
**NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS**

**SCHOOL OF SCIENCE**
**DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION**


**BSc THESIS**

# End-To-End On-Policy Reinforcement Learning on a Self-Driving Car in Urban Settings


**Konstantinos V. Dimitrakopoulos**

**Supervisor:** **Dimitrios Gunopulos, Professor**



**ATHENS**

**OCTOBER 2023**

**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Ενισχυτική Μάθηση από Άκρο-Σε-Άκρο με έμφαση στη Πολιτική σε ένα Αυτόνομο Όχημα σε Αστικό Περιβάλλον

**Κωνσταντίνος Β. Δημητρακόπουλος**

**Επιβλέπων:** **Δημήτριος Γουνόπουλος,** Καθηγητής

**ΑΘΗΝΑ**

**ΟΚΤΩΒΡΙΟΣ 2023**

**BSc THESIS**

# End-To-End On-Policy Reinforcement Learning on a Self-Driving Car in Urban Settings

**Konstantinos V. Dimitrakopoulos**
**S.N.:** 1115201500034

**SUPERVISOR:** **Dimitrios Gunopulos, Professor**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

Ενισχυτική Μάθηση από Άκρο-Σε-Άκρο με έμφαση στη Πολιτική σε ένα Αυτόνομο Όχημα σε Αστικό Περιβάλλον

**Κωνσταντίνος Β. Δημητρακόπουλος**

**Α.Μ.:** 1115201500034

**ΕΠΙΒΛΕΠΩΝ:** **Δημήτριος Γουνόπουλος,** Καθηγητής

# ABSTRACT

Urban Driving is undeniably one of the hardest problems to resolve in the Autonomous Driving field [5]. The high complexity of its indeterministic environment requires a massive set of variables to be estimated for the successful operation of the autonomous vehicle. The self-driving car must be able to navigate safely through the environment, having a full understanding of the scene.

This thesis provided a comprehensive analysis of the essential tools to resolve the problem of Urban Driving, including the key components, architectures, and machine learning in Autonomous Driving Systems. In addition, it reviewed recent approaches that brought remarkable results and highlighted their core features.

In the end, it combined these features, to implement a model, following an End-To-End architecture and trained with Reinforcement Learning, that can navigate safely inside a CARLA urban environment, following the traffic rules, respecting the other scene actors, and avoiding possible dangers.

Regarding the model evaluation, metrics analysis and visualization, including heading deviation, episode length and received rewards, highlight the improvement of the model in decision making, during training.

The final model is capable of navigating through urban settings, completing easy tasks, including lane following, heading, positioning, and avoiding collisions with preceding vehicles.



**SUBJECT AREA**: Autonomous Driving

**KEYWORDS**: Self-Driving Cars, Reinforcement Learning, Urban Driving, End-To-End Reinforcement Learning, On-policy Reinforcement Learning

# ΠΕΡΙΛΗΨΗ

Η Αστική Οδήγηση είναι αναμφισβήτητα ένα από τα δυσκολότερα προβλήματα που πρέπει να επιλυθούν στον τομέα της Αυτόνομης Οδήγησης. Η υψηλή πολυπλοκότητα του μη ντετερμινιστικού περιβάλλοντός της προϋποθέτει την εκτίμηση ενός τεράστιου συνόλου μεταβλητών για την επιτυχή πλοήγηση του αυτόνομου οχήματος. Το αυτόνομο όχημα πρέπει να είναι σε θέση να πλοηγείται με ασφάλεια στο περιβάλλον, έχοντας την πλήρη κατανόηση του.

Η παρούσα πτυχιακή εργασία παρείχε μια ολοκληρωμένη ανάλυση των βασικών εργαλείων για την επίλυση του προβλήματος της αστικής οδήγησης, συμπεριλαμβανομένων των βασικών στοιχείων, των αρχιτεκτονικών και της μηχανικής μάθησης στα συστήματα αυτόνομης οδήγησης. Επιπλέον, εξέτασε πρόσφατες προσεγγίσεις που έφεραν αξιοσημείωτα αποτελέσματα και τόνισε τα βασικά χαρακτηριστικά τους.

Στο τέλος, συνδύασε αυτά τα χαρακτηριστικά, για να υλοποιήσει ένα μοντέλο, που ακολουθεί την End-To-End αρχιτεκτονική και εκπαιδεύτηκε με Ενισχυτική Μάθηση, το οποίο μπορεί να πλοηγείται με ασφάλεια μέσα σε ένα αστικό περιβάλλον του προσομοιωτή CARLA, ακολουθώντας τους κανόνες κυκλοφορίας, σεβόμενο τους άλλους παράγοντες στη σκηνή και αποφεύγοντας πιθανούς κινδύνους.

Όσον αφορά την αξιολόγηση του μοντέλου, η ανάλυση μετρικών και η οπτικοποίηση τους, συμπεριλαμβανομένης της απόκλισης από την ιδανική πορεία του οχήματος, του μήκους των επεισοδίων και των λαμβανόμενων ανταμοιβών, αναδεικνύουν τη βελτίωση του μοντέλου στη λήψη αποφάσεων, κατά τη διάρκεια της εκπαίδευσης.

Το τελικό μοντέλο είναι ικανό να πλοηγείται σε αστικό περιβάλλον, πραγματοποιώντας εύκολες εργασίες, συμπεριλαμβανομένων της ομαλής πορείας σε λωρίδα κυκλοφορίας, της κατεύθυνσης και τοποθέτησης του οχήματος, καθώς και της αποφυγής συγκρούσεων με προπορευόμενα οχήματα.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ**: Αυτόνομη Οδήγηση

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ**: Αυτόνομα Οχήματα, Ενισχυτική μάθηση, Αστική οδήγηση, Ενισχυτική Μάθηση αρχιτεκτονικής Άκρο-Σε-Άκρο, Ενισχυτική Μάθηση με έμφαση στη Πολιτική

# CONTENTS

# LIST OF FIGURES

# LIST OF IMAGES

# LIST OF TABLES

# PREFACE

The thesis was developed within the undergraduate programme of the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens, under the supervision of Professor Dimitrios Gunopulos, whom I would like to thank for his valuable scientific assistance and guidance.

# 1. The Urban Driving Problem

Undeniably driving optimally in urban settings is one of the hardest tasks to accomplish in autonomous driving. Being computationally large, it requires a massive set of parameters to be considered for the successful operation of the agent guiding the autonomous vehicle. The car must be able to navigate safely, owning a full understanding of the environment, acknowledging its position and possible actions, while always choosing the best one. Navigating through lanes of traffic, interacting with intersections, following traffic lights, acknowledging other scene actors, as also predicting possible dangers constitute a typical driving scenario in urban settings, which the agent must cope with.

Ensuring safety is a key concept of driving in an urban environment. The agent must be able to predict human error and predict the best action to provide the optimal results. Cars driving in the opposite lane, exceeding maximum speed, violating traffic lights and rules are phenomena that should be recognized and handled explicitly. More than 94% of road accidents are caused by human errors [1]. This implies the need for refined driving experience, preventing the above errors and leading to a rapid evolution of the autonomous driving systems overall.

Vehicle positioning is a crucial factor for a successful agent. As the car decides to follow a specific lane, it must retain a stable behavior, not deviating from the center of the lane, until the exit. While in the lane, the agent must keep a reasonable distance from the car proceeding, keep a stable in below speed limit velocity while possible, avoiding nervous driving and abrupt changes in behavior. In the end, positioning must inspire safety to the agent from near vehicles.

Successful steering in intersections, complying with the way traffic lights work and interacting with nearby vehicles are also of immense importance. The agent must reduce speed if needed, detect and avoid upcoming vehicles, wait until the green traffic light, if any, and finally turn with respect to the other vehicles. As the agent turns, decides the next lane to follow.

The vehicle must obey the speed limit rules, especially in urban settings. Ideally, the vehicle must follow an optimal speed, dependent on its environment, including traffic lights state, nearby vehicles, speed limit, intersections, and obstacles.

All these parameters, regarding the uncertainty of acting in an indeterministic environment, define an exceedingly complex problem, considering the number of configurable parameters [1], which is computationally hard.

This thesis highlights most parameters composing the urban driving problem, analyses the most requirements that need to be utilized to encounter the problem (i.e. autonomous vehicles architecture, machine learning in autonomous driving), and utilizes them to implement a model that can cope with the problem of urban driving.

# 2. Analyzing The Autonomous Driving System

To encounter the autonomous driving problem in urban settings, it is essential to analyze the architecture of an autonomous vehicle, to perceive the way they operate in the environment.

An autonomous driving system utilizes sensor streams to receive input data, known as observations of the environment. Processing the observations is dependent on the system architecture. Following the processing, the actions predicted are forwarded to the vehicle actuators, that alter the vehicle's state as the vehicle accelerates, decelerates, or steers. The interpretability of the system heavily depends on the architecture.

## 2.1 Automation Levels

Autonomous driving systems are categorized according to their automation level. The Society of Automotive Engineers (SAE) identifies five levels of automation [2].

In level zero, there is no automation at all. The vehicle does not have an autonomous driving system and can only be guided manually.

In level one, regarding minor automation, the driver can utilize assistance systems, such as adaptive cruise control, antilock braking systems and stability control [2].

In level two, detectors for collision avoidance and emergency braking systems are integrated offering protection through partial automation.

In level three, a driver is not required for the vehicle navigation in the environment continuously and can temporarily disregard all driver obligations, although s/he must be ready to take over in case an emergency alert occurs. This level of automation takes place in operational design domains (ODDs), usually for autonomous cars driving on highways and not in urban settings.

In level four, the vehicle can navigate in complete automation only in certain operational design domains, granted with specific utility, such as highly detailed maps and infrastructure. Departing these operational design domains leads the vehicle to automatically park itself. Used in urban settings.

In level five, there is complete driving automation, in all road settings and weather conditions. The attention of the driver is not needed to any degree.

Automation of level three was easily reached, while being present in most modern cars and utilized in most urban environments. Level four and five form the real challenge. Toyota Research Institute stated that no one in the industry is close to attaining level five automation [2], while system failures have led to many accidents due to false estimation of weather, speed, and detection of other vehicles.

## 2.2 Architecture

The Autonomous Driving Systems are divided into two dissimilar algorithmic approaches, or architectures, the modular and the end-to-end approach [3]. The modular or mediated perception approach is composed by interconnected task-specific modules and is widely used by the industry nowadays, in both urban and rural environments, while the end-to-end or behavior-reflex approach is widely used in research and supports the optimization

of the entire driving pipeline, from receiving sensor data (initial end), to the resulting driving actions (final end), as a machine learning trained neural network.

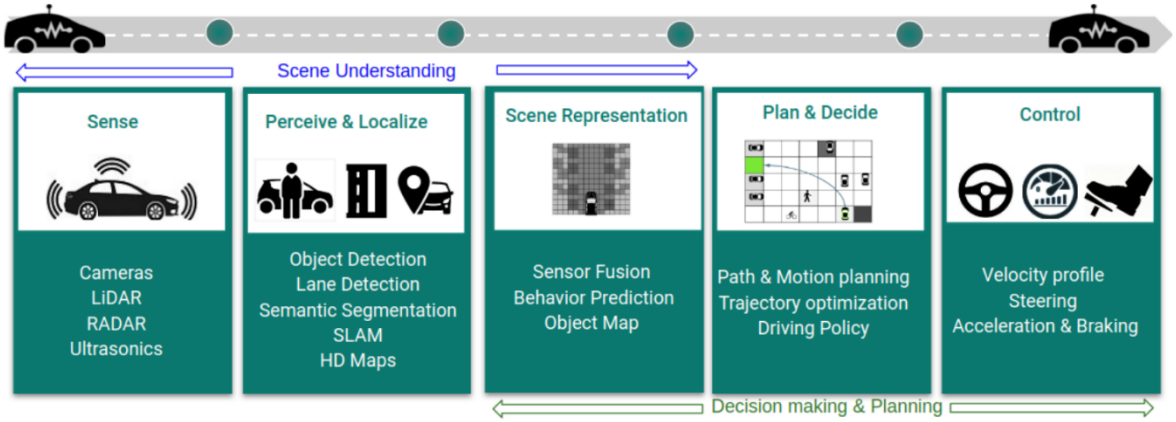


**Image 1: Autonomous Driving Systems module pipeline [1].**

### 2.2.1 Modular Approach

The modular approach consists of multiple interconnected modules resulting in a pipeline of subsystems connecting the retrieval of sensor data and the passing of driving actions to vehicle actuators [2]. The basic form of a modular approach consists of localization, mapping, perception, assessment, planning, decision making, vehicle control and human-machine interaction [2]. Sensor data are forwarded through all these modules (Image 1), to retrieve the resulting driving action.

A major advantage of modular approach is the deconstruction of autonomous driving task to sub-tasks, where each one can be addressed separately and has its own literature. Building autonomous tasks enables redundancy and boosts reliability, as it makes failure detection easier, which is particularly important in urban driving. On the downside, such architecture favors error propagation between the modules and increases the overall system complexity.

### 2.2.2 End-To-End Approach

The end-to-end approach refers to a single ego-motion interconnecting sensor stream input data to the final actuators (Image 2) [2]. This ego-motion may produce results from continuous actions (i.e. steering or throttle state), to discrete commands (i.e. “turn left”, “keep straight”).

The intermediate neural network could be trained via supervised deep learning, neuroevolution or deep reinforcement learning. Neuroevolution holds the advantage of avoiding backpropagation, which does not require supervision [2]. On the other hand, neuroevolution and reinforcement learning require the agent to act inside the environment while training, which cannot be done offline.

Implementations using convolutional neural networks, temporal, and spatiotemporal networks also have a strong presence. Imitation learning seeks to imitate the behavior of the expert demonstrator-driver, bypassing the aspiration to achieve better than human performance.

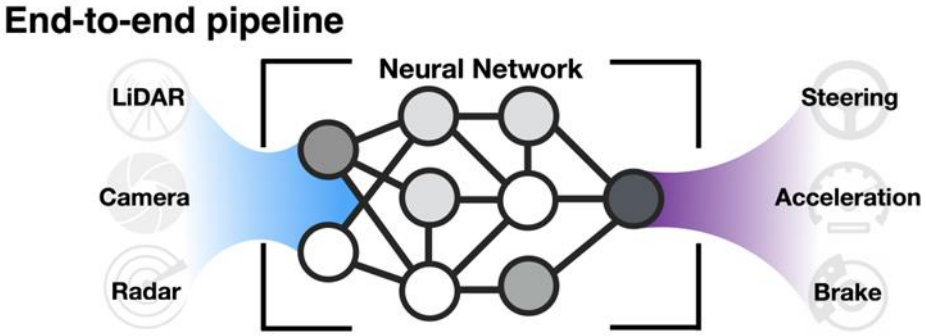


**Image 2: The end-to-end approach [3].**

Despite the implementation of various end-to-end systems for research, including offroad autonomous driving, there have been no real-world implementations yet, as they cannot cover basic safety measures and have poor system interpretability. Most self-driving cars implemented for research have been trained via Imitation Learning.

### 2.2.3 Comparison of Modular and End-To-End approaches

The **Modular Approach** on Autonomous Driving Systems represents a fine-grained pipeline of modules, whose interdependencies are inspired from robotics literature [3]. As a result, many of these modules make use of a robotic operational system (ROS). It consists of well-defined subtasks, which autonomous engineering leads to independent improvement of the whole driving task in total. The decisive action selected by the system is a result of deterministic rules on intermediate representations. The system is easily interpretable, making it easy to track down each error (i.e. class misdetection), which is useful in the challenging task of urban driving. On the downside, it offers limited results and scene understanding and cannot consider all driving scenarios, due to information dropped through subsystems (i.e. no hand rules for representing every unique obstacle through semantic segmentation or limit the count of detected obstacles to consider only the important ones). A complete understanding of the urban scene is crucial for achieving optimal behavior.

The **End-To-End Approach** on Autonomous Driving Systems refers to a single learning task determined by a neural network, as it converges to an optimal intermediate representation. No information is cut as there is no information bottleneck and all implicit knowledge is maintained. Neural networks can learn “task-specific features and build task-specific representations” [2], as end-to-end models achieve state-of-the-art results in games, natural language processing and computer vision. Problems with similar complexity as urban autonomous driving are solved via neural networks, which is encouraging. On the downside, the interpretability of neural networks is poor, finding the initial cause of an error is complex and it remains unknown how the model reached a decision.

The End-To-End systems constitute a promising approach to encounter the autonomous urban driving problem, as they have solved comparably complex problems and are not limited in urban scene understanding. The implementation of this thesis follows the End-To-End approach.

**Image 3: Feed from Omnidirectional camera [2].**

## 2.3 Input Modalities

The Self-Driving Car carries installed on-board sensors, to capture the urban scene. High Sensor redundancy offers robustness and reliability on the autonomous driving system. Many input modalities provide inherent knowledge, while combined sensors improve the generalization and accuracy of the system [2,3].

- **Cameras:** The most common sensor in autonomous driving systems is the monocular camera, able to detect color, especially useful for traffic light detection. Although, exposure to certain illumination conditions brings decreased performance. Definition of depth is hard using a single camera, as for depth-information-extraction two monocular cameras are used. Omnidirectional cameras are used to capture a larger field of view (Image 3), while event cameras record data asynchronously on events (i.e. brightness change). Temporal information can be extracted by utilizing past frames. The surrounding view is significantly useful in lane changes and intersections. Semantic segmentation cameras and depth maps provide additional semantic information.
- **Radar/Lidar:** Radar provides depth 3D information by emitting radio waves. It is lightweight and cost-effective. Lidar emits light waves, which travel for a shorter distance while being more accurate than radar. Both are not affected by weather or illumination conditions and can be transformed to a 3D or 2D bird-eye-view cloud, as an intermediate representation.
- **Proprioceptive sensors:** Provide information about speed, acceleration, and yaw of the ego vehicle.
- **Navigational inputs:** Provide navigational commands (i.e. “go left”, “keep straight”, “follow vehicle”, “drive aggressively”, “keep to the right of lane”), as well as a route planner-GPS.
- **HD maps:** Provide accurate localization and detailed information about the driving scene, such as lanes, roads, intersections, crossings, traffic signs, speed limits, dynamically changing traffic lights [3].

- **Multi modal fusion:** Fusion of information inside the network on early, middle, or late stage, depending on the desired level of feature extraction.
- **Multiple timesteps:** Extract features like speed and acceleration, from multiple past non-temporal inputs (i.e. monocular camera), usually from an RNN.

Selecting the ideal combination of input sensors, depending on the characteristics of the urban environment, as also considering the cost-effect of each one, is essential for the implementation of the autonomous driving system.

### 2.4 Output Modalities

The output modalities play a massive role as they determine the agent's behavior through the selected actions and change the state of the ego-vehicle. It is important for the agent to acknowledge the significance of every action selected and why it was chosen.

- **Steering / Speed:** Information is provided from the ego-car, while the next step speed and acceleration is predicted [3]. These metrics can be independent of the car's geometry. Additionally, predicting acceleration and angular velocity further improves the model.
- **Waypoints:** Higher level modality. Used for predicting future ego-vehicle waypoints and desired ego-vehicle trajectory. Most applied to control algorithms, such as PID. The extracted waypoints can be further smoothened and transformed into a curve.
- **Cost Maps:** Path creation indicating where it is safe to drive. Firstly, create and evaluate trajectories through MPC, then transform them to 2D cost maps. Image 4 illustrates the cost volume across time, where the planned trajectory is shown as a red line and the estimated cost is described in colour heat areas [3].
- **Direct Perception and Affordances:** A method laying between modular and end-to-end approaches. Instead of total input, retrieve a small set of crucial indicators called affordances [3].
- **Multitask Learning:** Predict and optimize multiple outputs concurrently (i.e. predict nearby cars trajectories).

Waypoints and cost maps should provide a detailed representation of the urban environment, reinforcing scene understanding.

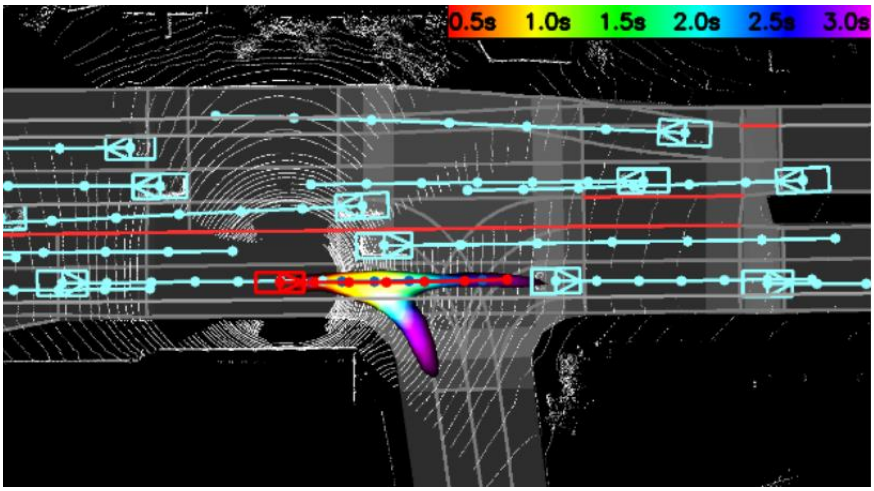


**Image 4: Cost Volume across Time [3].**

# 3. Learning Methods in Autonomous Driving

The most common learning approaches for “solving the Urban Driving problem in End-to-End Autonomous Driving are Imitation Learning and Reinforcement Learning” [3]. Both have a massive impact on academia and industry. This section highlights the characteristics of each method, to indicate the best-suited for utilizing in Urban Driving.

## 3.1 Imitation Learning

Imitation Learning, often referred to as behavioral cloning, is a supervised learning method, where the neural network is trained through mimicking the behavior of an expert [3]. It is the dominant paradigm in real world autonomous driving tasks. The driver's actions are correlated with driving commands, while the sensory input feeds the appropriate data for network training. Regarding the simplicity of the method, the model can acquire a massive amount of data and can successfully operate on simple tasks such as lane following. On the downside, this method tends to produce fewer satisfying results while operated on rare and complicated scenarios, like an urban scene.

Many real-world cases of applied Imitation learning produced impressive results. ALVINN was the first real-world self-driving car produced through imitation learning [12]. Utilizing a camera and a radar was able to predict steering wheel angle and perform lane following on simple roads. NAVLAB car, making use of ALVINN prototype, was able to successfully follow a lane on public roads [3]. DAVE was trained to avoid collisions with obstacles, making use of two cameras to estimate depth. Finally, NVIDIA trained a large-scale convolutional neural network, producing a car able to navigate on highways and residential roads [13].

One of the major disadvantages of Imitation Learning is **Distribution Shifting** [3]. While the method mimics the expert, all actions produced are a result of observations which the expert caused. In general, any output produced a new observation. This leads to situations where the agent cannot predict a valid action as it is in an unseen condition. The general driving actions differ a lot from the expert drivers' actions, as the expert driver makes no mistakes. The self-driving car will never experience a heading deviation case in training, where the agent must act on restoring the car back in the middle of the lane.

Generalization of data is of immense importance, to provide a solution to the above problem [3]. The most common methods of facing this case are Data augmentation, data diversification and on-policy learning [3].

- **Data Augmentation:** In most cases the live training of an agent using Imitation Learning provides an incomplete set of data. Artificial data are ideally synthesized and added to the set. Methods used for augmenting data, such as blurring, cropping and altering brightness provide noise addition to the data. Cameras attached to the ego vehicle can be shifted and rotated to provide data augmentation, while left/right shifted images respond to left/right navigating commands. In this way, accumulated errors are avoided.
- **Data Diversification:** Diversification refers to noise addition on input sensory data. The desired goal is to force an unwanted behavior to the agent (i.e. force the car off the trajectory). The agent must learn to face deviations and restore the car to its original state. Correlated noise and perturbations are used to provide data diversification. This method for data generalization is dangerous to apply in a real-world environment.

- **On-Policy Learning:** This method supports alternation between the expert and a model for agent guidance. In every situation the model produces, a compromised solution is received from the expert, as s/he can override the vehicle control. DAgger experiment utilizes a module for guidance selection between the expert and a model [14]. Additionally observational imitation learning can be used to along with a set of sensors that replace the expert, to provide a solution.

**Dataset Balancing** constitutes a crucial disadvantage for Imitation Learning. In most cases this phenomenon is caused by data bias. As a result, the test-time performance along with the generalization ability underperform. Imitation learning is susceptible to bias, as the agent is exposed mostly to frequently occurring behavior. In addition, the high dimensionality of the input data parameters of an urban environment may occur in multiple combinations, which makes rare phenomenons hard to solve in live driving. As the data increases, the generalization ability of the model decreases.

Promising solutions to data inbalancing are the upsample of rare data-samples and the downsample of the most common, utilize data weighting depending on the frequency of sample occurrence, balance categorized by steering predictions, errors, or navigational commands.

**Training Instability** is another challenge taking place in Imitation Learning. Training the model may not converge to the same minimum, as network initialization and batch sizing may differ between training steps, as also the input observations depend on agent's most recent decision.

## 3.2 Reinforcement Learning

The agent is directly acting in the urban environment online, interacting with other actors, while utilizing a reward system for training. In contrast to Imitation Learning, Reinforcement Learning provides sufficient exposure to the urban environment, as the agent compromises all relevant situations, and the training signal originates from rewards claimed. On the downside, training takes up enough time as the agent must explore a massive number of cases in the environment.

The **Reward System** used in Reinforcement Learning is responsible for the training course, as it influences the agent's behavior. A simple reward scheme makes decision understanding more interpretable, while a complex reward scheme defines the desired behavior more precisely.

Applying Reinforcement Learning in the real world is a challenging task [3], as a car-agent would have to take damage to explore the environment sufficiently. In this case, a safety driver could intervene indicating the end of an episode in any dangerous deviation.

Simulators offer a solution to real-world learning [3], as the agent can be trained safely in a virtual environment. Simulators like CARLA and GTA V have been used to train Imitation and Reinforcement Learning models, providing rich urban environments, including vehicles, traffic lights, intersections, pedestrians etc.

Reidmiller applied reinforcement learning for a real self-driving car to learn steering [15]. Koutnik used the virtual environment TORCS and neuroevolution to train an agent receiving image inputs [16].

Multiple algorithms are applied on autonomous driving models in urban settings such as Deep Q Learning, Deep Deterministic Policy Gradient and Proximal Policy Optimization [3]. Policy Gradient algorithms are also a common choice.

## 3.3 Comparing Reinforcement and Imitation Learning

Imitation Learning is a practical paradigm, used in most deployed autonomous driving cars [4]. While offering behavior cloning mimicking an expert's behavior, carries major disadvantages, like the distributional shift, data bias and causal fusion. Imitation Learning can succeed in simple tasks like lane following but underperform in more complicated and rare traffic events, taking place all the time in the urban driving scene [4].

Reinforcement Learning can manage a complex task like urban driving and create strong and precise policies [4]. Additionally, its interactive learning ability resolves the distribution shift, causal confusion, and data bias issues. However, it is less data efficient and much harder to train in complex tasks [4].

Ideally, the autonomous driving system may not learn from demonstrations, as the primary goal of the autonomous driving task in an urban environment is to achieve better-than-human performance to avoid human error. In this thesis, the autonomous driving car-agent is trained with Reinforcement Learning.

Alternatively, a combination of Imitation and Reinforcement Learning offers a compromising solution, as the agent is primarily trained via Imitation Learning and then fine-tuned via Reinforcement Learning, resulting in shorter exposure time.

# 4. Reinforcement Learning

The Reinforcement Learning method has been selected as the best-fit method for the implementation of this thesis on the Urban Driving problem. It is important to highlight the features and dynamics of this method, that can bring forth in this hard and complex environment, starting from the basics.

All machine learning algorithms are designed to improve performance by learning from experience, and belong into three categories, “supervised learning, unsupervised learning, and reinforcement learning algorithms” [1]. Supervised learning algorithms intend to regress or classify data. Unsupervised algorithms are designed to estimate data density or cluster data. Reinforcement learning algorithms improve performance by interacting online with the environment and gathering rewards.

All reinforcement learning models receive data from sensors and provide data to actuators. The main goal is to maximize all cumulative rewards described by a Reward Function [1]. The agent may explore the environment and gather rewards or exploit the already existing knowledge (learnt while interacting with the environment), shaping a challenge to manage the trade-off between exploration and exploitation. Algorithms like e-greedy and SoftMax capture this trade-off.

“A **Markov Decision Process** is a form describing a sequential decision-making problem. An MDP consists of a set of states, a set of actions, a transition function, and a reward function (<S,A,T,R>). When in any state, selecting an action will result in the environment entering a new state with a transition probability, and provide a reward [1]. The stochastic policy is a mapping from the state space to a probability over the set of actions. All MDPs satisfy the Markov property, declaring that system state transitions are dependent only on the most recent state and action, not on the full history of states and actions in the decision process” [1]. If the environment holds a massive number of features which cannot be considered, then the MDP is a Partially Observable Markov Decision Process (POMDP).

$$Q_\pi(s,a) = \mathbb{E}_\pi\left\{\sum_{k=0}^{H-1}\gamma^k r_{k+1} \mid s_0 = s, a_0 = a\right\}. \qquad \pi^* = \underset{\pi}{\operatorname{argmax}} \underbrace{\mathbb{E}_\pi\left\{\sum_{k=0}^{H-1}\gamma^k r_{k+1} \mid s_0 = s\right\}}_{:=V_\pi(s)},$$

**Equation 1: Q function (left) and optimal Policy (right).**

“The goal is to find the **Optimal Policy** (Equation 1), which results in the highest expected sum of discounted rewards” [1].

The **Value Function** highlights the current value of the model-policy (Equation 1). This function can alter this value favoring the short-term or long-term rewards, through the gamma parameter, and “depends on the horizon H that indicates the number of steps before the episode termination. The horizon can be infinite or dynamic depending on the goal state” [1].

Solving a Reinforcement Learning task means finding the Optimal Policy.

## 4.1 Value-Based Methods

Value based methods intend to optimize the value function. Q Learning is one of the most well-known value-based algorithms [1]. It is influenced by the Temporal Differences algorithm and converges near optimal on enough samples. The Q Learning is described in (Equation 2). Factor α is the learning rate, while γ is the same as above. The Q function can be initialized with arbitrary values, while initialization may be optimistic or pessimistic.

“A Deep Neural Network can be interpreted in Q learning, shaping Deep Q Learning, which utilizes experience replay technique to break the correlation between successive samples and improve sample efficiency” [1].

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)],$$

**Equation 2: Q-values update [1]**

$$\theta \leftarrow \theta + \alpha \gamma^t g \nabla \log \pi_\theta(a|s),$$

**Equation 3: Policy update [1]**

## 4.2 Policy-Based Methods

“Both Value-Based and Policy-Based methods propose actions and evaluate the resulting behavior. Though value-based methods focus on searching for the optimal cumulative reward, policy-based methods aim to evaluate the optimal policy directly, while value may not be calculated” [1]. Policy is often expressed through a neural network. A policy-based algorithm utilizes a stochastic policy to learn a deterministic policy, resulting in providing enough exploration. Equation 4 presents the way policy parameters and updated. In the case of whole episodes, no estimator is utilized.

Many problems require the use of continuous action spaces [1]. Deterministic Policy Gradient (DPG) can use continuous actions spaces, as all policy-based algorithms can. In addition, DPG only utilizes the state space, making it ideal for problems with massive action spaces. Trust Region Policy Optimization (TRPO) prevents the deviation between the updated and the old policy, as the parameter by limiting policy gradient updates, as measured by the Kullback-Leibler (KL) divergence [1]. Proximal Policy Optimization (PPO) offers parameter clipping, also preventing deviation between policies, while it has simpler implementation and better sample complexity.

## 4.3 Actor-Critic Methods

A hybrid combination of value-based and policy-based methods, where the policy selects an action and value criticizes the decision made. This evaluation takes place based on the expectations of the environment. Two networks constitute this category, that estimate policy and value.

Deep Direct Policy Gradient (DDPG) is an actor critic method, integrating noise on the selected action for sufficient exploration. In addition, it is an extension of the Direct Policy Gradient, that can manage large action spaces of high dimensions. The A3C algorithm uses asynchronous gradient descent, utilizing parallel training of instances, reducing data correlation.

The **Advantage** module is used to determine good actions while reducing variance and stabilizing the training. **Entropy** is another module like advantage, utilized from energy-based methods and specifically the Soft Actor-Critic method. Sufficient exploration must be performed, in a balance with knowledge exploitation, so most uncertain paths get discovered.

### 4.4 Model-based, Model-free, On-policy and Off-policy methods

Model-based methods store knowledge of the environment while interacting [1]. Learning the dynamics of the environment or applying limited interaction can favor model cost and safety, resulting in fewer but more costly interactions. Storing long-term and short-term memories of the environment is a general feature of model-based methods. Memories refer to model features and parameters providing efficient predictions. In model-free methods no such knowledge is stored, as the model learns the Markov Decision Problem through the value function directly.

A method can be considered as on-policy or off-policy depending on the way it generates trajectories and updates its policy [1]. On-policy methods, like SARSA, use a stable control policy to generate trajectories of states, while updating the same policy. Off-policy methods, like Q-learning, generate actions through the behavioral policy, which can be stochastic or non-deterministic to sample all available actions, while updating the improving target policy.

### 4.5 Deep Reinforcement Learning

Storing learned features requires information about all state-action pairs, bringing an exponential growth problem, known as the curse of dimensionality [1]. Discretizing continuous state and action spaces, to be method-compatible, increases this problem, depending on the discretized values count.

Function approximation offers a solution to this problem, with the practical use of neural networks [1]. Deep Reinforcement Learning refers to methods using neural networks as function approximators. Most common DRL methods utilize a replay buffer for selecting the next action, avoiding the forward operation of the neural network. Dueling networks have been implemented in DRL approximating value function and the advantage module.

DRQN combines Deep Reinforcement Learning and Long Short-Term Memory networks [1], commonly used for predicting the velocity of objects. DRQN is shown to generalize better than DQN. Deep Reinforcement Learning methods have reached human level performance in games like Atari and Go [1].

### 4.6 Reward Scheme

The Reinforcement Learning agent aims to learn an optimal policy through maximizing accumulated rewards, utilizing a reward function [1]. Depending on the environment, sparse and delayed rewards may be received, resulting in slowing down the training. An agent that takes a long time to learn its goal may be receiving delayed rewards. The main idea is to reward all behavior that leads to the desired behavior. Though that is not always true. In [17] a bicycle was rewarded for turning in a circle to stay upright rather than reach its goal.

## 4.7 Applying in Autonomous Driving

Reinforcement Learning is utilized for many autonomous driving tasks, beyond end-to-end training, like controller optimization, path planning, complex navigation tasks and driving policies, scenario-based learning, predicting the intent of other actors, ensure safety and risk estimation [1].

Defining a detailed **State Space** is of crucial importance. Most common state space schemes include position, heading, velocity of the agent vehicle, actors inside agents' vision, lane number, path curvature, future and past route, time to crash using longitudinal information, traffic laws, signal locations, etc. [1]. Compressed information reduces complexity and results in faster estimations.

The **Action Space** consists of information driven to the actuators [1]. Mainly, it contains steering, braking, accelerating, flashing car lights, changing gears, etc. Belonging in a continuous action space, these parameters may be discretized for compatibility with the method used. The count of discretized variables plays a massive role in the agent's behavior. A multitudinous discretization can lead to jerky movements and unstable trajectories, while a subbituminous discretization may lead to expensive tasks, shaping a trade-off between the two. Methods that can handle continuous action and state spaces, like DPG and PPO, learn the policy directly.

A **Reward Scheme** may contain multiple parameters such as distance to destination, velocity of the self-driving car, standstill time, collision with other actors, infraction on sidewalks, keeping in lane, following traffic rules, etc. [1]. The comfort and stability of the self-driving car are significant as they are directly affected from steering, breaking, and accelerating smoothness.

After displaying the core features of Reinforcement Learning, it is significant to utilize all these dynamics to design a model that can cope with the complexity of an urban environment, to achieve better-than-human performance.

# 5. Modern Approaches on Autonomous Driving in Urban Settings

Having the knowledge of the core features in Reinforcement Learning, as also highlighting the potential to encounter the Urban Driving problem, this section highlights the action course of current approaches attempting to solve this problem. Many of them achieved remarkable results.

## 5.1 End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances

Authors support that there is no Reinforcement algorithm capable of handling such a challenging task as urban driving [5]. In their experiment they incorporate a technic named implicit affordances that can improve data efficiency and interpretability of Reinforcement Learning algorithms.

The model consists of two parts. Initially, an encoder backbone creates the affordances, which are fed to the Reinforcement Learning End-to-End subsystem. The authors utilized

the value-based and model-based algorithm Rainbow IQN ApeX, which holds state-of-the-art results in the Atari game. The dueling network is removed from the initial algorithm and ran on multiple distributed instances of CARLA simulator, resulting in faster results and additional exploration.

The reward system was based on waypoint API of the carla simulator, which provides the optimal trajectories as operated by an expert driver and different choices of navigation. The reward function was associated with the speed, position, and rotation of the ego vehicle (Image 5 and Image 6).

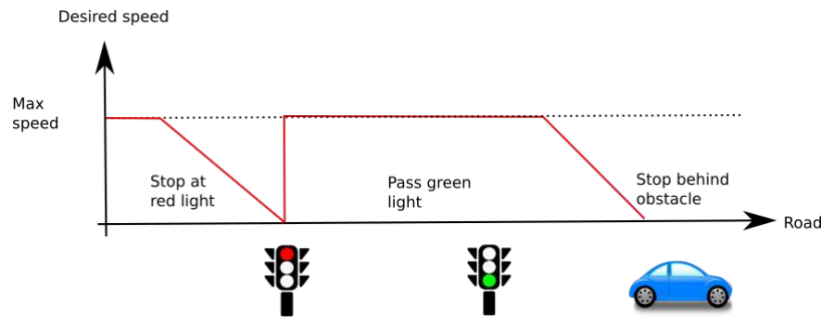


**Image 5: Optimal Speed**

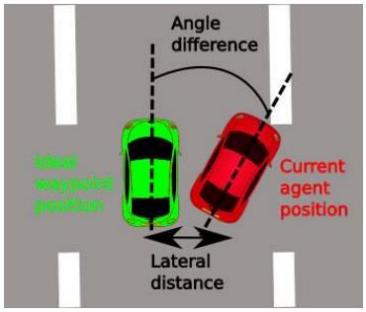


**Image 6: Heading Deviation**

More specifically, as the speed of the vehicle deviated from optimal speed the reward was reduced linearly [0,1]. The optimal speed is determined by the distance of the ego vehicle from preceding obstacles, traffic lights and environment maximum speed. Additionally, as the position of the vehicle deviated from the center of the lane the agent received a negative reward of [-1,0] inversely proportional to that distance. Finally, as the vehicle heading deviated from the optimal heading the agent again received a negative reward of [-1,0] inversely proportional to that deviation.

The architecture of the model (Image 7) was constituted by an 18-layered Convolution Neural Network, owning 30 million parameters, and trained with 20 million frames. The input was the concatenation of 4 consecutive frames, as single frames harden the traffic light detection. Resnet-18 was used utilizing residual connections and batch norm. Finally, a conditional network was integrated as a last step, ending to six actions about following road, turning left, turning right, keeping straight, changing left lane and change right lane.

The selected algorithm is not compatible with continuous state and action spaces, like steering, braking, and accelerating, resulting in their discretization.

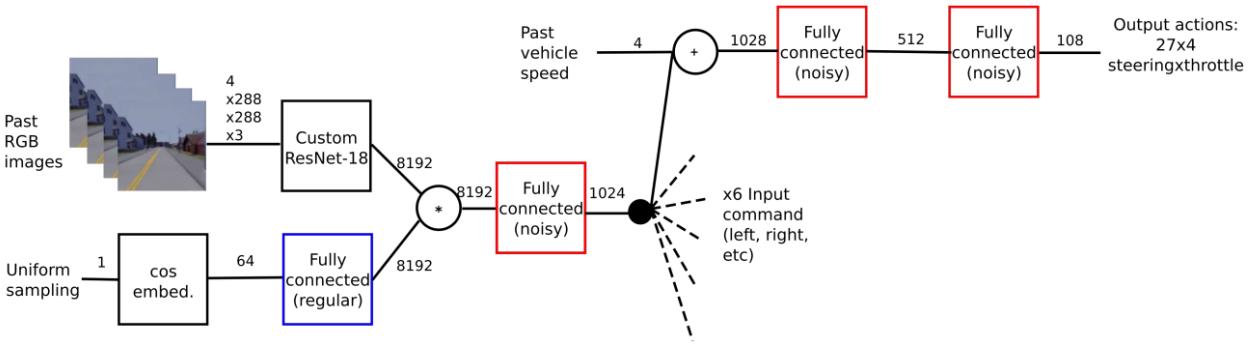


**Image 7: The network architecture [5]**

Acceleration and braking was discretized to three values, while steering was discretized to five, nine and twenty-seven values. Bagging and averaging were deployed for consecutive steps by the networks to provide a smooth transition between actions.

The environment of the experiment was carla town 5, a crowded urban environment with dynamic weather enabled.

The evaluation utilized as metrics the successful intersections (Inters), traffic lights crossed (TL), and pedestrians overpassed (Ped). The experiments evaluated the impact of different reward parameters on the agent's behavior (Table 1). Experiments without utilizing optimal speed or heading deviation, as also different discretization count in steering values took place. Agents not utilizing optimal speed failed to brake on most Traffic Lights and Pedestrian Crossings, acquiring the worst performance, as 70% of the traffic lights were run and 60% of the Pedestrian Crossings were unsuccessful. Utilizing the derivative of steering led to more oscillations. Finally, discretization of nine or twenty-seven values had no significant impact on the results.

The best reward scheme provided the remarkable results of 69.1% intersection, 97.6% traffic lights and 76% pedestrian crossings successfully crossed. The features of optimal speed, heading deviation and distance from the lane center provide a precise description of a self-driving car’s desired behavior in an urban environment and assist the model to achieve optimal performance.

The implementation of this thesis deploys all three of the reward scheme parameters.

**Table 1: Metric results on reward and discretization parameters. [5]**

| Input/output | Inters. | TL | Ped. | Osc. |
|---|---|---|---|---|
| Constant desired speed | 50.3% | 31% | 42% | 1.51° |
| No angle reward | 64.7% | 99% | 77.7% | 1.39° |
| 27 steering values (derivative) | 64.5% | 98.7% | 85.1% | 1.64° |
| 9 steering values (absolute) | 74.4% | 98.5% | 84.6% | 0.88° |
| 27 steering values (absolute) | 75.8% | 98.3% | 81.6% | 0.84° |

## 5.2 ModEL: A Modularized End-to-end Reinforcement Learning Framework for Autonomous Driving

The ModeL experiment, implemented by Haoyi Niu [4], consists of three modules: perception, Planning and Control modules [4] (Image 8).

The Perception module constitutes a supervised encoder receiving RGB images and provides semantically segmentated representations. The primary target of this module is to bridge the visual gap between simulation and reality. The Segmentated image consists of three categories, drivable (i.e. current road), partially drivable (i.e. requires lane change) and non-drivable zones.

The planning module uses End-to-End Reinforcement Learning, while utilizing a specially designed domain generalization scheme. Specifically, the module receives the semantic segmentated images from perception as states and produces actions as commands on steering and accelerating.

The reward function constitutes of four parameters. The agent receives greater rewards as his velocity is in a specific speed range and drives in the center of the lane, and a negative reward as he deviates from the optimal heading, collides with other obstacles, and crosses unwanted lanes.

The Reinforcement Learning algorithm used to train the model Is Soft Actor-Critic. The algorithm is executed in multiple distributed instances to integrate data efficiency and sufficient exploration.

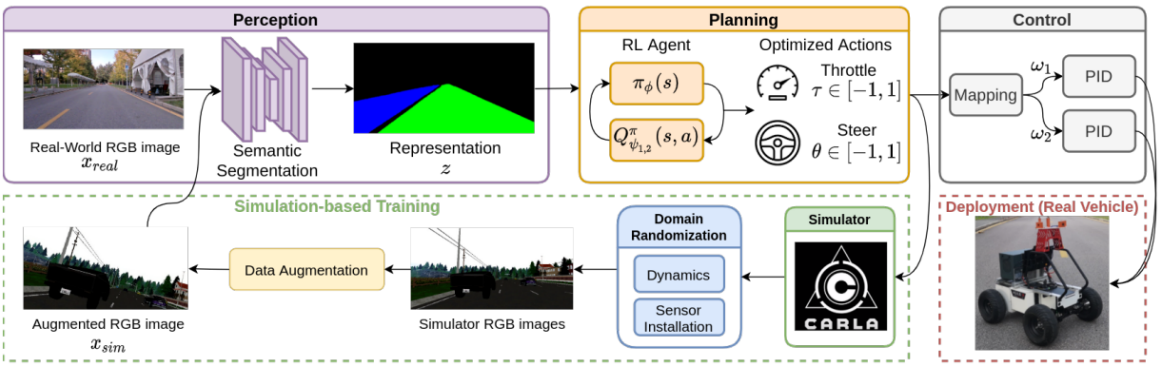


**Image 8: ModEL architecture [4]**

The model generalizes well, as it utilizes diverse simulation properties (i.e. dynamic weather), random camera spawn relevant to initial position, diverse vehicle physical properties (i.e. mass, size, height), data augmentation via camera shifting and rotating, resulting in data efficiency and overall generalization. The Control module smooths the

action provided from the Planning module into curves and extracts the result to the low-level actuators.

The metrics used for evaluation are meters per intervention, success rate (successful meters until intervention regarding total experiment meters), standard deviation of velocity and standard deviation of angle heading. Overall, the model fails to generalize well in simulation, resulting in 449.4 meters per intervention in training and 332.6 in testing, in contrast to the real-world. Applying the model in the real-world results in a high success rate and meters per intervention and is capable of lane following, turning, and avoiding dynamic obstacles.

Table 2 highlights the results compared to the Modularized IL implementation [18]. Overall, the model brings remarkable results, crossing many meters without intervention, keeping a low heading deviation and velocity deviation. Again, the effectiveness of the reward scheme parameters (also used in the previous implementation) is established and makes up an incredibly broad selection of parameters for the thesis's implementation.

**Table 2: Metric Results for ModEL deployed in the real-world. [4]**

| **Real World Task** | | | **Framework** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | **Modularized IL** | | | | **Modularized RL** (ModEL) | | | |
| Road Topology | Obstacle Setup | Lighting Condition | MPI ($m$) | SR (%) | Std[$\theta$] (°) | Std[$v$] ($m \cdot s^{-1}$) | MPI ($m$) | SR (%) | Std[$\theta$] (°) | Std[$v$] ($m \cdot s^{-1}$) |
| Straight | ✗ | Day | 92.1 | 48.9 | **1.30** | 0.30 | **>1163.5** | **100.0** | 1.72 | **0.19** |
| | ✗ | Night | 187.1 | 49.0 | **1.67** | **0.18** | **>1304.5** | **100.0** | 1.89 | 0.20 |
| | ✓ | Day | 4.1 | 16.7 | **1.25** | 0.37 | **34.5** | **75.0** | 2.59 | **0.30** |
| Turn | ✗ | Day | 7.2 | 53.2 | **3.03** | 0.32 | **>214.9** | **100.0** | 3.81 | **0.23** |

## 5.3 Autonomous Driving using Deep Reinforcement Learning in Urban Environment

The main goal of the authors is to replace accident prone human drivers and import comfort and safety to autonomous vehicles that can perform better than humans in urban environments [6]. The model was trained in a game environment implemented by Unity.

The Reinforcement Learning algorithm used for training is Deep Q Learning, utilizing an experience replay buffer, a target network, clipping rewards and skipping frames. The input consists of a fusion of a monocular camera and lidar sensors.

The architecture of the network consists of 3 fully connected CNN layers, followed by 4 dense layers, followed by a flattened layer, followed by a dense layer, followed by a final pooling layer [6].

The model (Image 9) can produce the actions(rewards) keep straight (speed/5), turn left (–0.6), turn right (–0.2), accelerate (1), brake (-0.4). In case of collision the agent receives the negative reward -5. The desired result is acceleration of the ego-vehicle when the road is empty and deceleration otherwise.

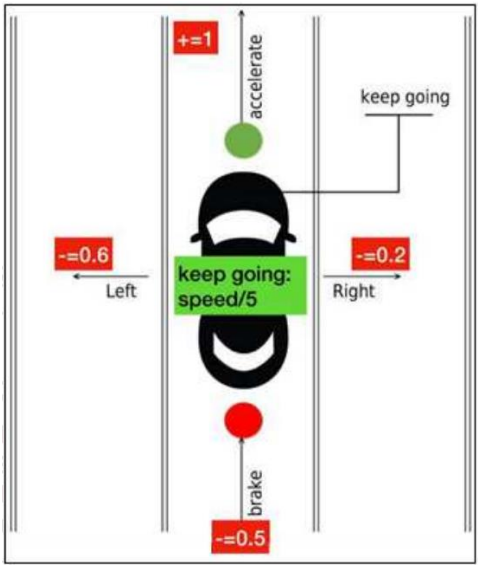


**Image 9: Actions and reward scheme of the agent. [6]**

The approaches presented in this section were deployed in recent years and provided remarkable results. Some of the key features for their success were the exceptional design of the reward scheme, distributed learning in training, data augmentation, reward clipping, dynamic weather, random physical properties and camera spawn positions, smooth actions by averaging the discretized action values, etc. This thesis utilizes features of all these approaches, as they appear to be highly effective in resolving the autonomous driving in urban environment problem.

# 6. Implementation

Following the introduction of all the appropriate tools required to implement an efficient model that can resolve the autonomous driving problem in urban settings, as well as displayed the key features of the most successful approaches in recent years, this section presents the thesis implementation along with the way that these features were utilized.

Undeniably, the high dimensionality of an urban driving environment makes it one of the hardest problems in the autonomous driving field. Regarding its complexity through the multiple parameters of its indeterministic environment, an optimal solution for the real-world is hard to obtain. The approaches introduced in the previous sections display many optimization technics, including dealing with learning, modeling, optimizing data, sampling efficiency and training time.

This thesis presents a method of confronting the urban driving problem, combining multiple promising features utilized in these implementations that brought optimal results and have been considered remarkable, intending to produce a self-driving car that can successfully navigate through an urban environment.

## 6.1 Model Decisions

This section highlights the decisions made while designing the model.

An **End-To-End Approach** was preferred, as modular approaches offer limited results regarding scene understanding, while cannot consider all driving scenarios, in such a complicated task as the autonomous driving in urban settings. Architectures utilizing neural networks solved numerous problems with similar complexity as urban driving, which is encouraging.

Regarding learning, **Reinforcement Learning** was chosen over Imitation Learning, as Imitation Learning provides non satisfying results while operating on environments consisting of rare and complicated scenarios that frequently occur, like an urban scene.

Regarding the vehicle **Sensors**, the self-driving car brings three semantic segmentation cameras, facing straight, vertically left and vertically right. It is of high importance to avoid the use of non-cost-effective sensors such as lidars, as they receive almost equal spatial information when multiple cameras are deployed. Providing an almost surrounding view can assist learning and optimize the agent's behavior during intersections and lane changes. Additionally, the agent utilizes multiple past frames on each step, which assists learning temporal information, optimizing steering on intersections and lane changes. Furthermore, the use of semantic segmentation cameras intends to reduce the simulation-reality-gap, as the intermediate semantic representation can be originated by a simulator or the real-world.

Regarding the **Reward Scheme**, the features of optimal speed, heading deviation and distance from the lane center utilized, provide a precise description of a self-driving car’s desired behavior in an urban environment, and assist the model to achieve optimal performance. A simple reward function makes decision understanding more interpretable, but a complex reward scheme defines the desired behavior more precisely.

The use of throttle, braking, and steering can optimally be represented through continuous action spaces. Using a value-based algorithm requires the discretization of these action spaces. A multitudinous discretization can lead to jerky car movements and unstable trajectories, while a subbituminous discretization may lead to expensive tasks and increase the curse of dimensionality problem, since information of all action-state pairs are utilized. The thesis implementation eliminates the discretization problem,

utilizing the policy-based **Proximal Policy Optimization** algorithm, which can manage continuous state and action spaces optimally, and is shown to provide optimal results. Additionally, PPO has a simpler implementation and better sample complexity than other policy-based algorithms like Trust Region Policy Optimization.

The model was deployed in CARLA simulator, which provides a dynamic environment and all necessary tools for a complete autonomous driving simulation in urban environments. Additionally, this thesis utilized the integration of CARLA simulator with Ray, a framework for building distributed applications including the reinforcement learning library RLLib, using a tutorial-project emerging this integration [8,9], as base project. The model was deployed on multiple parallel instances on distributed machines, acquiring additional agent exploration, more data-efficient and time-efficient training. All experiments were conducted on the infrastructure of Amazon Web Services. Tensorboard was used for the visualization and the analysis of all metric results.

The next sections present the model architecture in detail.

## 6.2 Simulators, Frameworks, and Infrastructure

The implementation of this thesis is built upon the CARLA simulator, making use of the integration of Ray framework in CARLA. The final model was trained on AWS infrastructure.

### 6.2.1 CARLA simulator

CARLA is an opensource simulator designed for autonomous driving deployment. CARLA API is ideal for developing, training, and validating autonomous driving projects as it offers a plethora of choices for interaction with urban driving environments [10]. It includes a blueprint library consisting of a massive number of vehicles, pedestrians, street signs, lights, and markings. Additionally, CARLA owns sensors, including cameras that provide rgb frames, depth estimation, semantic segmentation, optical flow, lidars, radars, collision, gnss, imu, lane invasion detectors etc.

Multiple buildings, vegetation choices, illumination regimes, environmental and atmospheric conditions are included. All these assets combined develop a wide variety of urban layouts.

CARLA provides support with AWS infrastructure, simulator carSIM, multi-physics simulator Chrono, map creation through mathWorks, OpenDrive roads compatibility, RSS road safety framework, robotic operating system API, scene creation through scenic. The Waypoint API in OpenDrive is also a powerful tool generating waypoints of vehicle optimal road position.

### 6.2.2 Ray/RLLib Library

Ray is an opensource framework for scaling Artificial Intelligence python applications [11]. It is deployed for parallel processing and creation of distributed systems. It includes highly scalable libraries for fields like Reinforcement Learning. It offers an integration with Kubernetes, AWS, Azure and GCP, while it can utilize multiple machine learning workloads nodes and GPUs.

Ray offers production level highly distributed Reinforcement Learning workloads, making use of externally connected parallel simulators. The RLLib library has been deployed in

many scientific fields, such as climate control, industrial control, manufacturing and logistics, finance, gaming, automobile, robotics etc [11].

The implementation used the Proximal Policy Optimization algorithm included in RLLib. The algorithm supports multiple Stochastic Gradient Descent passes of the same batch, while utilizes a multi-GPU optimizer and multiple worker experience collection.

The **Ray Dashboard** is a web-based board providing monitor and debugging of the distributed application. It analyses, monitors and visualizes the status of the application and resource utilization of the logical and physical components. It can locate logs and errors, utilize CPU and memory usage, and monitor the job task progress.

The thesis implementation applied distributed reinforcement learning through Ray on five distributed AWS machines, utilizing autoscaling.

### 6.2.3 Amazon Web Services

The implementation training took place on AWS cloud computing centers. “AWS provides cloud services from storage, databases and computing to machine learning, artificial intelligence, data lakes, analytics, internet of things, while establishing security” [19]. It provides on-demand delivery IT resources with agility, elasticity, and available scaling. In general, these services can be categorized as “Infrastructure as a service, Platform as a service and Software as a service” [20].

The AWS API can create and manage from users, roles, groups, permissions to instances, volumes connected, snapshots etc.

After configuring the boto3 environment, a Deep Learning AMI was installed on the initial CARLA EC2 instance. This implementation was deployed on at most five machines, including one GPU, eight virtual CPUs and thirty-two gigabytes of memory each, through autoscaling.

## 6.3 Software Architecture

This thesis utilizes the project implemented for highlighting the CARLA-Ray integration, as a baseline project [8,9]. The implementation consists of three directories.

- **rllib_integration:** Includes all information relevant to the CARLA infrastructure, server, and client settings along with some basic instructions regarding the training.
- **aws:** Includes files relevant to the AWS API handling, such as image and instance creation, start, stop, files sync etc. In general, it establishes functionalities for the easy manipulation of the EC2 instances used.
- **ppo_implementation:** Contains all files relevant to the training of the experiment.

The implementation core is defined in three sections:

- **Experiment Class:** Defines the training experiment. The BaseExperiment class (base_experiment.py) is inherited by PPOExperiment class (ppo_experiment.py), overriding the methods that define the training parameters, related to the state, actions, observations and reward of the training. The carla_env.py file integrates Ray in a Gym environment.

- **Environment Configuration:** Utilizes a ppo_config.yaml file to set the configurations of the environment. Provides most server-client settings, like server timeout, map quality. Sets special configuration variables, like town conditions, spawn points of ego-vehicle and sensors, sensor variables etc. Defines Ray-specific settings, the training model and its training parameters. The ppo_autoscaler.py sets the necessary variables handling the autoscaling. The carla_core.py file sets the necessary functions handling the CARLA environment setup.
- **Training and Inference script:** Includes a training (ppo_train.py) and inference (ppo_inference_ray.py) script, defining the training and inference procedure. It is dependent on Ray API.

The ppo_callbacks file saves metric data, while the ppo_trainer file enables data-and-model checkpointing for every step. The dynamic_weather file enables dynamic changes in weather during training.

### 6.4 Observations, Actions and Reward Shaping

Regarding the **Input Modalities** of the implementation, three semantic segmentation cameras are attached to the agent, heading straight (0 degrees), left (minus 90 degrees), and right (plus 90 degrees) from the vehicle. Side cameras assist not only on safety from collisions, but for better scene understanding and road turn recognition.

The semantic segmentation module of the cameras plays a massive role in reducing the **simulation-reality gap** of the self-driving car operation. It is an intermediate encoded representation of the scene as captured by the camera, which can be easily acquired from real-world frames too. Additionally, being in a human readable form provides cleaner and more precise information of the observation.

The agent receives three semantic segmentated frames as sensory input of [300x300x3] dimension, in each step. These frames are being concatenated into a [300x300x9] group-frame. The agent also keeps available the [300x300x9] group-frames from the last two steps. The **Observation**, which is fed to the agent, is the concatenation of the current and the last two group-frames. This implies that the **Observation Space** is [300x300x27], while can get values in [0,255].

Feeding multiple group-frames from recent steps to the model training provides a correlation of group-frames with acceleration, velocity, steering, braking, as well as recognition of abrupt-jerky or smooth car movements. Additionally, it presents the driving behavior of the agent i.e. an agent restoring the cars position to its original after a false decision. The agent is optimized utilizing temporal alternation of group-frames.

As an **Action**, the agent can use the throttle, steer, and brake. The **Action Space** is [[0.0,1.0] , [-1.0,1.0] , [0.0,1.0]], representing throttle, steering, and braking, respectively. This constitutes a continuous action space, which PPO can effectively encounter.

The **Reward Shaping** utilized the CARLA Waypoint API that can relate the position and heading of the ego-vehicle with the optimal position and heading [2]. While a reward shaping of a few rules is more easily understandable by the agent, it provides limited performance. A reward shaping of multiple rules has been deployed, to better define and communicate the optimal behavior to the agent. It includes the following features:

- **Reward if going forward:** The agent is rewarded according to the distance crossed compared with previous step.

- **Reward if going faster:** The agent is rewarded by +0.05*delta_speed if has higher velocity (in speed-limits) than the last step.
- **Reward if reached max time:** The agent is rewarded by +100 if reached the episode max time. Indicates that the episode did not stop by collision, or idle time.
- **Penalize correct lane heading deviation:** The agent is penalized by [0,1] inversely proposed to the deviation.
- **Penalize going backwards:** The agent is penalized by –0.5 if going backwards (i.e. being in the opposite lane).
- **Penalize optimal velocity deviation:** The optimal velocity of the agent if formed by the maximum speed-limit, the distance to red traffic light and the distance to preceding vehicle. The agent must not exceed the maximum speed-limit, while should linearly decelerate and stop one meter behind the preceding vehicle and the red traffic light. The agent is penalized by -abs(hero_velocity-optimal_speed) / max_speed+1
- **Penalize distance from the center of the lane:** The agent gets penalized by [0,1] inversely proposed to the distance from the lane center.
- **Penalize if not inside the lane:** The agent gets penalized –0.5 if not inside the lane.
- **Penalize not restoring driver heading:** The agent gets penalized –0.05 if does not restore the ego-vehicle to the optimal heading.
- **Penalize collision:** The agent gets penalized –100 if collides with any actor or obstacle. The episode is terminated.
- **Penalize max idle time:** The agent gets penalized –100 if remains stationary for multiple time steps. The episode is terminated.

### 6.5 Proximal Policy Optimization

This section analyses the Proximal Policy Optimization algorithm, which was used to train the thesis implementation model.

Proximal Policy Optimization is a model-free, on-policy Reinforcement Learning algorithm [7]. It alternates between sampling data and optimizing a surrogate policy function through stochastic gradient ascent [7]. Optimization occurs for multiple epochs on minibatches of data. It applies multiple benefits from the Trust Region Policy Optimization algorithm, while its more simple, general and has better sample complexity [7]. The clipping parameters version of PPO brings optimal results and works better than most other algorithms on continuous control tasks.

Most **Policy Gradient Methods** calculate an estimator, utilizing Stochastic Gradient Ascent and an Advantage module [7]. This estimator is obtained by differentiating an objective loss function L. Multiple optimization steps are applied to L function for the same trajectory of states. This leads to destructively large policy updates.

To solve this problem, **Trust Region Policy Optimization** maximizes this objective function subject to a constraint on the policy update size [7]. A subtraction of a coefficient (including this constraint) from the objective function could be used instead, in a penalty form. Utilizing a constraint is preferred, as it is hard to find an optimal factor going along with the penalty and working for multiple problems, as characteristics change on learning.

**Proximal Policy Optimization** ensures minor policy changes over objective function optimization, by utilizing the e parameter which clips the objective function in [1-e, 1+e] space [7]. Then, chooses the minimum of the clipped and unclipped objective function, as a pessimistic bound. As the objective function is improved, clipping is ignored, while as the objective function gets worse, clipping is enabled.

The algorithm makes use of a value function to calculate an advantage estimation function. A Neural Network is utilized sharing variables between policy and value functions. In addition, entropy may be used to ensure sufficient exploration [7]. As an option, the objective function may be optimized every T steps (much less than horizon length), enabling the use of data collection by parallel actors.

Clipping in PPO collects most cumulative rewards, compared with penalty and fixed KL, while outperforms A2C, A2C+Trust Region, CEM, VPG adaptive and TRPO in most continuous control environments [7].

Proximal Policy Optimization is an ideal choice for Reinforcement Learning on robotics tasks, as it is an on-policy algorithm that can handle continuous action spaces, frequently required in robotics. Firmly, it restricts large updates on policy optimization, which could lead to misbehaving policies, resulting in a non-deviating policy between steps, a desired behavior for a self-driving car in urban settings. Additionally, it provides sufficient exploration, regarding the entropy module and the distributed training of the agent.

### 6.6 Model Architecture

The model architecture of the implementation consists of a multilayer Convolutional Neural Network. A CNN is commonly used in computer vision, image, and video processing, making up an ideal selection for the autonomous driving task in urban settings, which develops perception through camera data. In this implementation many combinations of a CNN model architecture have been deployed, with the most promising being the following:

- **[output, kernel, stride]**
- **[16, [5, 5], 4]**
- **[32, [5, 5], 2]**
- **[32, [5, 5], 2]**
- **[64, [5, 5], 1]**
- **[64, [5, 5], 2]**
- **[128, [5, 5], 2]**
- **[512, [5, 5], 1]**

Models with more layers failed to operate on the infrastructure, while fewer layers brought deficient performance.

The model is optimized using the Adam optimizer.

A critic value-function is utilized, as also an Advantage estimator, which have been analyzed in the previous chapter.

The clip parameter epsilon is set to 0.3.

Stochastic Sampling is used for exploration, while the horizon is set to infinite.

## 6.7 Model Training

After setting up AWS, installing the required libraries, performing experiments, acquiring the most promising model architecture, a full model training was initiated. The distributed training of the agent, using Ray framework, enabled gathering sensor data from multiple parallel environments, even on the same machine, ran in autoscaling on AWS.

While monitoring the utilities and resources through the Ray dashboard, many workers crashed because of infrastructure unavailability, maximum memory reached etc. The automatic restart of a worker, when available, was set through the Ray API.

Utilizing the checkpointing system of Ray, the metric and model data were stored in each step. The best model was reached, after which performance was reduced.

# 7. Model Evaluation

This section presents the evaluation of the agent and highlights the operation of the self-driving car inside an urban environment.

For the metric evaluation, Ray offers the choice of storing metric values that arise during the agents training, through the checkpointing system. The stored metrics that this thesis analyses are the mean episode length, rewards provided to the agent, policy loss, and the metric of heading deviation. These metrics are of high importance to define the agent's updated behavior on the urban environment, compared with the initial model, as also the overall effectiveness of the training.

The metric values are visualized and analyzed through Tensorboard, since it is fully compatible with the Ray checkpointing system.

## 7.1 Model Inference

Following the training of the model, the agent is deployed on the urban environment for inference. The agent is moving at slow speed compared to the town's speed limit. While navigating, the agent avoids some easy scenarios of collision. The agent rarely collides with preceding vehicles or other lane vehicles. Most collisions occur in intersections and turns. Steering in turns seems hard for the agent as he keeps changing lanes and crashing. The agent will stop at some red traffic lights, with sudden changes being hard to detect. While operating, the agent will rarely remain stationary until an episode end is reached. On straight roads the vehicle heading is good, and the vehicle does not steer, while mostly remained in the center of the lane. When the heading deviates it rarely gets restored. The agent makes minor jerky moves occasionally.

In general, the agent has improved behavior, compared to the initial model. The agent is capable of coping with some easy navigation cases in an urban environment, completing most episodes successfully. He manages to stay in the center of the lane, keep the optimal heading and avoid other vehicles in many easy cases.

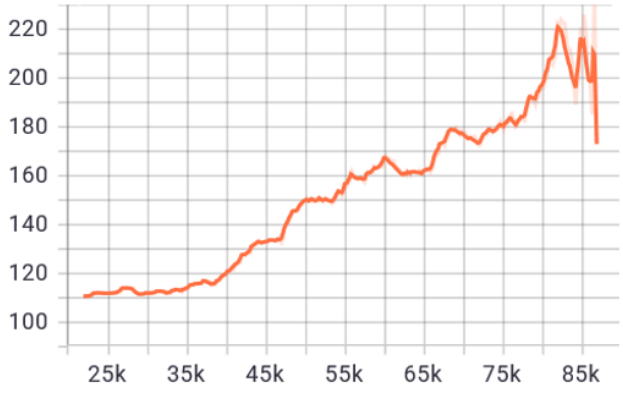


**Figure 1: Mean Episode Length by Timesteps**

## 7.2 Episode Length

Figure 1 represents the mean length of episode in timesteps, during different training steps. Typically, an episode terminates when maximum episode time is reached, or when

the ego-vehicle is colliding with another actor/obstacle, or when the ego-vehicle stays stationary for a long time. An increase in the episode length highlights the reduction of collisions of the ego-vehicle over time, along with the reduction of times being unacceptably idle. The count of maximum-time-reached has increased, as the mean episode length increases, especially during the end of training, where the value of graph does not increase.

This plot indicates that the self-driving car has improved its driving behavior over time, being able to navigate through the environment with reduced collisions and idle times. The agent learns to avoid other actors, breaking behind obstacles, making some successful turns, and probably restoring itself to the proper heading, to avoid collision.

## 7.3 Episode Reward

Figures 2 illustrate the mean reward acquired by the agent regarding the time steps of training. Mean reward is increased over the time steps, leading to better driving behavior of the agent in the urban environment. A new maximum ceiling is met every 5k timesteps on the maximum reward, showing that the agent gets less penalized and more rewarded for his actions. Particularly important is the increase in episode length, as the agent receives consecutive minor rewards with time.

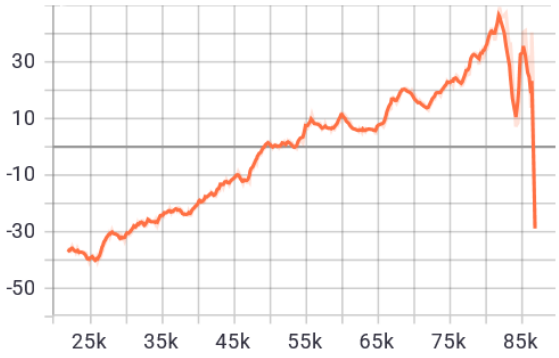


**Figure 2: Mean Episode Reward by Timesteps**

It is obvious that the agent converges to optimal behavior by time. The agent learns to avoid receiving big penalties initially, such as idle time and collision, since their influence on the mean reward is big, regarding their penalty at 45k-50k steps.

Concluding, the agent learns to avoid collisions with other actors and obstacles, while most probably collides on steering in turns, as keeping straight is more frequent behavior than turning in an intersection. Overall, the agent's behavior has improved.

## 7.4 Policy Loss

Figures 3 and 4 illustrate the policy loss over multiple time steps. The policy loss gradually decreases, resulting in a minor but stable policy improvement. Proximal Policy Optimization ensures minor changes from older policies to newer policies, avoiding abrupt changes in the plot results, excluding the policy loss near 70k time steps.

The agent tends to make better decisions over time, collecting high cumulative rewards, resulting in a diminishing loss function.

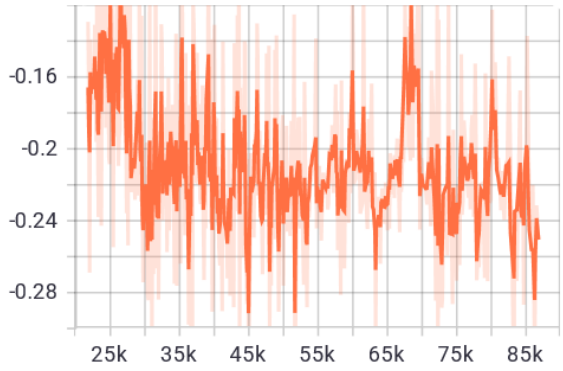


**Figure 3: Policy Loss by Timesteps**

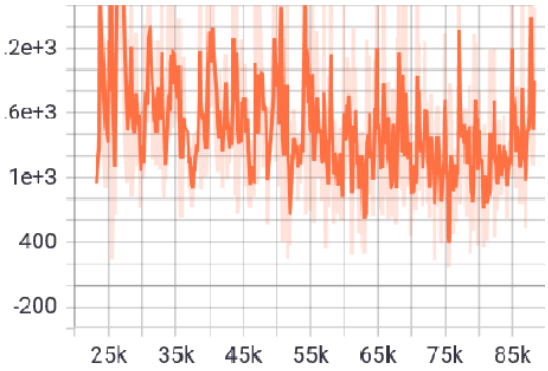


**Figure 4: Total Loss by Timesteps**

## 7.5 Heading Deviation

Figures 5, 6 and 7 present the heading deviation of the ego-vehicle by timesteps. All metrics, maximum, mean and minimum heading deviations are reduced over time, as the agent learns to head the optimal way.

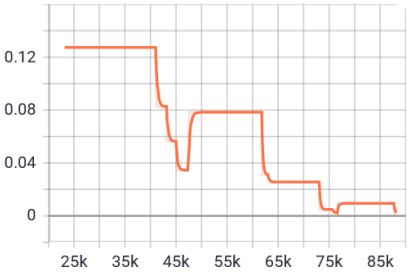


**Figure 5: Max Heading Deviation by Timesteps**

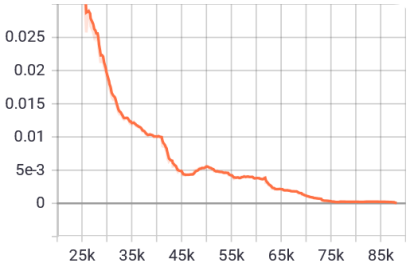


**Figure 6: Mean Heading Deviation by Timesteps**

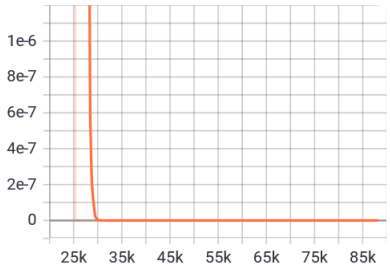


**Figure 7: Min Heading Deviation by Timesteps**

Result is the heading deviation converging to zero, which means the agent learns to restore the ego-vehicle to the original position heading, from deviating. As training continues the agent learns to successfully stay positioned and follow the lane, as also make successful turns, with best heading possible.

Finally, the agent prefers to brake behind a car rather than overpassing or deviating from the car heading. Minimum heading deviation is always zero, as the agent spawns with optimal deviation at the start of each episode. Meanwhile, during the episode, the minimum heading deviation remains zero.

# 8. Conclusion

Undeniably driving optimally in urban settings is one of the hardest tasks to accomplish in autonomous driving. Being computationally large, it requires a massive set of parameters to be considered for the successful operation of the agent guiding the autonomous vehicle.

The model implemented in this thesis utilizes the most promising features from recent works, learning and architecture in Autonomous Driving. The evaluation metrics of the model show that the model improves its decision-making during training, as the episode length and rewards are increased, and it does not deviate from the optimal heading.

In general, the agent has improved behavior, compared to the initial model. The agent is capable of coping with some easy navigation cases in an urban environment. He manages to stay in the center of the lane, keep the optimal heading and avoid other vehicles in many easy cases, completing most episodes successfully.

# REFERENCES


[1] B. R. Kiran *et al.*, “Deep Reinforcement Learning for Autonomous Driving: A Survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022, doi: 10.1109/tits.2021.3054625.

[2] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A Survey of Autonomous Driving: *Common Practices and Emerging Technologies*,” *IEEE Access*, vol. 8, pp. 58443–58469, Jan. 2020, doi: 10.1109/access.2020.2983149.

[3] Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, “A survey of End-to-End Driving: Architectures and training methods,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364–1384, Apr. 2022, doi: 10.1109/tnnls.2020.3043505.

[4] Haoyi Niu, “ModEL: A Modularized End-to-end Reinforcement Learning Framework for Autonomous Driving,” *arxiv.org*, Oct. 22, 2021. Accessed: Sep. 27, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2110.11573

[5] M. Toromanoff, É. Wirbel, and F. Moutarde, “End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances,” *HAL (Le Centre Pour La Communication Scientifique Directe)*, Jun. 2020, [Online]. Available: https://hal-mines-paristech.archives-ouvertes.fr/hal-02513566

[6] H. S. Ansari and R. Goutam, “Autonomous Driving using Deep Reinforcement Learning in Urban Environment,” *International Journal of Trend in Scientific Research and Development*, vol. Volume-3, no. Issue-3, pp. 1573–1575, Apr. 2019, doi: 10.31142/ijtsrd23442.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy optimization Algorithms,” *arXiv (Cornell University)*, Jul. 2017, [Online]. Available: http://export.arxiv.org/pdf/1707.06347

[8] “AWS - CARLA Simulator.” https://carla.readthedocs.io/en/latest/tuto_G_rllib_integration/

[9] Carla-Simulator, “GitHub - carla-simulator/rllib-integration: Integration of RLLib and CARLA,” *GitHub*. https://github.com/carla-simulator/rllib-integration/tree/main

[10] Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, “CARLA: an open urban driving simulator,” *arXiv (Cornell University)*, Nov. 2017, [Online]. Available: https://arxiv.org/pdf/1711.03938.pdf

[11] “Overview — Ray 2.7.0.” https://docs.ray.io/en/latest/ray-overview/index.html

[12] D. Pomerleau, “ALVINN: an autonomous land vehicle in a neural network,” *Neural Information Processing Systems*, vol. 1, no. 77, pp. 305–313, Jan. 1988, doi: 10.1184/r1/6603146.v1.

[13] M. Bojarski *et al.*, “End to end learning for Self-Driving cars,” *arXiv (Cornell University)*, Apr. 2016, [Online]. Available: http://export.arxiv.org/pdf/1604.07316

[14] S. Ross, G. J. Gordon, and J. A. Bagnell, “A reduction of imitation learning and structured prediction to No-Regret online learning,” *arXiv (Cornell University)*, Nov. 2010, [Online]. Available: https://arxiv.org/pdf/1011.0686.pdf

[15] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, “Learning to Drive a Real Car in 20 Minutes,” *Frontiers in the Convergence of Biosc*, Jan. 2007, doi: 10.1109/fbit.2007.37.

[16] J. Koutník, J. Schmidhuber, and F. Gomez, “Evolving deep unsupervised convolutional networks for vision-based reinforcement learning,” *Genetic and Evolutionary Computation*, Jul. 2014, doi: 10.1145/2576768.2598358.

[17] J. Randløv and P. Alstrøm, “Learning to drive a bicycle using reinforcement learning and shaping,” *International Conference on Machine Learning*, pp. 463–471, Jul. 1998, [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml1998.html#RandlovA98

[18] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, “Driving policy transfer via modularity and abstraction.,” *Conference on Robot Learning*, pp. 1–15, Oct. 2018, [Online]. Available: http://proceedings.mlr.press/v87/mueller18a/mueller18a.pdf

[19] “What is AWS,” *Amazon Web Services, Inc.* https://aws.amazon.com/what-is-aws/

[20] “What is Cloud Computing,” *Amazon Web Services, Inc.* https://aws.amazon.com/what-is-cloud-computing/?nc1=f_cc