collection and enable a smooth trajectory. A replay mechanism also forced unsuccessful scenarios to repeat, collecting more feedback.

The proposed method was evaluated by episode duration of different tasks compared to Deep TAMER and DQN. We observed that the novel method could converge faster than others and reach human-level performance on most of the tasks under 200 episodes. The smoothness and safeness of the policies are also measured in the purpose of usefulness. The proposed method is better than others and close to human performance in smoothness and safety.

The thesis is organized as follows. Section 2 provides background on driving simulations, datasets, Reinforcement Learning in self-driving, and humans in the Reinforcement Learning loop. Section 3 explains the simple driving environment built for the RL tasks. Section 4 describes how DQN learned a driving policy in the developed environment, expressed through evaluations. Section 5 extensively discusses capturing reward functions from humans and using them efficiently. Then the experiments and results are reported with comparisons. Finally, Section 6 presents the conclusions and plans for future studies.



2. BACKGROUND

This section will examine prior studies related to learning to drive in a simulated environment using Deep Reinforcement Learning. First, we will review driving simulations and datasets to boost learning techniques. Then we will go through what Deep Reinforcement Learning studies have in common with learning a driving policy. Finally, we will expand our research into the addition of humans into loop Reinforcement Learning, human preference learning, human reward shaping, and inverse reinforcement learning.

2.1. Driving Simulations and Datasets

Driving simulations are handy tools both for research and in the field. These simulations help develop and evaluate driving systems without using expensive cars, sensors, or tools. Also, simulations help with many datasets and interactive environments necessary to test safely autonomous driving algorithms. The simulations differ in development purposes, graphic engine, realistic-unrealistic features, traffic rules, action capabilities, and tools (camera, LIDAR, etc.).

Some researchers prefer to develop small simulation environments [6], but others use pre-developed and open-source frameworks. Lately, the most commonly used simulation has been CARLA [2]. CARLA has been developed from the ground up to support the development, training, and validation of autonomous urban driving systems. In addition to open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) created for this purpose and can be used freely. The simulation platform supports flexible specifications of sensor suites and environmental conditions.

Autoware [7] is the first all-in-one open-source software for autonomous driving technologies. It is based on ROS 1 and is available under an Apache 2.0 license. It contains the following modules: localization, detection, prediction, and planning. Also, the company LG has a division of this platform, named LGSVL [8].

Apollo [9] is yet another autonomous driving simulation platform developed by Baidu. They created a dataset called The Apollo Scape [10], which provides annotated street view images for semantic segmentation, lane detection, car detection, and pedestrians. The dataset helps evaluate the performance of methods with different

weather conditions and different day times. There are also lots of similar commercialized or open-sourced driving simulations, such as Nvidia Driveworks [11], OpenPilot [12], and SUMO [13].

Some studies also preferred to work with robot simulators [14]. Robotic simulations can simulate the mechanical system dynamics, making it possible to train and test AI algorithms and design robots. Gazebo [15] is an example of a robotic simulation that includes car dynamics and traffic simulations. Webots [16] is another example of robotic simulations that include car simulation. Webots also provides a tool to import maps from OpenStreetMap. Even though robotics simulations are not as realistic as simulations dedicated to driving, they are still valuable tools in their simplicity and dynamics that closely resemble the real world.

Some researchers also used games or game-like simulations to develop and evaluate their methods. For example, the open racing car simulator (TORCS [17]) is a modern, modular, highly portable multi-player, multi-agent car racing simulation. It is used as an ordinary car racing game, an AI racing game, and a research platform. Some researchers used Grand Theft Auto V [18]. The game provides basic traffic conditions and day-night conditions of the environment.

Datasets are crucial for researchers and developers, and they have played a vital role in the progress of research by providing testing before publishing. There are various datasets on the autonomous cars field. Datasets can be from the real world or sensors. Also, they can include sensor outputs and images from different driving scenarios.

The PASCAL VOC dataset [19] was one of the first datasets to include relevant classes for autonomous cars. KITTI Vision Benchmark [20] provided many labeled driving scenes in 2012. This dataset is most commonly used for driving automation. UC Berkeley DeepDrive [21] is a recent dataset with annotated image data. The Oxford RobotCar [22] collects over 1000 km of driving data with six cameras, LIDAR, GPS in the UK. However, the dataset is not annotated. Cityscapes [23] is commonly used for computer vision algorithms as a benchmark set. The nuScenes dataset is the most recent urban driving dataset with LIDAR and image sensors [24].

2.2. Learning to Drive & Reinforcement Learning

A deep reinforcement learning agent learns the optimum way of driving by earning rewards from its environment. The definition of reward function differs on goals what agent needs to be done. The environment can be a simulation or real-world recording. Each approach requires large state spaces to replicate a real-world driving scenario fully. One of the first end-to-end manner learning for autonomous driving work is done by Bojarski et al. [25] using CNN by mapping one front camera to steering control using real-world recordings.

As well as their demonstrated success in ATARI games, DQN [26] algorithms can be applied to different tasks such as continuous control and motion control. Lillicrap et al. showed that Deep Q-Learning could find policies that outperform planning algorithms with full access to the dynamics of the domain and its derivatives [27]. In a 2D-like grid-based state driving simulation, Deshpande et al. used DQN that successfully learns a policy that can drive safely around pedestrians by following traffic rules [28]. The extension and successors of the DQN method are also used in autonomous driving. Zhang et al. applied the Double DQN algorithm to learn the vehicle speed control [29]. Techniques other than Q-Learning, such as Actor-Critic based algorithms, are also applied to self-driving [29]. Jaritz et al. [30] built an Asynchronous Actor-Critic (A3C) [31] framework to learn a car control policy in a realistic rally game with simultaneous agents in different maps and scenarios.

Furthermore, the central task of driving can be split into smaller tasks, such as lane-keeping assistance. Sallab et al. [32] showed that deep reinforcement learning agents could successfully learn autonomous maneuvering in a scenario of complex road curvatures and simple interaction with other vehicles for lane-keeping assistance. They also evaluated the performance of an open-source car simulator for racing called TORCS. Another study by Sallab et al. [33] proposed an end-to-end deep reinforcement learning pipeline for autonomous driving which integrates RNNs to account for Partially Observable Markov Decision Process (POMDP) scenarios.

Reinforcement learning is considered a promising direction for driving policy learning until 2017. It could be seen that all these works developed over a simulation. However, according to Pan et al. [34], training autonomous driving vehicles with reinforcement learning in a real environment involves non-affordable trial-and-error. To make it work in the real world with insufficient data, it is inevitable to use transfer

learning from a simulated environment. Pan et al. proposed a network that could convert non-realistic virtual image input into a realistic one, with a similar scene structure using GANs [35]. This was the first successful case of driving policy trained by reinforcement learning that could adapt to real-world driving data [34]. Kendall et al. [36] provided a general, easily gained reward based on the distance traveled by the vehicle without the safety driver taking control. They used a continuous, model-free deep reinforcement learning algorithm, with all exploration and optimization performed on-vehicle. This study demonstrates a new framework for autonomous driving that moves away from reliance on defined logical rules, mapping, and direct supervision.

Imitation learning can also be combined with Reinforcement Learning algorithms. CIRL [37] is an example of successfully learning a driving policy by imitating human demonstrations and using the Deep Deterministic Policy Gradient (DDPG) [27]. It uses the CARLA simulation environment to outperform previous goal-directed driving tasks. Also, this study showed that the agents had the capacity for generalization because they could act in expected ways in previously unseen environments. Peng et al. extended the Imitation Learning section of their previous study by adding a steering angle and other elements [38].

The first extensive study of urban driving with Reinforcement Learning was done by Toromanoff et al. by teaching a policy on a CARLA environment [39]. They proposed an RL method to learn to drive combined with a novel technique called implicit affordances and trained a policy with RL that could successfully complete tasks such as lane-keeping, avoiding pedestrians and obeying traffic lights in urban driving.

Most works on learning to drive with reinforcement learning agents use model-free reinforcement learning methods. The model-free methods need a large number of interactions to combine learning. However, model-based reinforcement learning methods learn from the environment and use observational data to train driving policies. For example, Henaff et al. [40] proposed to train a policy by unrolling a learned model of the environment dynamics over multiple time steps while explicitly penalizing two costs: the original cost the policy sought to optimize and an undetermined cost that represented its divergence from the states on which it was trained. A similar study was conducted by Saxena et al. [41] for model-based reinforcement learning for driving. Henaff et al. still relied on labeled datasets and

differed in the input observation, action space, and policy parameterization. They also trained a policy end-to-end, but reinforcement learning had no supervisory label [29].

2.3. Human-in-the-Loop Reinforcement Learning

Humans are expert drivers. They are familiar with the environment and know how to act even in simulation settings (assuming the simulation to be a game). They learn how to drive through feedback from others. As a result, they get better and better at driving and can drive safely without crashing. Human-in-the-loop reinforcement learning methods are known by various names, such as reward shaping [42] and inverse reinforcement learning [43]. Reward shaping can be accomplished through different kinds of feedback, such as grounded languages [44] or scalar-valued feedback [45]. Scalar-valued feedback (binary feedback) in the positive or negative allows feedback from humans with no expertise in the task.

One of the first studies in learning to drive was Sharifzadeh et al. [6], regarding large state spaces. This study proposed an inverse reinforcement learning (IRL) approach using Deep Q-Networks to extract the rewards in problems with large state spaces. They evaluated the performance of this approach in a 2D simulation-based autonomous driving scenario, which they developed. After a learning phase, the results showed that their agent could generate collision-free motions and perform human-like lane-changing behavior.

Knox et al. [6] proposed the TAMER framework, which interactively shapes agents' policy with scalar-valued human feedback. They modeled the human feedback function and chose actions based on the modeled result. This framework was much more efficient than the other methods in terms of training time. Warnell et al. [5] proposed the Deep TAMER method, an extension of TAMER. Deep TAMER uses CNNs to learn directly from high dimensional state spaces. Deep TAMER learns a successful procedure in a short amount of time for different tasks. The study also used replay memory that was generally utilized in deep reinforcement learning algorithms [47]. Arakawa et al. used both Deep TAMER and DQN by combining their weighted estimated functions and efficiently incorporated both human feedback and environment rewards.

Preference-based learning is another approach that uses humans in the Reinforcement Learning loop. In preference-based learning, the aim is to capture

human preference by showing humans two trajectories and asking them to choose the best option for them. In their essential study on preference-based learning, Christiano et al. [48] estimated human preference reward function and used this model to teach a policy to play ATARI games by using advantage actor-critic (A2C [31]) [48]. In FRESH [45], instead of comparing them, they ask humans to provide feedback on whether the trajectory was good or bad. Another study from Biyik et al. [49] combined preferences with demonstrations in a novel way. They proposed a framework for multiple information sources collected from humans. While evaluating their ideas, they also used a simple driving environment to test the idea in a lane-changing task. Navidi et al. [50] proposed a new human-AI interaction through Reinforcement-Imitation Learning. They presented a novel framework that also included an evaluator of human feedback and demonstrated that it worked in most GYM [3] environments.

The TAMER and preference learning-based approaches are also applied to robotics environments with new ideas. PEBBLE is a preferred learning method that includes unsupervised pretraining before and during the policy training, based on human preferences [51]. Lastly, Chisari et al. [49] followed Deep TAMER with unsupervised pretraining and corrective feedback. This corrective feedback was collected and applied to the playing agent directly.

3. DRIVING SIMULATION ENVIRONMENT

Simulations are an essential part of the research on self-driving cars. They make it easy to develop and evaluate driving systems. As mentioned in the Background section, researchers use simulations to teach reinforcement learning models to drive, create driving datasets, and transfer the learned knowledge to the real world [53][54].

This thesis' expectations regarding driving simulations were realistic dynamics, sensor support, having different weather conditions, and having a flexible Python API. All these main concepts are supported by the CARLA Simulator [2]. It has different cars and realistic dynamics of them, as well as different maps with supporting custom maps and weather conditions. It includes different sensors such as a simple camera, a semantic segmented camera in Cityscape's palette [23], and LIDAR. It has a superior Python API that allows the user to control cars and the environment and helps create different scenarios.

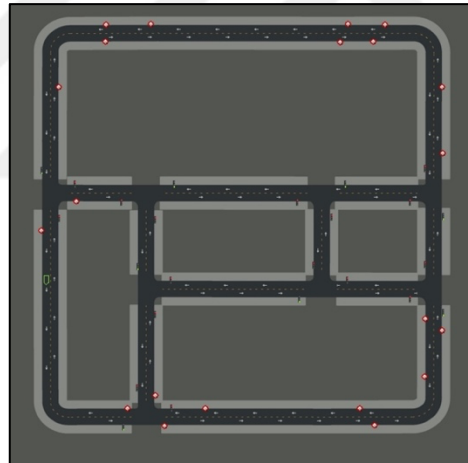


Figure 3.1: The default map of the environment chosen from the internal maps of CARLA.

CARLA includes various maps from simple towns to highways. For the starting point, we chose a simple town. The map of the town is shown in Figure 3.1. It includes junctions, curved roads, and roads separated by lines. The map also has various spawn points to spawn cars properly on the road.

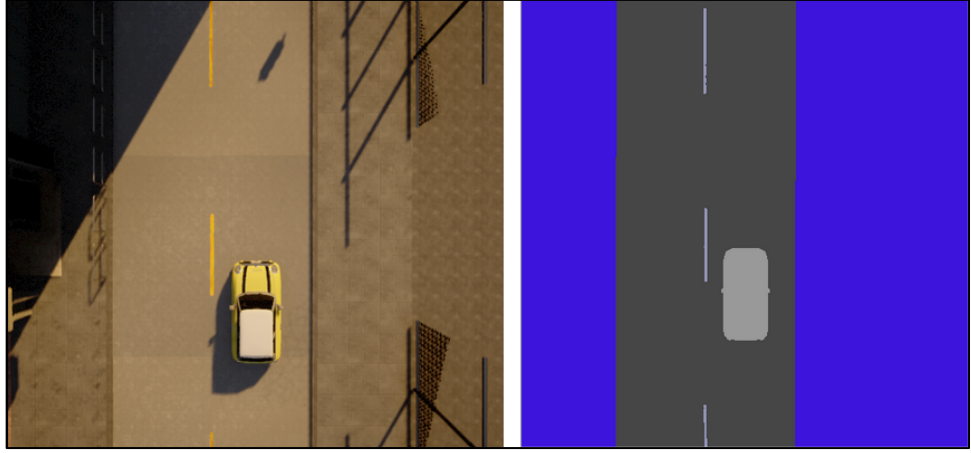


Figure 3.2: Birds eye view of the car with RGB camera (left) and its semantic segmentation (right).

In this study, the main idea was to keep the environment simple and focus more on the model and the process of the pipeline. We tried to build a 2D-like environment to train the models to achieve this simplicity. CARLA does not support a 2D camera sensor, so we put the default camera sensor on top of the car to have a bird's eye view (Figure 3.2). The front part of the car had $\frac{2}{3}$ of the image, and the back part had $\frac{1}{3}$. The camera always followed the vehicle, capturing even the car rotations. Thus, the car looked the same all the way through. A default camera sensor would provide a clear 2D view with these justifications.

The camera sensor is not that simple, even for a model with a 2D view. There are lots of objects that need to be detected by models. For this purpose, the semantic segmentation sensor is used in the same position as the camera sensor. The default sensor includes unnecessary categories such as banks and buildings. The default categories were reduced to six by mapping in-road and out-road with others. These categories are roads, lines, out of roads, cars, pipes, and uncategorized. The semantically segmented image is shown in Figure 3.2, along with an RGB version.

Because the roads are connected on the map, we selected a small short car (a Mini Cooper replica). The car takes up less space in the road, so there is more space to maneuver. The car is the agent of the models. It has only 3 actions set. These are coast, left, and right movements by a half degree. The velocity of the agent is static, and it is 20 kmh. Moving at this speed, it can easily turn corners and junctions. There is no break as action, so it keeps moving. The environment does not have any pedestrians or cars in traffic. Therefore, the environment does not check for any traffic

rules, except keeping in the right lane without collisions and not breaking any lane rules.

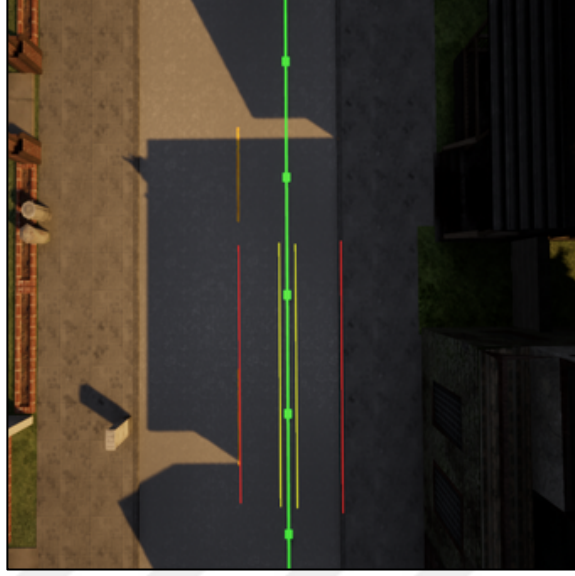


Figure 3.3: Important areas of the roads. Green line represents the center of the route. The area between yellow lines represents the positive reward section and the area between yellow and red represents negative reward section.

Reward mechanisms are an important part of the environment. The main goal of the thesis is to keep the car in the center of the road in this environment. The CARLA API provides waypoints in the center of the roads. The waypoints of the center of the road can be seen in Figure 3.3. The figure only shows the reward and road references, and these are not given with the observations. The reward is calculated based on the distance between the green line and the car. The yellow area on the figure represents the positive reward area. The red lines represent the not allowed area to drive. If the car collides with any point of the red, the episode will stop. If the car is in the yellow area, it earns a positive reward. If not, it gets a negative reward based on the distance. Let us assume that car location is l , the closest waypoint is w . So, the reward would be calculated with the equations in 1.1.

$$d = \sqrt{(l_1 - w_1)^2 + (l_2 - w_2)^2 + (l_3 - w_3)^2}$$

$$r = \begin{cases} 0.5 & \text{if } d \leq 0.5 \\ -\exp(d) & \text{otherwise} \end{cases} \quad (1.1)$$

Finally, the 2D driving simulation on top of the 3D driving simulation CARLA was complete with the reward function. It is a complete GYM-like reinforcement

environment, training and evaluating various kinds of Reinforcement Learning algorithms by providing diverse kinds of observations, rewards, and traffic rules. The next sections will discuss how to run RL algorithms and their evaluation results.



4. LEARNING to DRIVE USING REINFORCEMENT LEARNING

Deep Reinforcement Learning and Q-learning based methods have been applied to various autonomous driving tasks to teach machines to control cars in simulated environments [50][32][6]. It has also been shown that Deep Reinforcement learning models can also learn a policy to play ATARI games from high dimensional sensory input using reinforcement learning [26]. Most Atari games are 2D environments, like the one we built for driving. Some studies revealed that it was possible to learn a policy using DQN to play a 2D racing game in Open AI GYM [3].

This chapter will discuss how DQN methods are adapted to learn a driving policy in the 2D driving simulation environment end-to-end by mapping high dimensional semantic segmented images into actions. The performance of the best policy and method will be discussed, along with their issues.

4.1. Method

The DQN method has been derived from Mnih et al. [26] with some modifications in this study. It is assumed that there is a driving simulation environment, and that an agent interacts with it. The agent at every step t gets an observation $o \in O$ from the environment and takes an action $a \in A$, getting a maximum reward $r \in R$ from the environment. The goal is to learn a driving procedure by maximizing the discounted by γ cumulative reward R_T known as return, for which the reward is provided by the driving environment that prioritizes being in the center of the road.

$$R_t = \sum_t^n \gamma^t r_t \quad (4.1)$$

Assume that there is a Q function that can provide return by taking observation and action into consideration. The policy can be derived from The Q function by taking the action with maximum return.

$$\pi(o) = \arg \max_a Q(o, a) \quad (4.2)$$

Unfortunately, it is not possible to have the optimal Q function for every state, but neural networks are universal function approximates [4]. Thus, it could be possible to build a neural network to estimate the Q function. The Q network estimation can be derived from the knowledge that the Q functions are Bellman equations.

$$Q(o, a) = r + \gamma Q(o', \pi(o')) \quad (4.3)$$

The difference between two sides is called as temporal difference error and it can be used to optimize the parameters of the network that represents the Q function.

$$\delta = Q(o, a) - (r + Q(o', a')) \quad (4.4)$$

The temporal difference error is not a common loss function to optimize [56]. The Huber Loss [57] was chosen to minimize the error coming from the difference because the Q values can explode quite quickly. The Huber Loss acts like an MSE while the error is small and like an MAE while the error is large, which makes for smooth and robust learning. This is then optimized with SGD by sampling a batch from Replay Memory B . The Replay Memory stores observations as (o, a, o', r) in a buffer that lets the algorithm use the experiments multiple times.

$$L = \frac{1}{|B|} \sum_{(o, a, o', r) \in B} L(\delta) \quad (4.5)$$

where $L(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{if } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$

4.2. Training Details

In this study, the naive convolutional neural network architecture is followed up with a Q-learning algorithm to build a DQN. The model includes three convolutional layers following a linear layer that outputs the estimated Q values for each action. Batch normalization and the ReLU function are applied after every convolutional layer. Various sizes of models were used and only the results of the best one to learn a policy was reported.

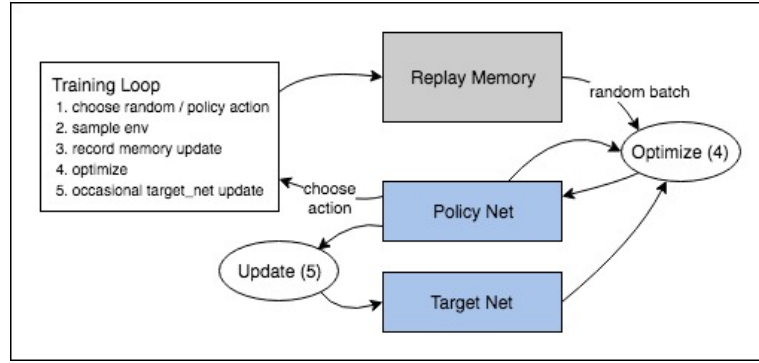


Figure 4.1: The flow diagram of the DQN training loop.

The general training usage of separating policy and target networks is observed in this study and can be seen in the overall model update flow in Figure 4.1 [58]. There are two models used in training: policy net and target net. In the beginning, those two models were initialized with the same random models. The policy net is always the trained model, and the target net is the actions-taken model. After a step, the target nets' weights are updated with policy nets. This method makes the training more stable [26]. The update step is also another hyperparameter to tune. In this case, the policy net is optimized after two episodes, although some DQN implementations do this after each step.

The study also used well-known regularizations and normalizations, such as applying batch normalization after every convolutional layer. In addition, gradient pruning was also applied in the optimization step [51]. An epsilon greedy method also allowed the model to explore actions and learn from mistakes [26]. In the original implementation, there were no random actions after the epsilon value decayed and after a step. This way, random actions will be only the beginning episodes of training. In this work, to extend the exploration of the training steps within control, the epsilon greedy method was reset after the epsilon is finished.

4.3. Results

The trained agents were evaluated with episode duration in the same map of the environment. Also, the loss was monitored to observe the stability of the learning, an important factor of these kinds of RL algorithms. The duration should increase if the model learns the driving policy. The maximum duration is kept as 300 steps, which is enough to teach a driving policy in this environment. The best model's duration and

loss graph is shown in Figure 4.2. The total reward per episode or average reward per step are not good ways of evaluating the driving policy in this environment. The reward results cannot always be stable because the agent can be in a negative position but still in the driving lane and can finish an episode without any crash or collision.

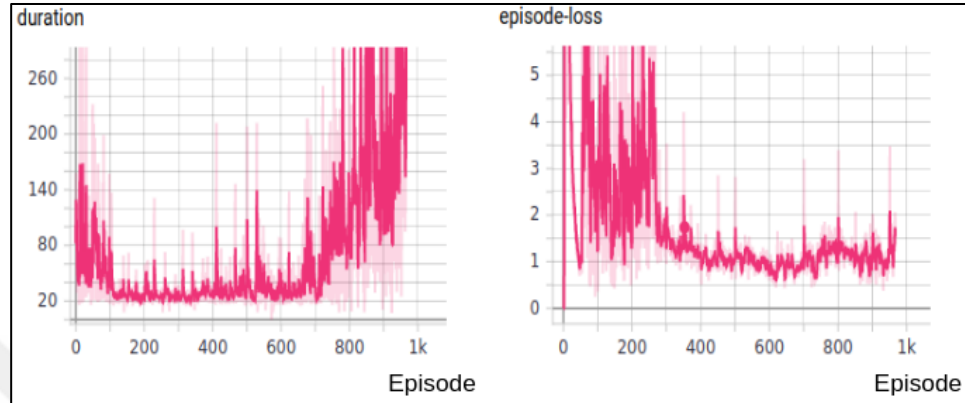


Figure 4.2: The average episode duration in evaluation phase of the agent (left) and the training loss per episode graph (right).

In the end, the DQN model learned a driving policy that could drive without crashing for a significant amount of time. It could turn right and left, follow curves and stay in the lanes. However, it was not successful in all junctions and could go out of the route.

4.4. Conclusion

The main goal was to build an end-to-end model that could learn to drive in a simulated environment. An agent successfully trained should run without crashing or getting stuck with the Deep Q Network model on the 2D environment. Still, the model has some problems, like erratic motion, uncertainties, and overfitting.

Even though the car continues to move without crashing, various problems prevent it from reaching its goal of end-to-end driving. The first issue is erratic movement. The car cannot always move stably and begins to slalom. Although sometimes it goes straight, it's not taken too much. Based on intuition, the reason behind this must be the non-derivative reward function parts. The yellow area on the reward function caused the derivation of rewards to malfunction.

Another problem is getting out of lanes. The agent gets out of lanes by turning at various turning points. This occurs not at every corner, only some of them. Also, the car tends to return to its driving lane, which narrow corners can cause. The car earns rewards on curves whether it is close to the lanes or not. The final, complex problem to solve is overfitting, which is hard to detect in the first place, as the agent was not evaluated in a different or similar map. So, the generalization of the method can be questioned. Some problems like getting out of lanes can be a sign of overfitting. Also, there are some long roads that cause overfit on coast action. Increasing replay memory and forcing spawn into various locations are tried to overcome this issue.



5. CAPTURING REWARD FUNCTIONS FROM HUMANS

Reinforcement Learning algorithms build on reward functions. A reward function could be enough to lead the algorithms to optimal policies. Rewards can be sparse or dense, and for the most part, they are fixed. Especially wrongly miss designed reward functions could be dangerous for self-driving systems [52]. The reward function could cover all the cases the designer wants in some cases unless it can have unsafe and unwanted behaviors. In Popov et al. [61], the goal was to build an agent that could stack red blocks on top of blue blocks. A failed reward example is to get a reward with the same altitude as the red block. So, the agent learns to reverse the red block instead of stacking it onto the blue block. When the agents learned policy on top of broken rewards, it could result in unexpected outcomes [62].

In most of the problems regarding self-driving cars, researchers primarily intended to have agents acting either like humans or better than humans. Humans are used as a benchmark [4] in reinforcement learning studies, as they are experts in games and driving because cars, roads, and other things are designed explicitly for them. Even if they are not experts in the environment, they can judge the status of the environment or the agent's behavior.

Adding humans to the learning loop may result in better and safer driving policies. Behavioral cloning, inverse reinforcement learning, and reward shaping are examples of algorithms that include humans. Reward shaping is a general concept that indicates engineering a reward function. Some methods use humans to shape the reward by getting feedback as grounded languages [44] or scalar-valued feedback. Having scalar feedback from humans can also result in policies that function better than humans [6][33].

Furthermore, reinforcement learning algorithms are inefficient, as they are intended to contain massive amounts of data and experiment with many cases until they learn a reasonable scenario. Even so, humans can drive safely without experiencing a crash and learn faster with guidance from others. While this might work for games or other systems, it is not easy to collect this amount of data for self-driving systems like real-world examples. The previous section reported that the DQN agent

could not produce a reasonable driving policy until 1000 episodes and that even then, the policy did not act entirely smooth and human-like.

This section discusses how to learn the reward function of autonomous driving from humans by teaching a policy on top of it. This approach aims to use humans as efficiently as possible by collecting demonstrations from them and letting them judge the latest agent while it plays in the environment. The judgment is accomplished through binary feedback with positive and negative. Also, for another point to have the human's idea, the human is allowed to annotate the actual action taken by corrective feedback. The human flow and the method are summarized and shown in Figure 5.1.

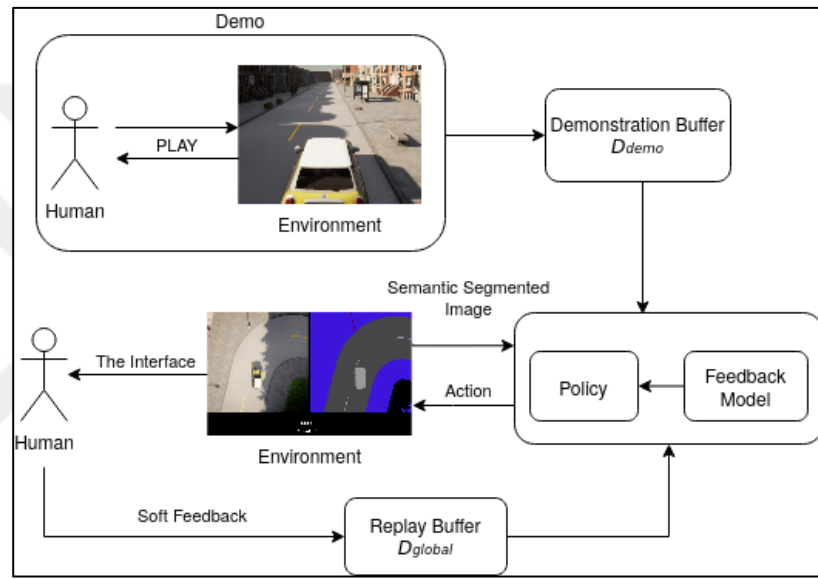


Figure 5.1: The general flow of the capturing reward function method. Human demonstrations are collected while playing the environment. The human provides feedback to the observation-action pairs generated while the agent playing the environment with most recent learned policy. The policy is updated with human demonstrations and feedback by SGD.

The proposed method is an extension of the Deep TAMER framework that works in driving simulations by improving it with demonstrations, a smooth feedback mechanism, replay scenes, and more. The human demonstrations accelerate the learning process and give an agent a chance to see the proper actions if it could not experiment with the scene. Smooth feedback is used instead of direct human feedback to optimize the use of the feedback in size and to pause before giving sharp feedback for situations in which the agent can take different actions for the same observations. Finally, the replay scene forces to restart the environment from the last position to let the agent overcome obstacles.

5.1. Preliminaries

Assume that there is an agent that interacts with a driving simulation environment in sequence time steps. In every time step t the agent receives an observation o_t from observation space O and takes action a from action space A . Actions are not continuous, like steering, but discrete, like left, right, and coast. In the classical reinforcement learning sequence, there is also a reward at time step t that is provided by the environment. But in our study, at time step t there was binary feedback from a human that was observing the system at the same time.

So, the agent-environment-human system could be formulated as a Markov Decision Process (MDP) $M = \langle S, A, H, P, p_0 \rangle$. S is the set of valid states, A is the valid action set, $P: S \times A \rightarrow P(S)$ is the transition probability function, and p_0 is the starting state distribution. As opposed to traditional RL, it is assumed that there is a hidden function reward function $H: S \times A \rightarrow R$ that the human inherently knows. In classical reinforcement learning, the aim is to build an agent that maximizes the discounted sum of rewards. In this study the goal of optimal driving policy would be derived from the estimated human feedback function $\hat{H}$ that could be used as a human feedback function. Thus, the optimal policy is the maximum reward for a state is the action taken by the estimated human function:

$$\pi(o) = \max_a \hat{H}(o, a) \quad (5.1)$$

This problem can be considered a supervised learning problem to learn human feedback. Let $\mathbf{o}$ and $\mathbf{a}$ be a sequence of observations and actions. The loss function to assess the quality of the estimation of human feedback will look like this:

$$L(\hat{H}; \mathbf{o}, \mathbf{a}, h) = \sum_{o \in \mathbf{o}, a \in \mathbf{a}} \|\hat{H}(o, a) - h\|^2 \quad (5.2)$$

The problem of maximizing the driving policy turns into minimizing the expected loss function (5.2) which is the quality of estimation of the human feedback function that can result in a successful policy to drive.

$$\hat{H}^* = \arg \min_{\hat{H}} E_{\mathbf{o}, \mathbf{a}} [L(\hat{H}; \mathbf{o}, \mathbf{a}, h)] \quad (5.3)$$

5.2. Method

The estimated Human Feedback Function $\hat{H}: O \times A \rightarrow R$ is parametrized by a deep neural network. The parameters of this deep neural network are optimized using common technique stochastic gradient descent (SGD) where η is the learning rate and the k is the update iteration, as in:

$$\hat{H}(o, a)_{k+1} = \hat{H}(o, a)_k - \eta_k \nabla_{\hat{H}} L(\hat{H}_k; \mathbf{o}, \mathbf{a}, h) \quad (5.4)$$

The neural network that represents estimated human function is shown in Figure 5.2. It takes a semantic segmented image as input, includes a CNN Encoder that trains and freezes with an autoencoder, and on the top of it, there is a feed forward layer that outputs rewards for every action, which is the only optimized part while agents play.

Updating the neural network can be summarized as follows:

- i) The policy derived by the estimated human feedback function interacts with the driving environment and generates a sequence of observable action sets.
- ii) A human observes the system and decides if the agent is going well or not, and this is collected as binary feedback for each observation and action pair.
- iii) The parameters are optimized to suit human feedback.

All three steps run asynchronously—the general algorithm of this flow is provided in Algorithm 1. The algorithm of the human feedback collection is given in Algorithm 2. The structure of the neural network is shown in Figure 5.2.

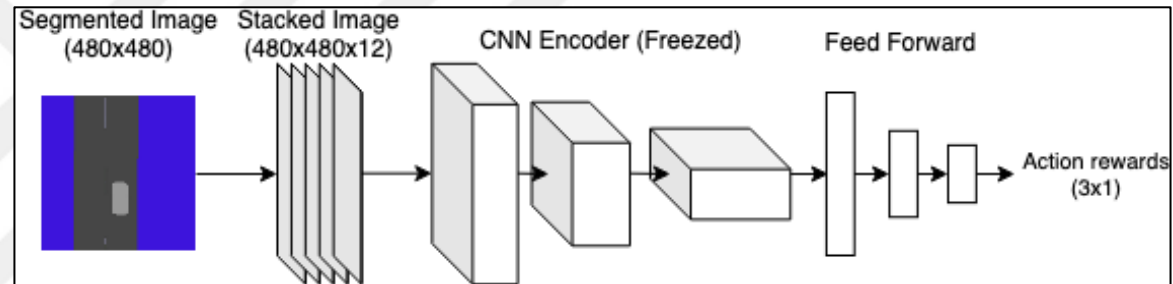


Figure 5.2: The Human Feedback Predictor model. It takes semantic segment images as input and provides estimated reward per action as output. The architecture including a pretrained CNN encoder with frozen weights followed by a feed forward layer.

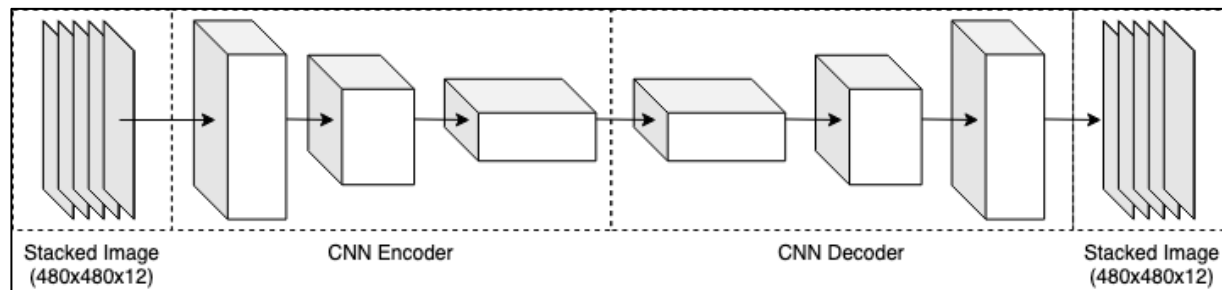


Figure 5.3: The Autoencoder architecture extract features from high dimensional semantic segmented images.

Algorithm 1 The Algorithm of Policy Updating and The Environment Loop

Require: Pretrain epoch size N_p . Initialized policy network π_θ . Train with demonstrations interval N_d . A buffer filled with human feedbacks F . Steps size to decide successful for an episode N_s . Starting locations size N_l . Starting locations L . Observation action-pairs queue to be matched with feedbacks Q_f .

```

1:  $D \leftarrow \text{demonstrations}$ 
2: for  $i \neq N_p$ ,  $i = 1, 2, \dots$  do
3:   sample batch from  $D$ 
4:   update  $\theta^\pi$  using SGD
5: end for
6: async for not interrupted,  $i = 1, 2, \dots$  do
7:   if  $i \% N_d == 0$  then
8:      $b \leftarrow \text{sample from } D$ 
9:   else
10:     $b \leftarrow \text{sample from } F$ 
11:   end if
12:   update  $\theta^\pi$  using SGD with mini-batch  $b$ 
13: end for
14: for Episodes  $i = 1, 2, \dots$  do
15:    $done \leftarrow 0$ 
16:   if steps  $< N_s$  then
17:      $l \leftarrow l_p$   $\triangleright l_p$  last positive rewarded location
18:   else
19:      $c \leftarrow i \% N_l$ 
20:      $l \leftarrow L_c$ 
21:   end if
22:    $o \leftarrow \text{Environment start from location } l$ 
23:   for not  $done$ , steps = 1, 2, ... do
24:      $a \leftarrow \pi(o)$ 
25:      $(o, done, \text{last positive location}) \leftarrow \text{execute action } a$ 
26:      $Q_f \leftarrow Q_f \cup (o, a)$   $\triangleright$  Add observation-action pairs to Queue
27:   end for
28: end for

```

Figure 5.4: The algorithm of policy updating and environment loop.

Environmental observations are high dimensional one channel semantic segmented images. The optimization of the parameters of the Human Reward Function should be near real-time to best serve the collection of feedback [5], [31]. So, to have one less parameter, we used an autoencoder (Figure 5.3) as in Deep TAMER, to encode the high dimensional space of the observations.

The autoencoder consists of encoder $\phi: F \rightarrow X$ and decoder $\psi: X \rightarrow F$. The encoder, encodes the high dimensional input to a low dimensional output and the decoder, decodes low dimensional input into high dimensional output. The autoencoders allows us to observe a hidden state that represents the high dimensional input. We built a neural network, as was the case with the encoder convolutional neural networks and the decoder with deconvolution neural networks. The network architecture can be seen in Figure 5.3. The parameters of the network are updated by minimizing the squared loss (5.5) using SGD. The observations to train the

autoencoder are collected by a human as well as a random policy, to make sure the observations cover most of the cases and in order to have a better autoencoder.

$$L(x, x') = ||x - x'||^2 \quad (5.5)$$

The collected human feedback is in a binary form that represents positive and negative reactions to the system status. There are three ways to collect feedback from humans: through demonstrations, by explicitly indicating positive or negative reactions, and by providing correct action. All the collected feedback are then converted into same shape. The feedback is then ranked between $[0.0, 1.0]$, that is, from the most negative to most positive. The usage and conversion of the feedback collected from demonstrations will be further explained in the following sections.

Algorithm 2 Soft Feedback Collection Algorithm Require: Observation action-pairs queue Q_f. A screen human observes S. A buffer to store observation-action-feedback pairs F. A function $G(x)$ that applies Gaussian filter to a signal. <pre> 1: Initialize feedback signal B 2: $f \leftarrow \text{Ignore}$ ▷ Initialize feedback status with ignore. 3: for not interrupted do 4: $(o, a) \leftarrow Q_f$ ▷ Receive observation-action pairs from queue 5: $S \leftarrow (o, a)$ ▷ Update screen. 6: $f_h \leftarrow \text{Human feedback}$ 7: if f_h is not Null then ▷ Keep same feedback status if there is no human feedback 8: $f \leftarrow f_h$ 9: end if 10: if f is positive or negative then 11: $B \leftarrow B \cup (o, a, f)$ 12: else 13: $F \leftarrow F \cup G(B)$ ▷ Apply Gaussian filter and store feedback signal 14: Empty feedback signal B 15: end if 16: if f is corrective then 17: for all actions a_l do 18: if $a_l == a$ then 19: $h \leftarrow \text{positive}$ 20: else 21: $h \leftarrow \text{negative}$ 22: end if 23: $F \leftarrow F \cup (o, a_l, h)$ 24: end for 25: end if 26: end for </pre>

Figure 5.5: The soft feedback collection algorithm.

The corrective feedback is collected from humans while the most up to date agents are playing. With corrective feedback, the human can reflect what they think about the status of the agent and directly indicate the correct action that the agent should take. The indicated action is labelled positive according to the feedback and the

other actions are labelled negative and stored in replay memory. We do not directly apply the action the humans indicate the agents to take as in Chisari et al. [49], because there will always be a delay which could cause issues with fixing the potential statuses. Corrective feedback could more efficient, but collecting large amounts of corrective feedback instead feedback-to-status could turn into behavioral cloning instead of teaching the reward function.

Giving feedback for every observation could be tiring, and humans cannot constantly maintain the same level of attention. Instead of providing feedback for every observation, we let humans give continuous feedback. The human pushes the positive or negative button, and the observations and actions are labeled as positive or negative until their status changes. The human can also continuously ignore the steps. Humans also cannot provide feedback immediately when they see an action and usually miss one or two steps while giving feedback. They can also make mistakes when annotating the states. There are also situations in which agents can take alternating actions. For example, after turning from right to left, the agent could turn left and coast, one after the other in the same sequence. The agent's control is not established by continuous steering but through discrete actions. This thesis proposes a soft feedback mechanism to fix these issues. We assume that human feedback is a signal from a status until the case is either ignored or done. So, we apply a gaussian filter to these signals before storing them in the feedback memory before the training. The smooth feedback is shown with an example in Figure 5.6.

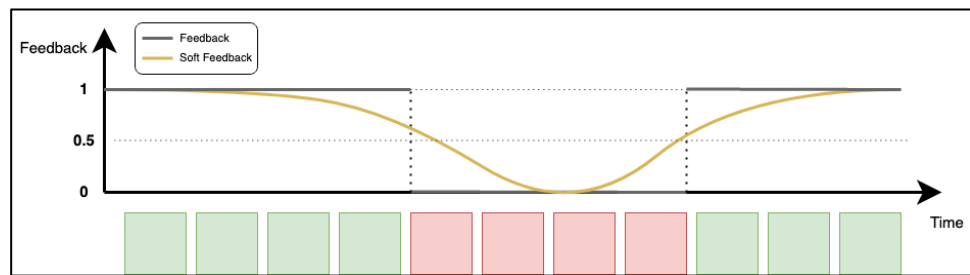


Figure 5.6: The comparison between direct feedback and soft feedback mechanism for a sample trajectory. The green boxes represent positive states, and the red boxes represent negative states.

Besides providing binary feedback, humans can also contribute by playing in the environment. Their demonstrations can be used as binary feedback, so every action taken by the human means positive feedback for the taken action and negative feedback for the others. Due to slow learning and limited data, it takes a lot of time for

the feedback to be combined, resulting in a good policy, so these demonstrations help to increase the data from human feedback. Furthermore, they help the human feedback estimator to have the same experiences as the humans.

The human feedback that is collected through demonstrations is used to set the parameters of the human feedback estimator. This leads to better starting models, which makes it more valuable and significantly easier to collect binary human feedback, as the model will not start by taking numerous random actions. Also, the estimator model is trained during training by subsampling from the demonstrations after every time j . The Replay Memory buffer, also used in Deep TAMER, allows us to use human feedback more efficiently in training multiple times. The DQN shows that the feedback replay buffer makes the learning more robust and quicker for Reinforcement Learning algorithms. As in previous studies, all the observations o actions a and feedbacks h sequences are stored in set of D_{global} . The update takes place through uniformly random sampled batches from D_{global} while optimizing the policy.

Because there are too many locations to start, the agent could not receive enough feedback in the location where it constantly crashes. It is hard to overcome its problems seeing only one time. The replay scene mechanism proposed here forces the agent to replay the same scene or start from the last good state if the agent could not stay alive for long enough. With this mechanism, the replay buffer acts as a prioritized replay buffer [63], because it includes the states that require fixing more than those already fixed before. This mechanism helps humans give more feedback for problematic states, making it slightly overfit the bad states.

5.3. Technical Details

Human-in-the-loop Machine Learning systems are more complex than classical ML systems because humans monitor the system and provide input to be considered in the flow. In Reinforcement Learning, agents learn by interacting with the environment. The environment can be a simulation, or it can be the real world. In this thesis, the environment is a driving simulation built on top of CARLA.

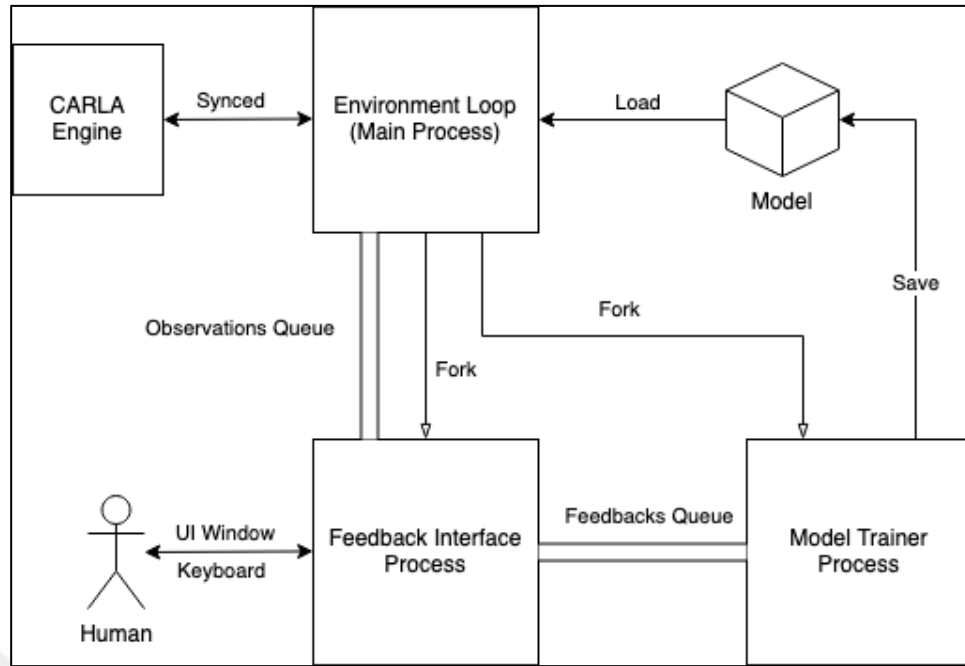


Figure 5.7 The complete design of the human-in-the-loop driving policy learning system.

The CARLA Engine is a separate program that can be operated from another server. The environment provides the synchronization between CARLA and the system. There are three primary processes: the Main Process, the Feedback Interface Process, and the Model Trainer Process. The main process creates two processes for feedback interface and model training.

The Feedback Interface Process has a queue to listen to observation-action pairs from the main process. It has a UI to show these observation-action pairs. The UI is implemented with a PyGame [64] library, which allows the creation of windows with the ability to update in a sequence and handle keyboard inputs. The Human Feedback Interface is shown in Figure 5.8. The interface includes top views, the action taken by the agent, the last feedback, and the feedback count so far. The views show the car as the agent sees it, the left is RGB, and the right is the semantic segmented version. In summary, in a loop, the process listens to the observation-action pairs queue and collects human feedback. In the end, it pushes collected observation-action-feedback pairs to the feedbacks queue.



Figure 5.8: The human feedback collection user interface. The left image shows the top view of the car, the right image shows the semantic segmentation which is the model's observation. Current action, the feedback, and the feedback count are shown at the bottom of the interface. Feedback is provided through the keyboard by using the interface window.

The human can give feedback as positive or negative, or by ignoring the action, and can provide corrective feedback with the arrow keys. The Model Trainer listens to the feedback queue and stores them in the Replay Buffer. It trains continuously by sampling from the Replay Buffer and Demonstrations Buffer. It logs the training results using TensorBoard [65]. It saves the model with updated weights to a file to be loaded by the main process. The save and load happens in a critical section using a shared mutex with the main process to prevent broken file issues.

5.4. Results and Discussions

The method proposed in this chapter to capture reward functions compared with Deep TAMER and DQN by training on the driving simulation environment shown in previous chapters. The comparison is based on the successful steps the agent has completed. The maximum number of steps in each episode is 300. This is enough to see if the agent has successfully passed the scenario. There are three scenarios that are handled by starting agents from the locations dedicated to these scenarios. These are straight road, left turn, right turn, and junction. Each scenario includes different starting points and angles except for the straight road. The straight road was tested only with a single straight location because the endings and beginnings of the other

scenarios start with a straight road. The evaluation is done on the same map as the training. However, because the starting locations are different from those of the training agents, they are evaluated in different positions on the map.

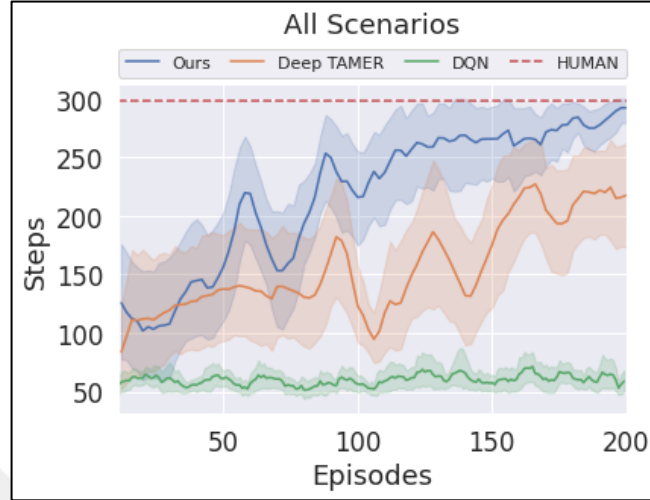


Figure 5.9: The average steps size per episode of the agents trained with different policy methods. Our method takes enough, longer steps proving that it learned a useful and better driving policy.

The models are not evaluated based on rewards' sum because rewards can lead to missed judgments. The system rewards agents in the middle of the road and punishes them in the corners. So, an agent can be passing specific steps, but then a crash can have a positive cumulative reward. Also, an agent who has completed all the steps without a collision can have a negative cumulative. Because being center in the road is safer, it is measured by comparing the average Euclidean distance from the center of all episodes. There is also a smoothness score that measures the smoothness of the actions taken by the agents by looking at the action distribution.

The same model structure is used for Deep TAMER, DQN, and the proposed model for a fair comparison. The pre-trained CNN encoder is also the same for all models. The Replay Buffer size is defined as 1024, and the batch size is also the same, at 32. Because our method has continuous feedback, we intended humans to give feedback as much as possible to have the same amount of feedback as Deep TAMER. So, we adapted the algorithm as much as possible to our environment. Lastly, because the evaluation is expensive, we saved all the models' weights after every episode and evaluated the test scenarios later.

The environment is straightforward for human evaluators, and humans can pass all scenarios successfully within 300 steps without any collisions. However, humans

cannot perform the level on the first try. So, we let them play one or two episodes to get used to the keyboard interaction. Figure 5.9 shows the average steps of each scenario's model comparisons. Overall, our method combines the feedback faster than others and reaches human performance in most scenarios in 200 episodes while others do not. The adapted Deep TAMER is the closest to our method, but its results are unstable and fluctuate. The DQN, as expected, could not show any improvements in the defined limited evaluation episodes.

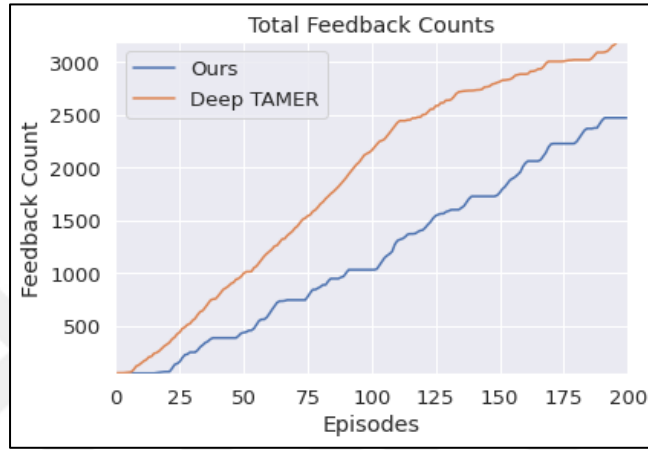


Figure 5.10: Total feedback counts in training time.

The human feedback collection effort is also improved by the methods applied over Deep TAMER. Figure 5.10 shows that our method requires less feedback from humans in total, while the model outperformed, as seen in Figure 5.9. The method reached human performance only after 2500 feedbacks, while the Deep TAMER requires more than 3000 feedbacks and still struggles to reach that level.

In the turning right-turning scenario, we see that Deep TAMER has fewer but similar successful episodes with our method (Figure 5.11). However, we clearly see that our method connected faster than Deep TAMER. Our method worked faster than all others in the left turning scenario and kept stable after reaching the human level. The adapted Deep TAMER method could not learn and could not keep up with the rate of success during the steps. Our method and Deep TAMER showed similar results in the simple straight road, and the DQN method showed some learning clues, but not enough.

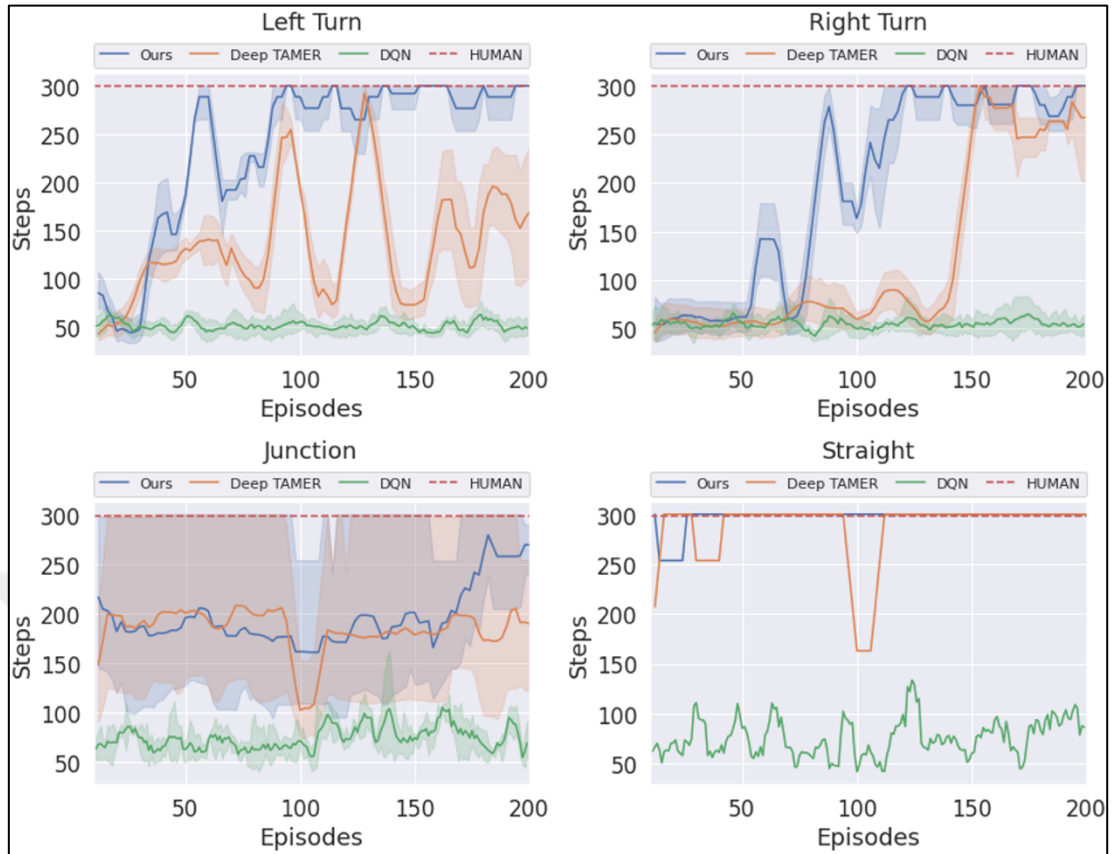


Figure 5.11: The average steps size per episode of the agents trained with different policy methods in different scenarios. Our method leaned a useful driving policy to last longer in all scenarios. In the junction scenario none of the models succeed to find an optimal policy in human-level performance.

The junction scenario is a little bit different than the others. There are three different junctions in the evaluation phase, and a junction can have multiple ways to turn. In one episode, an agent needs to turn left; in another episode, it should turn right or go straight. In this scenario, our method had better results than the others, which did not converge at acceptable levels. However, there is still room for improvement on this.

Table 5.1: The comparison of the average Euclidean distance from the center of the road and the smoothness score of the models and humans.

Method	Distance From Center	Smoothness
Human	0.31	0.046
Ours	0.33	0.089
Deep Tamer	0.39	0.205
DQN	0.46	0.658

It can be seen that the proposed method is safer than the others simply by looking at the measurements of the average distance from the center. In the distance from center measurement the lower is better and we can take less than 0.35 as safe. Humans always intend to be safe in this simple environment, and our method was the closest to humans in this safety performance. The method is more stable while taking actions than others by close to human smoothness performance (Table 5.1). In addition to these, the learned policy with human in the loop Reinforcement Learning methods does not have the erratic movement behavior problem as in the DQN method.

5.5. Summary

This chapter shows how to learn a reward function from humans to build a policy to drive in a simulated environment by proposing novel ideas. The proposed method learns to drive from a high dimensional state space end-to-end manner using human feedback instead of the inefficient number of experiments and a static reward function. It can be accepted as an extension to previous works derived from TAMER.

The human effort and the efficiency of the collected feedback are improved by demonstrations and a soft feedback mechanism. It is shown that proposed methods converge with requiring less human feedback than previous works done by collecting feedback. Human demonstrations are used to regularize and provide better initialization for the model. The proposed feedback mechanism lets humans give smooth, continuous feedback as well as corrective feedback. The applied replay scene mechanism lets the model slightly overfit to scenarios the model could not learn a successful policy with fewer data.

The method was evaluated in different scenarios and compared results with Deep TAMER that adapted to the driving environment and DQN. The proposed method learns a better driving policy by reaching human-level success in most scenarios with less than 200 episodes of training, while others did not. Also, the methods are compared based on smoothness and safety. The proposed method has similar smoothness and safety characteristics to human experts while others do not. The learned policy does not have erratic movements as in DQN.

6. CONCLUSION AND FUTURE WORK

This thesis studied the problem of learning a Deep Reinforcement Learning policy to drive in a simulated environment. First, we developed a driving simulation environment to have a simple driving environment that controls a more advanced environment. The environment is a simple 2D environment that is compatible with RL algorithms with discrete actions. Then, a generally used algorithm, DQN, is trained on the simple driving environment. The method seems able to learn an applicable driving policy. However, the final policy has safety and smoothness problems and is not well-oiled enough to compare with humans. The method also requires massive amounts of experiments to end up with a policy. Replicating existing results is difficult because of the randomness of the exploration. Also, finding a reward function that leads to a safe policy might be impossible because of the need to cover infinite cases. So, the research broadens by including humans in the learning loop.

The idea is to capture reward function by collecting human feedback instead of engineering a reward function as previous studies did with human-in-the-loop Reinforcement Learning frameworks such as Deep TAMER and TAMER. The existing solutions are applied to the autonomous driving domain by improving feedback mechanisms with smooth feedback, explanation-based learning through collecting demonstrations and replaying scenes that force the repetition of problematic scenarios to collect more feedback. The proposed method was evaluated and compared with Deep TAMER and DQN algorithms. The results showed that the proposed method learned a better driving policy that lasted longer than other methods. Moreover, the policy could drive without collision in different scenarios. It was also closest to human performance in measurements of safety and smoothness.

Feedback collection mechanisms are essential parts of human-in-the-loop Reinforcement Learning systems. Collecting more accurate and effective feedback would end up with better learned policies. Collecting feedback by showing humans two separate trajectories to choose one would be more efficient than collecting feedback after every step. Demonstrations are similar to imitation learning methods. Involving imitation learning could also bind to gathering more behavioral steps from humans.

The human-in-the-loop system seems reasonable and saves us from conducting numerous experiments. However, the study was only conducted in a simple environment, so it would be beneficial to improve the environment by adding pedestrians, other cars, and traffic lights. In addition, even if we evaluate methods in different locations that they did not see, they can also be evaluated on a different map to test their generalization. Furthermore, the method uses discrete actions; however, the actual control of a car is accomplished through continuous actions such as the angle of the steering wheel. In addition to the steering wheel, the pedals should also be used to control the car's velocity. Finally, it would be helpful to have observations in 3D camera sensors instead of 2D to have a real-world comparison.

The estimated human reward function can be used to teach a policy by training RL algorithms to outperform existing methods and to have a better, more complex system that could increase safety. For the reward function to be estimated from human feedback might also be problematic for some safety analysts. Knox et. Al. used sanity checks for reward functions in autonomous driving environments and reward shaping to identify flaws [60]. The reward functions have not yet been tested in terms of uncertainty. The estimated reward function can be used jointly with the engineered reward function.

While the human in the loop Reinforcement Learning can fix some problems of the Reinforcement Learning algorithms, it also has cons such as lack of experimenting and relying on the human monitor. Combining Reinforcement Learning and human-in-the-loop algorithms would be interesting. The combination can be done by summing up the engineered reward functions with an estimated human reward function. Also, the policies can be combined by the weighted sum of DQN's output with an estimated reward.

REFERENCES

- [1] Committee S. A. E. O.-R. A. V. S., (2014), “Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems”, International Organization for Standardization, ISO/SAE PAS 22736.
- [2] Dosovitskiy A., Ros G., Codevilla F., Lopez A., Koltun V., (2017), “CARLA: An Open Urban Driving Simulator”, Proceedings of the 1st Annual Conference on Robot Learning, 1–16, Mountain View, California, USA, 13-15 November.
- [3] Brockman G., Cheung V., Pettersson L., Schneider J., Schulman J., Tang J., Zaremba W., (2016), “OpenAI Gym”, ArXiv preprint ArXiv:1606.01540.
- [4] Mnih V., Kavukcuoglu K., Silver D., Rusu A., Veness J., Bellemare M., Graves A., Riedmiller M., Fidjeland A., Ostrovski G., Petersen S., Beattie C., Sadik A., Antonoglou I., King H., Kumaran D., Wierstra D., Legg S., Hassabis D., (2015), “Human-Level Control Through Deep Reinforcement Learning”, Nature, 518 (7540), 529–533.
- [5] Warnell G., Waytowich N. R., Lawhern V., Stone P., (2018), “Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces”, ArXiv preprint ArXiv:1709.10163.
- [6] Sharifzadeh S., Chiotellis J., Triebel R., Cremers D., (2016), “Learning to Drive Using Inverse Reinforcement Learning and Deep Q-Networks”, ArXiv preprint ArXiv:1612.03653.
- [7] Kato S., Takeuchi E., Ishiguro Y., Ninomiya Y., Takeda K., (2015), “An Open Approach to Autonomous Vehicles”, IEEE Micro, 35 (6), 60–68.
- [8] Rong G., Shin B. H., Tabatabaee H., Lu Q., Lemke S., Mozeiko M., Boise E., Uhm G., Gerow M., Mehta S., Agafonov E., Kim T. H., Sterner E., Ushiroda K., Reyes M., Zelenkovsky D., Kim S., (2020), “LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving”, IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), 1–6, Rhodes, Greece, 20-23 September.
- [9] Web 1, (2017), <https://github.com/ApolloAuto/apollo>. (Accessed: Dec 19, 2021).
- [10] Wang P., Huang X., Cheng X., Zhou D., Geng Q., Yang R., (2019), “The ApolloScape Open Dataset for Autonomous Driving and Its Application”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 42 (10), 2702-2719.
- [11] Web 2, (2020), <https://developer.nvidia.com/drive/driveworks>. (Accessed: Dec 10, 2021).
- [12] Web 3, (2018), <https://github.com/commaai/openpilot>. (Accessed: Dec 08, 2021).

- [13] Krajzewicz D., Hertkorn G., Feld C., Wagner P., (2002), "SUMO (Simulation of Urban Mobility); An Open-Source Traffic Simulation", 4th Middle East Symposium on Simulation and Modelling (MESM2002), 183–187, Sharjah, UAE, 28-30 September.
- [14] Llatser I., Jornod G., Festag A., Mansolino D., Navarro I., Martinoli A., (2017), "Simulation of Cooperative Automated Driving by Bidirectional Coupling of Vehicle and Network Simulators", IEEE Intelligent Vehicles Symposium (IV), 1881–1886, Los Angeles, California, USA, 11-14 June.
- [15] Koenig N. P., Howard A., (2004), "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2149–2154, Sendai, Japan, 2-6 October.
- [16] Michel O., (2004), "Webots: Professional Mobile Robot Simulation", Journal of Advanced Robotics Systems, 1 (1), 39–42.
- [17] Wymann B., Espié E., Guionneau C., Dimitrakakis C., Coulom R., Sumner A., (2000), "TORCS, The Open Racing Car Simulator", Sourceforge, 4 (6), 2.
- [18] Richter S. R., Hayder Z., Koltun V., (2017), "Playing for Benchmarks", IEEE International Conference on Computer Vision (ICCV), 2232-2241, Venice, Italy, 22-29 October.
- [19] Everingham M., Williams C., Zisserman A., van Gool L., Leuven K., (2006), "The 2005 Pascal Visual Object Classes Challenge", Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment, 3944 (1), 117–176.
- [20] Geiger A., (2012), "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite", IEEE Conference on Computer Vision and Pattern Recognition, 3354–3361, Providence, RI, USA, 16-21 June.
- [21] Yu F., Xian, W., Chen Y., Liu F., Liao M., Madhavan V., Darrell T., (2020), "BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling", IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2633-2642, Seattle, WA, USA, 13-19 June.
- [22] Maddern W., Pascoe G., Linegar C., Newman P., (2016), "1 Year, 1000 Km: The Oxford RobotCar Dataset", The International Journal of Robotics Research, 36 (1), 3-15.
- [23] Cordts M., Omran M., Ramos S., Rehfeld T., Enzweiler M., Benenson R., Franke U., Roth S., Schiele B., (2016), "The Cityscapes Dataset for Semantic Urban Scene Understanding", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3213-3223, Las Vegas, NV, USA, 27-30 June.
- [24] Caesar H., Bankiti V., Lang A. H., Vora S., Liong V. E., Xu Q., Krishnan A., Pan Y., Baldan G., Beijbom O., (2020), "NuScenes: A Multimodal Dataset for Autonomous Driving", IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 11618-11628, Seattle, WA, USA, 13-19 June.

- [25] Bojarski M., del Testa D. W., Dworakowski D., Firner B., Flepp B., Goyal P., Jackel L. D., Monfort M., Muller U., Zhang J., Zhang X., Zhao J., Zieba K., (2016), “End to End Learning for Self-Driving Cars”, ArXiv preprint ArXiv:1604.07316.
- [26] Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M., (2013), “Playing Atari with Deep Reinforcement Learning”, ArXiv preprint ArXiv:1312.5602.
- [27] Lillicrap T. P., Hunt J. J., Pritzel A., Heess N. M. O., Erez T., Tassa Y., Silver D., Wierstra D., (2016), “Continuous Control with Deep Reinforcement Learning”, 4th International Conference on Learning Representations, 1-14, San Juan, Puerto Rico, 2-4 May.
- [28] Deshpande N., Spalanzani A., (2019), “Deep Reinforcement Learning Based Vehicle Navigation Amongst Pedestrians Using a Grid-Based State Representation”, IEEE Intelligent Transportation Systems Conference (ITSC), 2081–2086, Auckland, New Zealand, 27-30 October.
- [29] Zhang Y., Sun P., Yin Y., Lin L., Wang X., (2018), “Human-like Autonomous Vehicle Speed Control by Deep Reinforcement Learning with Double Q-Learning”, IEEE Intelligent Vehicles Symposium (IV), 1251–1256, Gothenburg, Sweden, 19-22 June.
- [30] Jaritz M., de Charette R., Toromanoff M., Perot E., Nashashibi F., (2018), “End-to-End Race Driving with Deep Reinforcement Learning”, IEEE International Conference on Robotics and Automation (ICRA), 2070–2075, Brisbane, Australia, 21-26 May.
- [31] Mnih V., Badia A. P., Mirza M., Graves A., Lillicrap T. P., Harley T., Silver D., Kavukcuoglu K., (2016), “Asynchronous Methods for Deep Reinforcement Learning”, ArXiv preprint ArXiv:1602.01783.
- [32] Sallab A., Abdou M., Perot E., Yogamani S., (2016), “End-to-End Deep Reinforcement Learning for Lane Keeping Assist”, ArXiv preprint ArXiv:1612.04340.
- [33] Sallab A., Abdou M., Perot E., Yogamani S., (2017), “Deep Reinforcement Learning Framework for Autonomous Driving”, Electronic Imaging, 29 (19), 70–76.
- [34] Pan X., You Y., Wang Z., Lu C., (2017), “Virtual to Real Reinforcement Learning for Autonomous Driving”, Proceedings of the British Machine Vision Conference, 11-13, London, GB, 4-7 September.
- [35] Goodfellow I. J., Abadie J., Mirza M., Xu B., Farley D., Ozair S., Courville A., Bengio Y., (2014), “Generative Adversarial Networks”, Advances in Neural Information Processing Systems, 27-32, Montreal, QC, Canada, 8-13 December.
- [36] Kendall A., Hawke J., Janz D., Mazur P., Reda D., Allen J. M., Lam V. D., Bewley A., Shah A., (2019), “Learning to Drive in a Day”, International

Conference on Robotics and Automation (ICRA), 8248-8254, Montreal, QC, Canada, 20-24 May.

- [37] Liang X., Wang T., Yang L., Xing E., (2018), “CIRL: Controllable Imitative Reinforcement Learning for Vision-Based Self-Driving”, European Conference on Computer Vision, 604–620, Munich, Germany, 8-14 September.
- [38] Peng M., Gong Z., Sun C. Chen L., Cao D., (2020), “Imitative Reinforcement Learning Fusing Vision and Pure Pursuit for Self-Driving”, IEEE International Conference on Robotics and Automation (ICRA), 3298–3304, Paris, France, 31 May-31 August.
- [39] Toromanoff M., Wirbel É., Moutarde F., (2020), “End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances”, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 7151–7160, Seattle, WA, USA, 13-19 June.
- [40] Henaff M., Canziani A., LeCun Y., (2019), “Model-Predictive Policy Learning with Uncertainty Regularization for Driving in Dense Traffic”, ArXiv preprint ArXiv:1901.02705.
- [41] Saxena D. M., Bae S., Nakhaei A., Fujimura K., Likhachev M., (2020), “Driving in Dense Traffic with Model-Free Reinforcement Learning”, IEEE International Conference on Robotics and Automation (ICRA), 5385-5392, Paris, France, 31 May-31 August.
- [42] Wiewiora E., (2010), “Reward Shaping”, Encyclopedia of Machine Learning, 1 (1), 863–865.
- [43] Abbeel P., Ng A. Y., (2004), “Apprenticeship Learning via Inverse Reinforcement Learning”, Proceedings of the Twenty-First International Conference on Machine Learning, 1, New York, NY, USA, 4-8 July.
- [44] Waytowich N. R., Barton S. L., Lawhern V., Warnell G., (2019), “A Narration-Based Reward Shaping Approach Using Grounded Natural Language Commands”, ArXiv preprint ArXiv:1911.00497.
- [45] Xiao B., Lu Q., Ramasubramanian B., Clark A., Bushnell L., Poovendran R., (2020), “FRESH: Interactive Reward Shaping in High-Dimensional State Spaces Using Human Feedback”, ArXiv preprint ArXiv:2001.06781.
- [46] Knox W. B., Stone P., (2009), “Interactively Shaping Agents via Human Reinforcement: The TAMER Framework”, Proceedings of the Fifth International Conference on Knowledge Capture, 9–16, Redondo Beach, California, USA, 1-4 September.
- [47] Liu R., Zou J. Y., (2018), “The Effects of Memory Replay in Reinforcement Learning”, 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton), 478–485, Monticello, IL, USA, 2-5 October.

- [48] Christiano P., Leike J., Brown T. B., Martic M., Legg S., Amodei D., (2017), “Deep Reinforcement Learning from Human Preferences”, ArXiv preprint ArXiv:1706.03741.
- [49] Biyik E., Losey D. P., Palan M., Landolfi N. C., Shevchuk G., Sadigh D., (2021), “Learning Reward Functions from Diverse Sources of Human Feedback: Optimally Integrating Demonstrations and Preferences”, The International Journal of Robotics Research, 41 (1), 45–67.
- [50] Navidi N., Landry R., (2021), “New Approach in Human-AI Interaction by Reinforcement-Imitation Learning”, Applied Sciences, 11 (7), 3068.
- [51] Lee K., Smith L., Abbeel P., (2021), “PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-Training”, ArXiv preprint ArXiv:2106.05091.
- [52] Chisari E., Welschhold T., Boedecker J., Burgard W., Valada A., (2022), “Correct Me If I Am Wrong: Interactive Learning for Robotic Manipulation”, IEEE Robotics and Automation Letters, 7 (2), 3695–3702.
- [53] Grigorescu S. M., Trasnea B., Cocias T. T., Macesanu G., (2020), “A Survey of Deep Learning Techniques for Autonomous Driving”, ArXiv preprint ArXiv:1910.07738.
- [54] Kiran B. R., Sobh I., Talpaert V., Mannion P., Sallab A. A. al, Yogamani S. K., P’erez P., (2020), “Deep Reinforcement Learning for Autonomous Driving: A Survey”, ArXiv preprint ArXiv:2002.00444.
- [55] Watkins C. J. C. H., Dayan P., (1992), “Q-Learning”, Machine Learning, 8 (3), 279–292.
- [56] Hessel M., Modayil J., Hasselt H., Schaul T., Ostrovski G., Dabney W., Horgan D., Piot B., Azar M. G., Silver D., (2017), “Rainbow: Combining Improvements in Deep Reinforcement Learning”, ArXiv preprint ArXiv:1710.02298.
- [57] Meyer G. P., (2021), “An Alternative Probabilistic Interpretation of the Huber Loss”, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 5257–5265, Virtual, 19-25 June.
- [58] Web 4, (2021), https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html. (Accessed: Jun 01, 2021).
- [59] Han S., Mao H., Dally W., (2016), “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”, ArXiv preprint ArXiv:1510.00149.
- [60] Knox W. B., Allievi A., Banzhaf H., Schmitt F., Stone P., (2021), “Reward (Mis)Design for Autonomous Driving”, ArXiv preprint ArXiv:2104.13906.
- [61] Popov I., Heess N. M. O., Lillicrap T. P., Hafner R., Barth-Maron G., Vecerík, M., Lampe T., Tassa Y., Erez T., Riedmiller M. A., (2017), “Data-Efficient Deep

Reinforcement Learning for Dexterous Manipulation”, ArXiv preprint ArXiv:1704.03073.

- [62] Web 5, (2016), <https://openai.com/blog/faulty-reward-functions/>. (Accessed: Aug 08, 2021).
- [63] Schaul T., Quan J., Antonoglou I., Silver D., (2016), “Distributed Prioritized Experience Replay”, ArXiv preprint ArXiv:1511.05952.
- [64] Web 6, (2011), <https://www.pygame.org/>. (Accessed: Jul 28, 2021).
- [65] Abadi M., Agarwal A., Barham P., (2016), “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems”, ArXiv preprint ArXiv:1603.04467.



BIOGRAPHY

M.Cemil Güney graduated from Gebze Technical University in March 2018 with a bachelor's degree in Computer Engineering. During the following years he was employed as Machine Learning Engineer and Data Scientist in several companies. In February 2019, he started the Computer Engineering Master's program at the Gebze Technical University.



APPENDICES

Appendix A: The Publications about the Thesis

Güney M.C., Genç Y., (2021), "Capturing Reward Functions for Autonomous Driving: Smooth Feedbacks, Random Explorations and Explanation-based Learning", 3rd International Conference on Artificial Intelligence and Applied Mathematics in Engineering (ICAIAME 2021), Antalya, Turkey, 1-3 Oct.

