Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Aleksander Scherman Olsen

Adaptive General Reinforced Imitation in Autonomous Driving

Master's thesis in Computer Science
Supervisor: Frank Lindseth
Co-supervisor: Gabriel Kiss
June 2022

NTNU
Kunnskap for en bedre verden

Aleksander Scherman Olsen

Adaptive General Reinforced Imitation in Autonomous Driving

Master's thesis in Computer Science
Supervisor: Frank Lindseth
Co-supervisor: Gabriel Kiss
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

The task of autonomous driving is hard, and the task of autonomous driving based on visual input is even harder. Advances in visual intelligence and autonomous driving systems could have great societal benefits if such a system achieves better-than-human performance. A reinforcement learning approach could potentially be the solution to such a system. Reinforcement learning does however have challenges with sample efficiency and stability.

This thesis presents Adaptive General Reinforced Imitation, an adaptive method for combining the exploratory features of reinforcement learning with the expert demonstrations from imitation learning. The method seeks to reduce the number of samples needed for the agent to learn, by injecting expert demonstration data into the training data of the reinforcement learning algorithm. Experimental results indicate that for one of the implementations the method exhibits traits of being more robust than a corresponding vanilla reinforcement learning algorithm, and is able to learn a better policy.

Sammendrag

Oppgaven med autonom kjøring er vanskelig, og oppgaven med autonom kjøring basert på visuell input er enda vanskeligere. Fremskritt innen visuell intelligens og autonome kjøresystemer vil kunne ha store samfunnsmessige fordeler dersom et slikt system oppnår bedre ytelse enn mennesker. En tilnærming som tar utgangspunkt i forsterkningslæring (reinforcement learning) kan potensielt være løsningen på et slikt system. Forsterkningslæring har imidlertid både utfordringer med datamengden som er nødvendig for å lære og stabilitet i læringen.

Denne oppgaven presenterer Adaptive General Reinforced Imitation, en adaptiv metode for å kombinere de utforskende egenskapene til forsterkningslæring med ekspertdemonstrasjonene fra imitasjonslæring. Metoden forsøker å redusere mengden data som trengs for at agenten skal lære. Dette gjøres ved å injisere data fra ekspertdemonstrasjoner inn i treningsdataene til forsterkningslæringsalgoritmen. Resultater fra eksperimenter indikerer at for en av implementasjonene viser metoden tegn til å være mer robust enn en tilsvarende vanlig forsterkningslæringsalgoritme. Den er også i stand til å lære en bedre policy.

Preface

This thesis is a part of the research conducted at the NTNU Autonomous Perception Lab (NAPlab) at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisors Frank Lindseth and Gabriel Kiss for giving me the opportunity to write my master's thesis within this field of research, and for their feedback on my thesis. I would also like to thank Jan Grønsberg for providing me with access to a virtual machine with a GPU.

Contents

Figures

Tables

Code Listings

Acronyms

Chapter 1

Introduction

1.1 Background and Motivation

The research field of autonomous vehicles has been blooming in recent years, and over the last decades advances in machine learning and computational power have made autonomous vehicles closer to reality than ever before. As most autonomous driving approaches are dependent on a visual input of the surroundings, advances in the ongoing research field of visual intelligence have been, and still are, crucial for the development of successful autonomous driving systems.

Successfully creating an autonomous driving system that achieves better-than-human performance could be greatly beneficial to society. The U.S. Department of Transportation's National Highway Traffic Safety Administration estimates that in the first half of 2021, 20,160 people died in traffic accidents [1]. An autonomous driving system that is better at driving than humans can therefore help save the lives of thousands of people worldwide.

Deep reinforcement learning is considered a promising family of machine learning methods for achieving better-than-human performance in the task of driving vehicles. It has shown a great ability for complex problem solving based on self-learning and simulations. Complex tasks where deep reinforcement learning has performed better than humans include games such as Go, where Silver et al. created AlphaGo [2] that beat the world champion. Berner et al. also show in [3] that a reinforcement learning agent can be trained to beat the world champions in the e-sports game Dota 2, a highly complex game with imperfect information.

One of the main obstacles of reinforcement learning is that it requires very much data to be able to learn and generalize. It also often lacks stability in training and promising results could rely heavily on parameter tuning. Overcoming these obstacles could be crucial for the future of reinforcement learning for autonomous vehicles.

1.2 Goals and Research Questions

The goal of this thesis is to explore the possibility to speed up and enhance reinforcement learning by combining the exploration of a simulated environment with expert demonstration data. To do this, the method Adaptive General Reinforced Imitation (AGRI), an extension of General Reinforced Imitation by Chekroun et al.[4], is introduced. To test if the proposed method can fulfill the overall goal the following research questions are formulated:

RQ1: Can AGRI be used with value-based reinforcement learning methods to improve learning?

RQ2: Can AGRI be used with policy gradient reinforcement learning methods to improve learning?

RQ3: Can a kinematic bicycle model be used to enrich the state representation of the environment with basic knowledge about vehicle dynamics to improve learning?

1.3 Contributions

The following are the contributions of this thesis to the research field of deep reinforcement learning for autonomous driving agents:

- A literature review of state-of-the-art performing reinforcement learning and imitation learning approaches to autonomous driving.
- The introduction of the method Adaptive General Reinforced Imitation. The method is trained and tested using different implementations of reinforcement algorithms and evaluated on a separate dataset.
- A study of the use of policy gradient algorithms when combining reinforcement learning with expert demonstrations.
- A study of the effects of augmenting the state representation with basic knowledge of vehicle dynamics.

1.4 Thesis Structure

This thesis is divided into six chapters:

Chapter 1: Introduction This chapter introduces the background and motivation for this thesis, together with the research goal and questions that it aims to answer. The contributions and the thesis structure are also presented.

Chapter 2: Background and Related Work This chapter presents the relevant theory behind imitation learning, reinforcement learning, and deep reinforcement learning. Further, an overview of the programming tools and simulator is explained before, in the end, a selection of related work with state-of-the-art performance is presented.

Chapter 3: Methodology This chapter presents the methodology used to conduct experiments to answer the research questions. The method Adaptive General Reinforced Imitation, an extension to GRI by Chekroun et al. [4], is proposed and its implementation is explained. The chapter also describes the Python implementation of the environment and how the experiments are conducted.

Chapter 4: Results This chapter presents the experimental results obtained in the thesis. Both the results from training and evaluation are presented.

Chapter 5: Discussion This chapter discusses the results obtained in the thesis and answers the research questions. A discussion about the shortcomings of the implementations in this thesis is also provided.

Chapter 5: Conclusion and Future Work This chapter draws a conclusion from the results and the discussion. It also presents a possible suggestion for future work on extensions to AGRI.

Chapter 2

Background and Related Work

This chapter presents the theory behind imitation learning, reinforcement learning, and deep reinforcement learning relevant to this thesis. It explains how reinforcement learning tasks are modeled as Markov Decision Processes with a state space and an action space and the calculations for state values and action values are explained. Later the concept of policies will be explained as well as the difference between on-policy and off-policy. The programming libraries used in the thesis will be explained together with the simulated environment used, and other relevant simulated environments will be listed. In the end, a selection of related work with state-of-the-art performance will be presented.

2.1 Learning Agents

This section introduces the two most common approaches for end-to-end training of autonomous agents, imitation learning and reinforcement learning. An overview of imitation learning is presented before an in-depth explanation of reinforcement learning. The explanation of reinforcement learning addresses Markov decision processes, reward functions, value functions, and methods for finding optimal policies through policy gradients and value-based methods. In the end, an overview of relevant reinforcement learning algorithms will be presented.

2.1.1 Modular Learning vs End-to-End Learning

When training autonomous driving systems there are generally two approaches used. The first is a modular approach consisting of a pipeline of modules. A module could be anything that solves a sub-task based on the input or the output from other modules. Modules could be neural networks, rule-based approaches, or other kinds of expert systems. A drawback of the module-based approach is that they often require a lot of domain knowledge to implement the modules.

End-to-end approaches on the other hand require little to no domain knowledge to implement. In this approach, observations are mapped directly to actions, instead of solving sub-tasks that are then combined to solve the main task. The major benefit of this approach lies in its simplicity. The drawback of the end-to-end approach is its black-box nature. It is close to impossible to explain why an agent chooses an action based on a state.

2.1.2 Imitation Learning

Imitation learning is a supervised learning approach where the agent is trained to imitate actions from a dataset. The dataset consists of state observations and actions selected by an *expert*. The expert could be anything from a rule-based system to a human or even a trained neural network. The goal of the imitation learning agent is to learn a mapping, or a policy, that maps observations to actions.

Early approaches to imitation learning suffered from what Bellman refers to as the curse of dimensionality [5]. However, recent advances in deep learning have provided solutions to overcome these problems, and imitation learning has shown to be able to perform well in complex tasks such as autonomous driving tasks.

A challenge with imitation learning is that it usually generalizes poorly to new situations. The agents have only been trained on expert observations that never, or very rarely, make mistakes. Therefore, it tends to struggle to learn a policy that is able to recover from mistakes or adapt to new situations. A learning approach that tends to generalize better to new situations is reinforcement learning.

2.1.3 Reinforcement Learning

Reinforcement learning (RL) is a method in machine learning for agents to learn a wanted behavior in an environment without explicitly being told what actions are better than others. Instead, the agent learns how to behave by interacting with the environment through actions that affect the state of the environment. Figure 2.1 shows a simple visualization of how the agent observes the state s_t at time t and executes an action a_t on the environment and receives a reward r_{t+1} as the environment transitions into state s_{t+1}. In some literature r_{t+1} is denoted r_t to indicate the reward received for selecting action a_t. This is only a matter of notation, and in this thesis, the notation r_{t+1} is used. The reward received can be any real number, and the ultimate goal is for the agent to maximize the cumulative reward collected over a sequence of transitions, called an episode. In the first episodes, the agent will select actions at random, but as it experiences more episodes it will learn to create a map from actions to states that maximizes the reward.

2.1.4 Markov Decision Processes

To model RL problems Markov Decision Processes (MDPs) are used. A MDP is a time-discrete stochastic process that describes the evolution of a system over

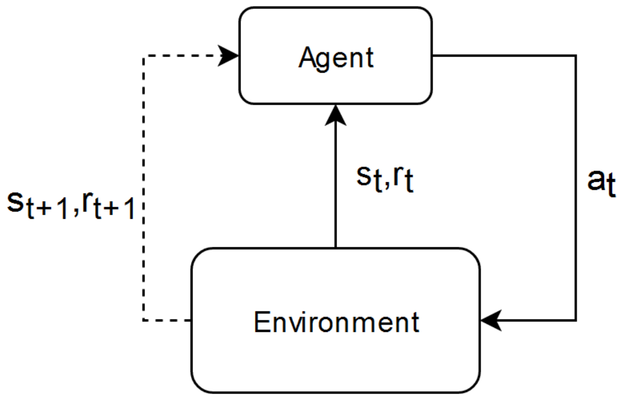


Figure 2.1: An illustration of an agent interacting with an environment. At time t the agent observes the state of the environment s_t and the reward r_t from applying action a_{t-1} in the previous time step. The agent then decides on an action a_t and executes it on the environment, that transitions into state s_{t+1} and the agent receives reward r_{t+1}.

time. The MDP has a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R}$, and a transition function p. The reward function determines the reward r_{t+1} for transitioning from a state s_t in time t to state s_{t+1} by applying action a_t to the environment. The transition function is defined as:

$$p(s', r|s, a) = Pr(s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a) \tag{2.1}$$

It describes the probability for moving from state $s_t = s$ to state $s_{t+1} = s'$ and receive reward $r_t = r$ when executing action a_t on the environment.

As the transition function is a probability distribution the following must hold

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \tag{2.2}$$

This puts a requirement on the state. It must contain all the information that makes a difference for future states, so that future states are only dependent on the current state and not the trajectory of all states leading up to it. If the state fulfills this requirement it is said to exhibit the Markov property.

From Equation (2.1) the dynamics of the environment can be calculated. The state-transition probability function can be expressed as

$$p(s'|s, a) = Pr(s_{t+1} = s'|s_t = s, a_t = a) = \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{2.3}$$

and the expected reward from selecting action a in state s can be expressed as

$$r(s,a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s,a). \tag{2.4}$$

2.1.5 Reward Functions

The cycle of an MDP consists of observing the state of the environment s_t, selecting and executing an action a_t, and receiving a reward r_{t+1}. Rewards are only given on the transition from one state to another so the agent will receive the first reward at time $t = 1$.

As the agent explores the environment by repeating the cycle of the MDP it generates a sequence of observed states, actions executed and rewards received. This sequence, or trajectory of state-action-reward tuples can be written as

$$\tau = \{(s_0, a_0, r_1), ..., (s_t, a_t, r_{t+1}), ...\}.$$

The goal of the agent is to find the action to take for each state so that the total reward of the trajectory is maximized. In its simplest form, this cumulative reward for the trajectory can be expressed as

$$R(\tau) = \sum_{t=0}^{T-1} r_{t+1} \tag{2.5}$$

Where T is the total number of steps in the trajectory. T can be an arbitrary number and must not necessarily be the same for every episode. It may also very well be infinite. RL problems that can be said to have a well-defined ending are referred to as episodic tasks. In these kinds of problems, T is finite and Equation (2.5) could be used as the trajectory reward. However, in other tasks, the episode may not have a natural ending but goes on forever. These kinds of problems are referred to as continuous tasks. In continuous tasks Equation (2.5) could be too simple to represent the total reward for the trajectory. As T approaches infinity $R(\tau)$ could also be approaching infinity. To avoid this, discounting of future rewards is used to make the sum of rewards converge. The total reward for a trajectory can then be expressed as

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}. \tag{2.6}$$

Here γ is the discount factor, and is defined so that $0 <= \gamma <= 1$. The discount factor indicates how much emphasis should be put on future rewards. In the case where $\gamma = 0$ the discounted rewards becomes $\gamma^t r_{t+1} = 0$ for all rewards except for the first reward received in time $t = 0$. In the other extreme case $\gamma = 1$ and the expression becomes the same as in Equation (2.5). To be able to use the same notation for both episodic and continuous tasks it is assumed that $r_t = 0$ for $t > T$. By setting γ so that $0 < \gamma < 1$, the sum of the rewards is guaranteed to converge. For example, if the reward is given as $r_t = 1$ for all t the total reward for the trajectory would be

$$\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}.$$

Sometimes, as will be seen in Section 2.1.9, it is useful to be able to calculate the reward of the remaining of a trajectory starting at time t. This can be expressed as

$$R_t(\tau) = \sum_{k=0}^{\infty} \gamma^t r_{t+k+1} \tag{2.7}$$

and by the property of the sum of a series, this can also be written as

$$R_t(\tau) = r_{t+1} + \gamma R_{t+1}(\tau) \tag{2.8}$$

2.1.6 Designing Environment Reward Functions

When trying to solve RL problems the design of the reward function plays a crucial role in how well the agent will learn to behave in the environment. In some types of environments, there could be a very well-defined set of rules for what should give a reward. An example of such an environment is the pole balancing environment described on page 56 in [6]. Here the reward function can be as simple as rewarding +1 for transitioning to any state that is not a failure state. In other problems, such as chess, designing the reward function could be far more difficult. The chess environment is so complex that even an expert would not be able to accurately estimate the reward for all state transitions.

One solution to the problem of complex environments is to only give a reward when the agent finishes an episode. If the agent finishes with success the agent is given a positive reward, and if the agent failed the episode the agent is given a negative reward. In the chess environment, this would mean giving a positive reward if the agent wins and a negative reward if the agent fails. For all other state transitions, the reward is 0. This is the exact approach used by Silver et al. in [2], where they train an agent to achieve expert-level performance in the game of Go.

While this approach might work well for episodic tasks such as chess and Go, it quickly becomes problematic when used in continuous tasks or problems with very long episodes. Imagine training an agent that is learning to drive an autonomous where the goal is to reach a final destination. Even if the destination is relatively close, e.g. less than a kilometer away, the agent would still have to observe and act on thousands of states before reaching the desired destination. The probability of an agent selecting the correct sequence of actions to reach the desired destination is infinitesimally small. This would lead to the agent wandering aimlessly and most likely learning to stand still to avoid collisions.

Giving the agent rewards for completing sub-goals could be a solution to the problem of sparse rewards. Sutton and Barto [6] do however argue that this should not be done as it could potentially make the agent learn to only complete sub-goals and not care about the overall goal. Therefore, the challenge of designing a good reward function plays a large part for RL agents to be able to complete goals successfully.

2.1.7 Policies

When the agent explores the environment and tries to learn how to maximize the total reward received it follows what is called a policy. The policy determines how the agent behaves in the environment and can be seen as a set of rules for what action to take in every state. Mathematically the policy is a mapping from a state to a probability distribution of actions that can be selected in the given state. The policy is defined as

$$\pi(a|s) = Pr(a = a_t | s = s_t) \tag{2.9}$$

for all $a \in \mathcal{A}(s)$ and all s $\in \mathcal{S}$. The notation π is used for simplicity. When learning, the agent will update its policy to maximize the expected reward. The theoretical policy that achieves this is called the optimal policy π^*. The goal when training is to learn a policy that is as close as possible to the optimal policy.

A policy that always selects the action with the highest probability is called a greedy policy. This approach may work well when applying a trained policy to a problem, but while the agent is exploring the environment this could easily cause the agent to get stuck in a local optimum. This problem is known as the challenge of exploration versus exploitation. Typically, early in the learning stages, it is desirable for the agent to explore the environment more than it exploits what it has learned about the environment up until that point. Take for instance the very simple environment shown in Figure 2.2. If in the first episode the agent selects the actions $a_0 = right, a_1 = right$ it will receive a total reward of 0, given no discounting. If the agent then in the second episode selects $a_0 = left, a_1 = left$ it will receive a total reward of 1.5. At this point, the agent should have updated its policy so that the probability for selecting $a_0 = left$ in state s_0 is higher than the probability for selecting $a_0 = right$. The optimal policy in this environment is $a_0 = right, a_1 = left$, but if the agent stops exploring after these two episodes it will get stuck in the local optima by exploiting the knowledge about the reward received from $a_0 = left, a_1 = left$. By exploring, the agent will select a random

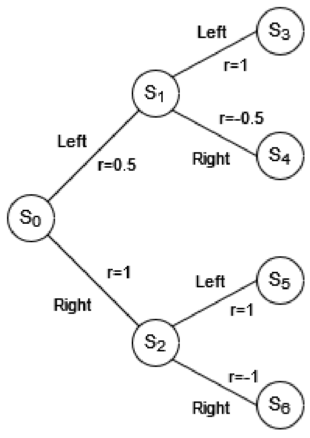


Figure 2.2: An example of a very simple environment with only two actions for each state.

action instead of following the policy in hopes of finding new actions that can give higher rewards. This trade-off between exploration and exploitation is an important aspect of successfully learning an optimal policy. Too much exploration can make the agent wander around without ever coming any closer to its goal, whereas too much exploitation may lead to the agent getting stuck repeating a sub-optimal pattern. A common way to let the agent explore the environment is to use stochastic policies. That way the agent will start out with relatively random actions and as it learns the better actions will be assigned a higher probability of being selected.

2.1.8 On-Policy vs. Off-Policy

When exploring the environment the agent needs to follow a policy. One method is to let the agent explore the environment using the same policy it is trying to optimize. This approach is called on-policy and is used in algorithms such as A2C [7], PPO [8], or TRPO [9]. The idea for on-policy is that there is always a small probability for selecting any action in any state so that $\pi(a,s) > 0$ for all states s and all actions a. As the agent learns, the policy will converge toward a determ-inistic policy. The challenge with on-policy algorithms is how fast to let the policy converge. Converging too fast will likely result in a sub-optimal policy while con-verging too slow might cause training to take too long to be feasible. This is the problem of exploration vs exploitation discussed in Section 2.1.7.

In contradiction to on-policy algorithms, off-policy algorithms use different policies for exploring the environment and learning the optimal policy. The exploring policy is often referred to as the behavior policy and the policy being learned is referred to as the target policy. The benefit of off-policy algorithms is that they can use experience collected by any policy. This allows them to reuse experiences collected earlier in the learning phase by storing state transitions, actions taken and the rewards collected in *replay buffers*. By using replay buffers, off-policy algorithms open up the possibility of inserting other data into the replay buffers, such as expert data, as Chekroun et al do in their work [4].

2.1.9 Value Functions

For the agent to be able to learn a policy it needs a way to determine the value of a state and the value of selecting an action in a given state. These value estimates are called value functions. The value function for a state is often called a state-value function and is denoted $V_\pi(s)$. It estimates the value of starting in state s and selecting actions according to the policy π from there on. The value function under the policy π is defined as

$$V_\pi(s) = \mathbb{E}_\pi[R_t(\tau)|s = s_t] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|s = s_t\right]. \tag{2.10}$$

Similarly the value of taking action a_t while in state s_t at time t is called the action-value function and denoted $Q_\pi(s,a)$. It is defined as

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_t(\tau)|s = s_t, a = a_t] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|s = s_t, a = a_t\right]. \tag{2.11}$$

It is worth noting that since the action-value is the value of taking action a_t in state s_t at time t, and then follow the policy from there on out it must be equal to the reward r_{t+1} received for transitioning from s_t to s_{t+1} plus the discounted future value of state s_{t+1} under the policy. The action-value function can therefore be expressed as a function of $V_\pi(s)$:

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[r_{t+1} + \gamma V_\pi(s_{t+1})|a = a_t, s = s_t\right]. \tag{2.12}$$

It is also worth noting that the following must hold:

$$\underset{a_{t+1}}{max} Q_\pi(s_{t+1}, a_{t+1}) = V_\pi(s_{t+1}) \tag{2.13}$$

For both $V_\pi(s)$ and $Q_\pi(s,a)$ the value of terminal states are always defined to be 0 so that $V_\pi(s_T) = Q_\pi(s_T, a_T) = 0$. Using both Equation (2.10) and Equation (2.11) it is possible to calculate the advantage of selecting another action a_t is compared to selecting based on the policy π. This is called the advantage function and is defined as the difference between the action-value function and the state-value function as follows:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.14}$$

Given that the agent follows an optimal policy π^* the advantage function $A_\pi(s,a)$ should have a maximum value of 0. If on the other hand, the advantage function is positive for any action it indicates that for the state s there exists an action a so that executing a and then following the policy is better than following the policy in s. This would mean that the agent is not following an optimal policy.

2.1.10 Tabular RL vs. Deep RL

When implementing the state-value function and action-value function one choice for implementation would be a one-to-one mapping from states to values so that every state is uniquely mapped to a value. An example of this is a tabular approach, where for each state an associated state-value is stored for the value function. Similarly, the action-value function would be represented by a mapping from a state-action pair to a value. The benefit of such an approach is that it accurately maps the states and state-action pairs to their respective values, and updating a value for one state or state-action pair has no influence on the other mappings. Another benefit of such an approach is that it is very fast as a lookup in a table is computationally cheap.

There are however drawbacks to this approach that often has a tendency to outweigh the benefits. The most important drawback is that it becomes very memory-consuming as the state space of the environment grows. Even a simple game such as tic-tac-toe has a state space size of $3^9 = 19683$ states. The number of state-action pairs would be even larger as for each state, several actions would be associated with it. For very simple environments, such as tic-tac-toe, the tabular approach is feasible in terms of memory consumption, but state space sizes can quickly grow too large to fit in the memory of any computer. Imagine an environment with a grayscale image of size 10×10 pixels. If each pixel can be a value in the range $[0, 255]$ then the size of the state space is 256^{100}. This is far more states than are possible to store on any computer. Another drawback with the tabular approach is that both the state space and action space have to be discrete. In some cases, this problem could be solved by discretization. Only the state space is assumed to be continuous in the following, but the same applies to action spaces. Take for instance pole balancing problem mentioned in Section 2.1.6 from [6]. In this environment, the state space is represented by the angle of the pole. The state space could then be discretized by rounding the angle to the closest integer and using that as the new state space. If the state space is already discrete, but large, one could sometimes use bins to cluster similar states and thus reduce the state space size. In [10] Chen et al. used a combination of discretization and bins.

In many environments, state spaces may already be discrete and very large, such as in chess or Go. In these cases, the state space is also very difficult to reduce in size by clustering states into bins, as it is very hard to determine accurately which states are similar. This is where deep neural networks as function approximators come in. By representing the state-value function, action-value function, or policy using a deep neural network it is possible to map states to values or actions without one-to-one mappings. Parameterized approximations are denoted $V_\theta(s)$, $Q_\theta(s,a)$, $\pi_\theta(s,a)$ where $\theta \in \mathbb{R}^d, d << |\mathcal{S}|$ is the d-dimensional vector of parameters for the functional approximation. Note that θ in the notation is not necessarily the same for V, Q, and π. The functional approximation has the property that updating the value for a single state, by updating the parameters θ, also updates the value of other states. In some cases, this could be an advantage, as it could speed up the learning process by updating multiple states at the same time. It could potentially also make learning harder because of the same property. Updating the value for one state could simultaneously corrupt the value for another.

2.1.11 Policy Gradient Methods

Methods that rely on a functional approximator of the policy needs a way to update the policy. The ultimate goal of the agent is to maximize the expected reward $R(\tau)$ for any trajectory τ. The expected reward over a trajectory τ is denoted $J(\pi_\theta)$ so that $J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathbb{E}}[R(\tau)]$. $\underset{\tau\sim\pi_\theta}{\mathbb{E}}$ reads the expected value of a trajectory when following policy π_θ. For the agent to learn an optimal policy it has to find the optimal parameters θ^* for the policy. This is done by solving the following optimization problem

$$\max_\theta J(\pi_\theta) \tag{2.15}$$

To solve Equation (2.15), θ is updated over a series of small steps by gradient ascent. After k updates the new value for θ at the next iteration $k+1$ is given by

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}. \tag{2.16}$$

In Equation (2.16) $\nabla_\theta J(\pi_\theta)$ is referred to as the policy gradient. α is the size of the step to take along the gradient when up θ and is referred to as the learning rate. Silver et al. shows in [11] how the expected reward $J(\pi_\theta)$ can be written as

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(s,a) r(s,a) da ds. \tag{2.17}$$

Here $\rho^\pi(s)$ is the probability distribution of states under the policy π. They further show how the policy gradient can be expressed as

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(s,a) Q_\pi(s,a) da ds \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi}[\nabla_\theta log \pi_\theta(s,a) Q_\pi(s,a)].\end{aligned} \tag{2.18}$$

2.1.12 Value Based Methods

An alternative to policy gradient methods is value-based methods. Instead of calculating the gradient $\nabla_\theta J(\pi_\theta)$ to do gradient ascent, value-based methods aims to maximize the equation

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[r_{t+1} + \gamma \underset{a_{t+1}}{max} Q_\pi(s_{t+1}, a_{t+1}) | a = a_t, s = s_t \right] \tag{2.19}$$

This is the same equation as Equation (2.12), but with Equation (2.13) inserted for $V_\pi(s_{t+1})$. Equation (2.19) is known as the Bellman equation for value based methods. Value based methods aim to maximize this equation through Bellman iterations, so that

$$Q_{i+1}(s,a) = \mathbb{E} \left[r_{t+1} + \gamma \underset{a_{t+1}}{max} Q_i(s_{t+1}, a_{t+1}) | a = a_t, s = s_t \right] \tag{2.20}$$

Sutton and Barto explains in [6] that Equation (2.20) can be shown to converge to Q^* as $i \longrightarrow \infty$.

2.1.13 Deep Q-Learning

Deep Q-Learning (DQN) was introduced by Mnih et al. in [12]. They argue that using Equation (2.20) directly is impractical, as the action-value function is estimated separately for each sequence. Instead they propose using a neural network as a functional approximator for Q, so that $Q_\theta(s,a) \approx Q^*(s,a)$. The authors refer to this neural network as a Q-network. They then train the Q-network by minimizing the loss function

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{\pi_{behavior}} \left[(y_i - Q_{\theta_i}(s,a))^2 \right] \tag{2.21}$$

where

$$y_i = \mathbb{E}_\pi \left[r_{t+1} + \gamma \underset{a_{t+1}}{max} Q_{\theta_i}(s_{t+1}, a_{t+1}) | a = a_t, s = s_t \right]. \tag{2.22}$$

In the implementation provided by Stable-Baselines3 [13] $\pi_{behavior}$ is implemented using a second Q-network. As DQN is an off-policy algorithm it is able to use replay buffers. This is utilized in the algorithm by randomly sampling a minibatch of transitions at each step and then doing a step of gradient descent based on the gradient of Equation (2.21).

2.1.14 Deep Deterministic Policy Gradient

With DDPG [14] Lillicrap et al. introduced algorithm that estimates a deterministic parameterized actor function $\mu_\theta(s)$ that maps states to actions. By doing this they are able to use $\mu_\theta(s)$ as an estimate for the action a when training the Q-network. This approach allows them to use deep Q-learning in continuous action spaces. By using the definition of $J(\pi_\theta)$, the policy optimization problem in Equation (2.15) can then be written as

$$\underset{\theta}{max} \underset{s \sim \mathcal{D}}{\mathbb{E}} \Big[Q_\phi(s, \mu_\theta(s)) \Big]. \tag{2.23}$$

During the optimization of the policy, the parameters ϕ of the action-value function are treated as constants. $s \sim \mathcal{D}$ indicates states collected from a replay buffer $\mathcal{D}$. The Q-network is trained separately as explained in Section 2.1.13.

2.1.15 Twin Delayed Deep Deterministic Policy Gradient

In [15] Fujimoto et al. presents the Twin Delayed Deep Deterministic Policy Gradient (TD3) in an attempt to improve DDPG [14]. It is an algorithm that takes into account the error that occurs from using function approximators for both the value function and the policy. They recognize that the estimation $Q_\phi(s, \mu_\theta(s))$ is susceptible to error and may introduce overestimation bias. To handle the problem of overestimation bias they use clipping by simultaneously training two Q-networks (twins) and using the minimum value of the two.

The second improvement Fujimoto et al. makes is to delay the updating of the policy network compared to the value networks. They suggest that updating the policy based on value estimates with high errors could lead to divergent behavior. By updating the policy network at a lower frequency the Q-networks will have more iterations of updates to more accurately estimate the actual value.

The last improvement to DDPG that TD3 makes is to add a smoothing regularization term to the target in Equation (2.22). They write this as

$$y = r + \mathbb{E}_\epsilon \Big[Q'_\theta(s', \pi'_\phi(s') + \epsilon) \Big] \tag{2.24}$$

This expectation can then be expected by adding a small amount of random noise, so that

$$\begin{aligned} y &= r + \gamma Q'_\theta(s', \pi'_\phi(s') + \epsilon) \\ \epsilon &\sim clip(\mathcal{N}(0, \sigma), -c, c) \end{aligned} \tag{2.25}$$

where ϵ is sampled from a normal distribution and clipped to the interval $[-c, c]$.

2.2 Reinforcement Learning Libraries and Tools

This section presents a selection of RL libraries and tools relevant to this thesis. The presented technologies help simplify the learning process of RL agents by providing a standardized interface to interact with.

2.2.1 OpenAI Gym Environment

OpenAI Gym [16] is an open-source toolkit that aims to simplify the process of handling RL environments. The authors deliberately only create an abstraction for the environment and not the agent. They argue that this is to maximize the convenience for the users while allowing for different implementations of the agents. The environment interface proposed mainly consists of two methods: *reset()* and *step()*. *reset()* sets the environment back to an initial state and returns the state. *step()* takes an action as an argument and applies it to the environment. The implementation of the environment is then responsible for calculating the next state and the reward received. A tuple of four values is returned: (state, reward, done, info). The first three values are self-explanatory, and the info-value is a dictionary of metadata about the environment. This value should not be accessed by the agent but is for logging and debugging purposes only. In this thesis, the info-value is never used, and for simplicity in notation, the tuple returned by *step()* will hereafter only be referred to as (state, reward, done).

In addition to *reset()* and *step()* the interface also requires the environment to define an action space and an observation space. Action spaces and observation spaces can be discrete, continuous, or a combination of the two. In complex environments, such as in autonomous driving, parts of the state are represented with continuous numbers, such as the speed of a vehicle, while other parts of the state could be discrete, such as the current gear. Gym environments make this possible by allowing action spaces and observation spaces to have sub-spaces that can be discrete or continuous independently.

2.2.2 Stable-Baselines3

Stable-Baselines3 [13] is an open-source framework that aims to simplify the process of using RL algorithms. The authors created the framework for researchers to have the exact same implementation of RL algorithms when conducting experiments. The authors argue that Henderson et al. in [17] show that very small differences in the implementation of an algorithm can have a greater impact than the choice of the algorithm itself. This is especially important when new RL algorithms are developed. If the new algorithm is compared to an unstable or unreliable implementation of a baseline algorithm, the performance of the new algorithm might seem better than it actually is. The algorithms implemented are A2C [7], PPO [8], DDPG [14], SAC [18], TD3 [15], HER [19], and DQN [12]. All algorithms have been thoroughly tested by comparing the learning curves of the agent with published work and by the use of unit tests for most methods. All algorithms are designed to interact with an implementation of a Gym environment. This is to ensure that the environments have a standardized interface to interact with.

2.3 Simulated Environments for Autonomous Vehicles

This section presents a selection of simulated environments used for representing the environments for the agents to explore. The main focus will be on CARLA [20], as this is the most widely used simulator for autonomous vehicles, and the one used in this thesis. At the end of the section, some alternatives to CARLA will also be presented.

2.3.1 CARLA

In [20] Dosovitskiy et al. present CARLA (Car Learning to Act), an open-source simulator intended to support research for autonomous urban driving. The simulator features realistic physics and photo-realistic graphics as well as logic for the other vehicles. The simulator is implemented as a client-server architecture. The server is responsible for running the simulation and handling the rendering of the environment, and the client is responsible for all agent logic. The client communicates with the environment through an API implemented in Python[1].

Through the Python API the client can send commands to control the agents, or meta-commands to control the environment itself. Commands control the agents by steering, throttling, braking, gear change, etc. Meta-commands control the environment itself, such as changing the weather or the city the agent is driving in. The client is also responsible for receiving sensor readings from the server.

Agents observe the environment in CARLA through sensors. There is a variety of sensors that can be attached to the vehicle. Some of the sensors provide realistic data, as would have been in the real world, whereas other sensors provide ground truth data from the simulator, allowing for supervision when training the autonomous agent. Below is an overview of some of the sensors available in CARLA.

Realistic Sensors

- RGB camera: Provides RGB images of the surroundings of the vehicle. The camera can be rotated in any direction and the field of view and image size can be customized.
- LIDAR: Provides the coordinates and intensity of the surroundings of the vehicle. The intensity can be used to measure the distance to other objects.

[1]Documentation for the Python API can be found here: `https://carla.readthedocs.io/en/0.9.13/python_api/`

- GNSS: Provides the altitude, latitude, and longitude of the vehicle.
- IMU: Provides information about the acceleration in m/s^2, the orientation in radians relative to North, and the angular velocity in m/s.
- Radar: Provides information about elements in sight and their movement.

Ground Truth Sensors

- Collision detector: Signals if the vehicle has collided. If this is the case an indicator is returned making it possible to determine what the agent has collided into.
- Depth sensor: Provides an image where the pixels represent the distance to the elements in the view.
- Lane invasion detector: Signals if the vehicle has crossed a lane marking. Tells if the vehicle has crossed over a lane marking. If this is the case an object representing the lane marking is returned, making it possible to determine what kind of marking was crossed.
- Semantic segmentation camera: Provides an image where each pixel is an integer in the range $[0, 22]$. Each pixel represents a tag for what class of object is in the pixel.
- Semantic LiDAR: Similarly to the semantic segmentation camera, the semantic LiDAR provides semantic information about what class of object is present at each point in the LiDAR reading.

In addition to sensors, the agent also has access to an HD map of the driving area in the OpenDrive format[2]. The HD map is static and only has to be fetched from the server once by the client. The CARLA Python API provides methods for parsing these files to represent the map in a manageable way. One of the most relevant features provided by the Python API is the Waypoint API. The Waypoint API is one of the main building blocks for most of the methods presented in Section 2.4 and allows for the client to generate so-called waypoints. Waypoints are 3D objects that provide information about roads and lanes on the HD map. As the waypoints are generated directly from the HD map no interaction with the CARLA server is required. This means that any information provided by waypoints can be used for training and inference as it is not privileged information.

CARLA Leaderboard and the Scenario Runner

In recent years challenges for creating the best autonomous vehicle driving in CARLA have appeared. The CARLA challenge[3] was one of the first challenge to arrive with realistic driving scenarios. Later the CARLA Autonomous Driving Leaderboard[4], or just CARLA leaderboard for short, was also created and is today the

[2]https://www.asam.net/standards/detail/opendrive/
[3]https://carlachallenge.org/
[4]https://leaderboard.carla.org/

de-facto standard for benchmarking autonomous driving agents in CARLA.

The CARLA leaderboard challenge comes in two flavors. The first is the SENSOR challenge where the driving agents only have access to sensors and not the HD map provided by the server. In the MAP challenge, the driving agents have access to the HD map in addition to the same sensors as in the SENSOR challenge. In both challenges, the agent has a limit on how many of each sensor can be used. The sensors include GNSS, IMU, LiDAR, RADAR, RGB camera, and a custom pseudo sensor that provides an approximation of the speed.

The CARLA leaderboard challenge is run through the Scenario Runner[5], an open-source GitHub repository. The Scenario Runner provides functionality to define routes and scenarios that the agent should follow. Routes are given as a list of waypoints with a high-level command associated with each waypoint. A high-level command is an integer value ranging from 1 to 6 representing the actions: turn left, turn right, keep straight in an intersection, follow the lane, change lane left, and change lane right. Other mentions of high-level commands in CARLA in this thesis refer to these commands. Scenarios are predefined tasks that the agent should encounter while driving the route. An example of such a scenario is making a right turn in an intersection while a pedestrian suddenly crosses in front of the vehicle.

2.3.2 Alternatives to CARLA

CARLA is not the only simulator available for training autonomous driving agents. TORCS [21], short for The Open Racing Car Simulator, is another alternative. Similar to CARLA, it is also implemented in a server-client architecture with a low-level API for controlling the car. The simulator is made for simplicity while at the same time being able to provide a realistic simulation. The default interface through the API can provide ground truth information about the position of the car and the distance from the lane center. As the graphics of the simulator are relatively basic TORCS is more suitable for trajectory planning tasks rather than visual perception tasks. It also does not have support for urban driving with traffic rules and pedestrians.

The LG SVL simulator [22] is another simulator intended for training autonomous driving agents. It is built using the Unity engine and was intended to be a simple-to-use simulator that could be integrated with already existing autonomous driving stacks. The authors argue that CARLA and other simulators are too difficult to add custom sensors and more complex logic for other actors. The SVL simulator is intended to handle this problem. It features several sensors, including all the realistic sensors mentioned in Section 2.3.1. It also provides an HD map of the environment similar to CARLA.

Another simulated environment that is not specialized for autonomous agents is

[5]`https://carla-scenariorunner.readthedocs.io/en/latest/`

the Unity ML-Agents Toolkit[6] is an open-source toolkit for training machine learning agents in the Unity engine. While it is not specialized for autonomous driving tasks it is still possible to do. This does however more designing of the environment than the aforementioned simulators. The toolkit provides a python API for RL algorithms, as well as functionality to do imitation learning. The Python API also provides a Gym wrapper for the environment so that researchers can interact with the environment through the Gym interface presented in Section 2.2.1. This is intended to reduce the time researchers spend learning the toolkit so that they can focus their time on the machine learning tasks.

2.4 Related Work

This section presents a selection of related work that has had success in training autonomous agents that achieves state-of-the-art performance. The main focus will be on their method for training their agents and how they have implemented their solutions.

2.4.1 Learning by Cheating

Learning by Cheating [10] from 2019 was the first of the three submissions by Chen et al. currently on the CARLA leaderboard. At the time of writing this thesis, it is ranked number 14 out of 18 submissions. Their method consists of three stages of training.

Stage 1 First, a privileged agent is trained that have access to a map of ground-truth information about the environment. The ground-truth information includes lane and road information as well as the location of other vehicles, pedestrians, and traffic lights. The agent is trained to predict the next waypoints that the agent should steer towards. Predictions are done using a convolutional network where the input is a binary map anchored at the agent's position with ground truth data. The input map has dimensions (H, W, 7) for height H, width W, and 7 categories of objects. The predicted categories are roads, lane boundaries, vehicles, pedestrians, and one category for each traffic light state. A visual representation of the map can be seen in Figure 2.3. The output of the network is a series of heatmaps. One heatmap for each waypoint and high-level command (follow the lane, turn right, etc). The heatmaps are then converted into waypoints using soft-argmax.

Stage 2 The second stage of the training consists of training the privileged agent on a dataset of prerecorded expert trajectories using behavior cloning. The dataset of trajectories is noise-free, and augmentations are added offline by shifting and rotating the ground-truth maps, and therefore require no modification to the collection procedure of the expert data. The unprivileged, sensorimotor agent is

[6]`https://github.com/Unity-Technologies/ml-agents`

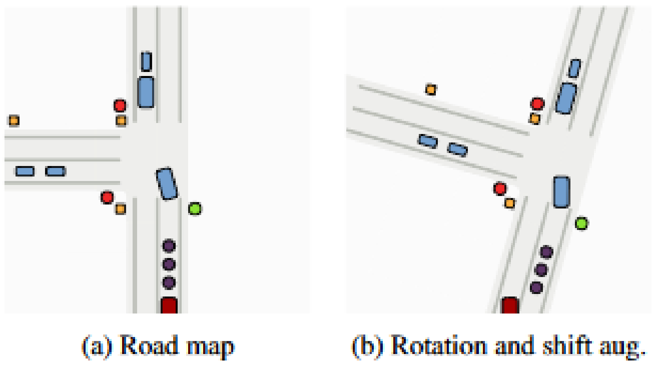


Figure 2.3: A visual representation of the map that is used as input to the privileged agent. Roads and lane boundaries are represented as light gray and gray, respectively. Vehicles are blue, pedestrians are orange and red, and yellow and green are represents traffic light states. The purple dots are the predicted waypoints and are not input to the model. The image is from the original paper [10]

then trained on the same trajectories as the privileged agent using off-policy behavioral cloning. The sensorimotor agent does not have access to the ground-truth map that the privileged agent used. Instead, it uses a single RGB image as input. The objective of the sensorimotor agent is to predict the same waypoints and high-level commands that the privileged agent predicts for each input.

Stage 3 The last stage of the training consists of on-policy imitation learning via DAgger [23]. The loss function is given as the L_1 distance between the waypoints predicted by the privileged agent and a transform of the predicted waypoints by the sensorimotor agents. The transform of the sensorimotor agent is calculated by utilizing the fact that waypoints in the map are related to waypoints in the image by a fixed perspective given by the position of the RGB camera. This lets the sensorimotor train on data not in the original dataset of expert trajectories by using the privileged agent model as target predictions.

Implementation For training the privileged agent and the sensorimotor agent uses different versions of ResNet [24] as backbone. The privileged agent uses an untrained ResNet-18 whereas the sensorimotor agent uses a pretrained ResNet-34 pretrained on ImageNet [25]. Both agents use three up-convolutional layers where each of these layers is given the velocity of the agent as an additional input. The up-convolutional network branches into four heads, each predicting a heatmap for high-level command (follow the lane, turn right, etc). These heatmaps are then converted into spatial coordinates using a differentiable soft-argmax. The input map to the privileged agent has a resolution of 192x192 and the output is a heatmap with resolutions 48x48. The input is cropped so that the agent is at the center bottom of the map. The augmentations applied include rotating the image by [-5, 5] degrees and shifting the image [-5, 5] pixels either left or right. Both the number of degrees for rotating and the number for pixels to shift are selected uniformly at random. The input to the sensorimotor agent is a 384x160 RGB image and the output heatmap is 96x40. The same augmentations as in CIL [26] are used.

Results The results achieved in the paper achieve state-of-the-art performance at the time. Compared to CILRS [27] it reduces the number of traffic light violations by an order of magnitude for all scenarios tested. It also has significantly fewer collisions than CILRS. Chen et al. conclude by stating that vision-based autonomous driving tasks can be more effectively trained using imitation learning by decomposing the learning process into first learning to drive, then learning to see.

2.4.2 End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances

In their work from 2019, Toromanoff et al. present a technique they coin implicit affordances [28] to effectively leverage RL for autonomous driving. Their method won the CARLA challenge[7] 2019, a challenge similar to the CARLA leaderboard. At the time of writing this thesis, their method is currently at 8th place in the CARLA leaderboard. They train their model in two stages.

To train their model they start by training a ResNet-18 visual encoder using a dataset of images from a single RGB camera. The encoded output is what will later be used to represent the state of the environment during the RL phase. The input to the backbone is four consecutive 288×288 pixel RGB images stacked on top of each other. This is done to add temporal information to the input. The output of the encoder is a vector with shape $512 \times 4 \times 4$. The encoder is trained similarly to how auto-encoders are trained. The encoded vector is passed to a semantic segmentation decoder that predicts the semantic segmentation of the images. The authors argue that using an RGB decoder instead of a semantic decoder would not work, particularly for traffic light detection. They argue that traffic lights represent very few pixels in an image, but that these pixels are important for the agent behavior, and that an RGB decoder is not able to capture this importance sufficiently.

In addition to training the encoder using the semantic decoder, the encoded vector is also flattened and passed to a fully connected network that predicts affordances. The authors argue that this is to ensure that the encoded RL state has enough relevant signals. There are six affordances predicted. These are traffic light presence, traffic light state, distance to the traffic light, intersection presence, distance to the center of the lane, and the rotation relative to the lane. The ground truth data for the first four affordances can be acquired directly from CARLA, whereas the last two have to be acquired by augmenting the viewpoints of the images in the

[7]`https://carlachallenge.org/`

dataset. The authors explain how this is necessary as the dataset only consists of images from an autopilot that is always in the middle of the lane with no rotation. The losses from the fully connected affordance network and the semantic decoder are then used in the backpropagation when training the ResNet-18 encoder.

During the RL stage the ResNet-18 encoder weights are frozen and therefore not trained any further. The encoded vector is used in a conditional network as in [29] together with a Rainbow-IQN [30] to handle high-level commands for following the lane, turning left, right or keeping straight in intersections, or changing lanes left or right. As the agent never predicts the affordances explicitly during the RL training phase the authors say that it predicts them implicitly, hence they coin the term implicit affordances.

The reward function used in the RL training phase consists of three components. It has a speed component, a positional component, and a rotational component. The speed component is in the interval $[0,1]$. The reward is 1 when the agent is driving at the desired speed and is reduced linearly to 0 if the agent is driving faster or slower. The desired speed is set to a maximum of 40 km/t and is reduced linearly to 0 when approaching a red light or a vehicle or pedestrian. The positional reward component is in the interval $[-1,0]$. It is 0 when the agent is in the center of the lane and is reduced to -1 when the agent approaches a 2 meter distance from the lane center. If the agent reaches the maximum distance from the lane center of 2 meters the episode terminates and the agent is rewarded -1. Using only these two reward components the authors experienced that the agent oscillated around the lane center. To prevent this they added the third component that rewards the agent for having the same relative rotation as the lane center. The reward is in the interval [-1, 0] and is inversely proportional to the difference in angle between the agent and the lane center.

At the time of release Toromanoff et al. achieved state-of-the-art performance for an end-to-end model-free RL approach. They compare their results to the RL experiment in [20], as well as CAL [31], CILRS [27], and Learning by Cheating [10] mentioned in Section 2.4.1. Toromanoff et al. outperform all methods except Learning by Cheating in most metrics. In the CoRL2017 [8] (Conference on Robot Learning 2017) benchmark, the implicit affordance approach scores 100% on all metrics except for the one the author calls the hardest metric. In the *NoCrash* benchmark [32] Toromanoff et al. achieve results that are slightly worse than Learning by Cheating. This is however the first time an RL approach has matched an imitation learning approach according to the authors.

2.4.3 Learning to Drive From a World on Rails

Learning to Drive From a World on Rails [33] is the second submission by Chen et al. to the CARLA leaderboard. When it was submitted in 2021 it ranked first

[8] `https://sites.google.com/a/robot-learning.org/corl2017/corl2017`

on the leaderboard with a Driving Score 25% higher than the second place and is currently ranked 7th. They use a model-based approach where they create a semi-parametric forward model of the environment. The model is factorized into two components, one component for the forward model of the ego vehicle and one for the forward model of the world. In doing this, they make the assumption that "*the world is on rails, meaning neither the agent nor its actions influence the environment.*" [33] They show how this assumption greatly simplifies the learning problem, and that despite that the assumption does not hold, the agent learns to drive well in a dynamic and reactive world. The ultimate goal of Chen et al. is to train a visuomotor policy that maps input RGB images to actions. Their approach consists of three modules: A forward model, $\mathcal{T}$, of the world, an action-value function, Q, and a policy π.

Forward model The forward model estimated is a semi-parametric model that consists of a ego-component $\mathcal{T}^{ego}$ and a world-component $\mathcal{T}^{world}$. The ego component is approximated using a simple deep neural network while the world component uses prerecorded driving logs as a non-parametric approximation. The driving state of the world is denoted L_t for time step t. The ego-component computes the forward ego-driving state as $L_{t+1}^{ego} = \mathcal{T}^{ego}(L_t^{ego}, L_t^{world}, a_t)$, given the current ego driving state, world state and action a. The world forward model computes the forward world state as $L_{t+1}^{world} = \mathcal{T}^{world}(L_t^{ego}, L_t^{world}, a_t)$. By the world on rails assumption, this simplifies to $L_{t+1}^{world} = \mathcal{T}^{world}(L_t^{world})$ as the agent state and actions are assumed to not have any influence on the world. Since the forward world state is now only determined by the previous world state the entire trajectory of world states can be determined from the initial world state. This allows for the prerecorded trajectories to model the world transitions directly. The ego-component is modeled as a parametric bicycle model [34] and is trained on rollouts of T=10 steps using the following L1 regression:

$$E_{\hat{L}^{ego}_{t:t+T}, \hat{a}_t} \left[\sum_{\Delta=1}^{T} \left| \mathcal{T}^{ego\Delta}(\hat{L}_t^{ego}, \hat{a}_{t+\Delta-1}) - \hat{L}_{t+\Delta}^{ego} \right| \right]$$

In the equation above, hat notation denotes data from prerecorded driving logs.

Action-value function To estimate the action-value function, $Q(\hat{L}_t, a$, a factorized version of the Bellman equation is used. This is possible due to the world-on-rails assumption. The original γ-discounted Bellman equation is

$$V(L_t) = \underset{a}{max}\ Q(L_t, a) = \underset{a}{max}\ \gamma V(\mathcal{T}(L_t, a)) + r(L_t, a),$$

but with the simplifying assumption and the factorized forward model, the authors show how this simplifies to

$$\hat{V}(L_t^{ego}) = \max_a \hat{Q}(L_t^{ego}, a)$$
$$\hat{Q}(L_t^{ego}, a) = r(L_t^{ego}, \hat{L}_t^{world}, a_t) + \gamma \hat{V}_{t+1}(\mathcal{T}^{ego}(L_t^{ego}, a)).$$

Here the action-value function is dependent on all ego states, but only prerecorded world states. Therefore, the agent is able to calculate the value of different ego states without actually moving to them. For this to work, the reward function has to be explicit and not just a black box output from the environment. The ego-state, L_t^{ego} is compact enough so that it is possible to compute a tabular approximation of $V_t(L_t^{ego})$ instead of using a deep network. This is done by creating bins based on discrete values of the position, orientation, and velocity of the ego vehicle. At evaluation time linear interpolation is used to estimate values that are not in the center of the bins.

Reward function As stated earlier the reward function must be explicit for a given ego-state, world-state, and action. The reward is designed to be in the range [0, 1] where 1 indicates that the vehicle has the desired speed, position, and orientation. The reward is smoothly decreased to 0 for errors in any of these criteria. In scenarios where the agent is in so-called zero-speed zones, such as behind another stopped vehicle, it receives an additional reward of +5 for braking. This braking reward can only be received once for every zero-speed zone so the agent does not actively seek out zero-speed zones.

Policy The policy is trained using the action-value function $Q_t(\hat{L}_t^{ego}, \cdot)$ to supervise the policy $\pi(\hat{I}_t)$. The learning objective is to maximize the expected return of the policy given by:

$$E_{\hat{L}_t^{world}, \hat{L}_t^{ego}, \hat{I}_t} \left[\sum_a \pi(a|\hat{I}_t) \hat{Q}_t(\hat{L}_t^{ego}, a) + \alpha H\big(\pi(\cdot|\hat{I}_t)\big) \right]$$

where H is an entropy regularizer [18] with temperature α.

Implementation Through discretization the ego-state is represented as a 4D vector containing 96 × 96 bins for positions, 4 bins for velocity, and 5 bins for orientation. Each of the positional bins represents a $\frac{1}{3} \times \frac{1}{3} m^2$ area in the simulated world. The bins for velocity and orientation represents $2m/s$ and 38° respectively. Orientations are only represented in the range $[-95°, 95°]$, and any value that is either too high or too low to be inside the discretization is given a value estimate of 0. Actions are discretized to 9 values for steering, 3 values for throttle, and one value for braking for a total of $9 \cdot 3 + 1 = 28$ possible actions.

The policy network is implemented using ResNet34 [24] as backbone for the RGB images. The output of the backbone is flattened using global average pooling before being fed to the fully connected network together with the vehicle velocity. The output of the fully connected network is a vector of size 28, one for each action, representing the probability of selecting each action. The authors use the same augmentations as in their previous work, Learning by [10], presented in Section 2.4.1.

Results The policy trained by Chen et al. achieved state-of-the-art performance, outperforming the previous leader, Implicit Affordances [28] presented in Section 2.4.2, by 25% on the Driving Score metric. This was done using only 1 million time frames for training versus 40 million in Implicit Affordances. The authors conclude by addressing that even though the world-on-rails assumption rarely holds, the benefits of training and sample efficiency outweigh the constraints.

2.4.4 GRI: General Reinforced Imitation and its Application to Vision-Based Autonomous Driving

In their work Chekroun et al. presents General Reinforced Imitation (GRI) [4], a method for combining RL with imitation learning to combine the exploration from RL with expert data from imitation learning. In the autonomous driving domain, they call their method GRI for Autonomous Driving (GRIAD). At the time of submission in 2021, their method ranked first on the CARLA leaderboard, and at the time of writing this thesis, their method is ranked 5th.

GRIAD aims to leverage expert demonstration through imitation learning to speed up the learning process compared to standard RL. At the same time, they want to keep the exploring nature of RL to generalize better to unseen scenarios. To do this they create an end-to-end system based on off-policy reinforcement algorithms. The authors build upon the hypothesis that expert demonstrations can be seen as perfect data following an optimal policy. By using off-policy RL using replay buffers they are able to insert expert data directly into the replay buffer for the policy to learn from. At the beginning of an episode, the system randomly collects an episode by letting an online agent explore the world in the CARLA simulator, or sample an episode trajectory from a prerecorded dataset. The authors denote the probability for sampling from expert demonstrations p_{demo}.

Chekroun et al. explain that the training pipeline in GRIAD is inspired by Toromanoff et al. in Implicit Affordances [28]. They first train two visual encoders for transforming visual input from RGB cameras to a compressed vector representation before freezing the weights before the RL stage. The authors use a specialized EfficientNet-b1 [35] as the encoder backbone. They argue that the rationale for using EfficientNets is that it lets them keep the same accuracy for the segmentation task and classification task as Toromanoff et al. while reducing the encoded vector size by a factor of almost 5.

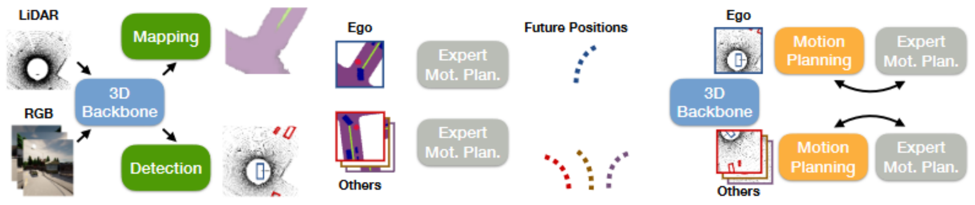
The first visual encoder is a semantic segmentation encoder that is trained as an auto-encoder by trying to decode the semantic segmentation map from three RGB images as input. The three cameras are mounted in front of the vehicle at different angles to give a wide view of the environment ahead. The output from the semantic encoder is three encoded vectors, each of size 448, one for each camera. The second visual encoder also uses EfficientNet-b1, but instead of predicting segmentation maps, the encoded vector is passed to a classification decoder. It also differs from the semantic encoder by only using input images from the center camera. This decoder learns to predict whether or not there is an intersection present, the traffic light state, and the distance to the traffic light if there is one present. The visual encoders are trained on a dataset of 400,000 samples collected using the autopilot provided by the CARLA Python API.

For training the RL agent Chekroun et al. use Rainbow-IQN Ape-X [30]. As this method uses DQN [12] the action space had to be discretized. The authors did this by using 27 values for steering and 4 values for controlling the speed through braking or throttling. The discretized vector space then had a size of $27 \times 4 = 108$. In their experiment, the authors use a distributed system with 12 agents running at the same frequency. Three of these collect expert data, and the authors explain that this is then equal to $p_{demo} = 25\%$. The demonstration dataset of expert trajectories consisted of 200,000 samples. Based on the underlying assumption that expert agents always follow an optimal policy the authors set a fixed reward for demonstration samples as $r_{demo} = 1$. For online exploration in the environment, Chekroun et al. used the same reward signal as Toromanoff et al. used in Implicit Affordances [28].

The authors show that by training their agent for 60 million time steps they achieved state-of-the-art performance at the time. They outperformed World on Rails [33] by approximately 17% on the Driving Score, and also achieved a higher score on the Route Completion metric and Infraction Score. The authors note how the assumption that the demonstration data introduces noise into the training if the assumption does not hold. They also explain how their expert data have approximately 10% noisy demonstrations, but that the agent is still able to improve learning compared to regular RL.

2.4.5 Learning From All Vehicles

Learning From All Vehicles [36] is the third and most recent submission by Chen et al, submitted in March 2022. Their approach was ranked first on the CARLA leaderboard until an anonymous submission outperformed them on May 7th, 2022. Their goal is to build a driving policy that maps sensor readings, high-level commands, and vehicle states to actions. They also aim for this approach to be deterministic. To do this they train their model in a three pipeline with three stages.



(a) Perception training. (b) Privileged motion models training. (c) Final distillation.

Figure 2.4: The training pipeline created by Chen et al. **a)** trains a perception model to map sensor inputs to a map-view feature representation of the environment. **b)** trains a motion planner using traces all nearby vehicles as input. The training is supervised using future trajectories from driving logs. **c)** The sensory model from **a)** is combined with the motion planner from **b)** using policy distillation. The figure is from the original paper [36].

Stage 1 The first stage in the pipeline is to train a perception model that is able to map sensory input to a map-view feature representation of the surroundings. The sensors used are three RGB cameras and one LiDAR attached to the ego vehicle. The RGB images and the LiDAR readings are combined to provide a map-view feature representation with shape $W \times H \times C$, for width, height, and channels respectively. The perception model is trained using both semantic segmentation losses and detection losses. Figure 2.4 **a)** illustrates the training of the perception model. As an additional goal, the model is trained to create features that are indistinguishable between the ego vehicle and other vehicles. This is to create a richer feature representation than using only the ego vehicle. The outputs from the trained perception model will later be used as input to the motion planner, trained in stage 2.

Stage 2 In the second stage Chen et al. train a motion planner that takes the predictions from the perception model trained in stage 1 as input and predicts future waypoints that the vehicles should steer towards. As the features from the perception model are indistinguishable between the ego vehicle and other vehicles, ground truth data about the future trajectories from all vehicles can be used as supervision during training. One challenge the authors address is that CARLA only provides ground truth information about high-level commands that the ego vehicle follows. The high-level commands for the other vehicles have to be estimated. The authors argue that using a rule-based approach inferred from future trajectories will be noisy and ambiguous. Therefore, they design their model to infer the high-level commands directly and then select the most likely high-level command. The motion planner is trained in a similar fashion as in [10] where they use a privileged planner to supervise another planner. Figure 2.4 **b)** illustrates the training of the motion planner. The privileged planner has access to ground truth features whereas the unprivileged planner only has access to the predicted features from the perception model from stage 1. Finally the models from stage 1 and stage 2 are combined using policy distillation, illustrated in Figure 2.4 **c)**.

Stage 3 The third and last stage trains two PID controllers [37] that map plans from the motion planner into actions to be executed in CARLA. The authors use one PID controller for steering and another for acceleration control. For acceleration control, they also use an additional neural network classifier for predicting traffic lights and hazards that should cause the vehicles to stop. This classifier takes in four RGB images as input, where the first three are the same as in stage 1 and the last image is an additional image for detecting traffic lights far away. The classifier is trained using braking actions from prerecorded driving logs. Motion plans from other vehicles are also used to predict potential predictions or stoppages. This is done by predicting the future trajectories for all nearby vehicles using the models from stage 1 and stage 2. These trajectories are then matched with the ego vehicle trajectory to prevent collisions.

Results The authors compare their method to other state-of-the-art methods on the CARLA leaderboard. They show how their method outperforms the previous best method, GRIAD [4], by a great margin. On the Driving Score metric, they obtain a score of 61.85. Compared to the previous best at 36.79 this is an improvement of more than 60%. They also achieve the highest Route Completion score of all submissions with 94.46 points. The Route Completion score is 32.61 points higher than GRIAD and 24.62 points better than Transfuser+ [38], the previous highest Route Completion score. The authors conclude with the fact that even though they have achieved state-of-the-art performance their system still incurs traffic infractions, and would not be safe to deploy directly in the real world.

Chapter 3

Methodology

This chapter presents the methods and implementations used in this thesis. Section 3.1 introduces the algorithm Adaptive General Reinforced Imitation (AGRI), an extension of GRI [4] created by Chekroun et al. Further Section 3.2 describe how the algorithm is implemented in Python and Section 3.3 describes the experiments conducted in this thesis.

3.1 Training the Autonomous Agent: Adaptive General Reinforced Imitation

This section presents the method used for training autonomous vehicles in this thesis. It is a slightly modified version of GRI [4] is used for training autonomous driving agents. The algorithm is an Adaptive General Reinforced Imitation (AGRI) and is shown in Algorithm 1. The underlying hypothesis and assumptions for the algorithm are presented in Section 3.1.1. Section 3.1.2 and Section 3.1.3 then provides an overview of how episodes are rolled out and how the probability p_{demo} from GRI is made adaptive to favor RL over imitation learning as the agent learns.

3.1.1 Underlying Assumption

AGRI builds on the same hypothesis that Chekroun et al. present in GRI [4], that expert demonstrations get a constant high reward. The authors assume that expert demonstrations can be seen as perfect data. In AGRI this assumption is slightly modified so that expert actions are assumed to be good actions, but not perfect. They should therefore be given a high reward, but not the maximum reward. Chekroun et al. give expert demonstrations a constant reward of 1, the maximum in their reward function. In this thesis, expert demonstrations have been given a constant reward of 0.6, 60% of the maximum possible. Section 3.2.5 describes

Algorithm 1 AGRI: Adaptive General Reinforced Imitation

```
Input: initial demonstration probability $p_{initial}$, expert demonstration reward $r_{demo}$, episode success threshold $Thresh$, demonstration probability decay factor $\gamma_p$;

initialize empty replay buffer $\mathcal{B}$;
initialize dataset $\mathcal{D}$;
$p_{demo} \leftarrow p_{initial}$;

while not converged do
    if $random.random() > p_{demo}$ then
        Perform an episode using the RL agent;
        Store the trajectory in $\mathcal{B}$;
        $R_{episode} \leftarrow \sum_i^{len(episode)} r_i$ ;
        $\Delta p_{demo} \leftarrow -\left(\frac{R_{episode}}{Thresh} - 1\right)$;
        $\Delta p_{demo} \leftarrow clip(\Delta p_{demo}, -1, 0.1)$;
        $p_{demo} \leftarrow clip(p_{demo}, 0, 1)$;
        $p_{demo} \leftarrow p_{demo} + \Delta p_{demo}$;
    else
        Sample an episode from $\mathcal{D}$;
        Store the trajectory in $\mathcal{B}$;
        $p_{demo} \leftarrow \gamma_p p_{demo}$;
    end if

    Sample minibatch from $\mathcal{B}$ and update network weights;

end while
```

how the reward function is implemented. The hypothesis is that the expert demonstrations can provide a guide to how the agent should behave while allowing the reinforcement exploration to stay relevant, even with a high r_{demo}.

3.1.2 Episode Rollouts

Algorithm 1 shows how AGRI uses p_{demo} to do a rollout of an episode using either the prerecorded expert demonstrations or the exploring RL agent. When using the expert demonstrations an episode of fixed length is added to the replay buffer $\mathcal{B}$. In this thesis, an episode length of 300 has been used for the expert demonstrations. If the rollout is performed using an RL agent, the agent explores the environment. The exploration continues until a terminal event happens, the agent reaches the maximum episode length, or the total episode reward reaches a lower threshold. A terminal event is defined as a collision, or if the agent strays too far from the lane center. For the RL agent the maximum episode length in this thesis is 1,000 steps and the lower threshold for the total episode reward is -100. The rationale for using a longer episode length for the RL agent stems from the underlying hypothesis that expert demonstrations should only be used for guiding the RL agent in the right direction, and should not be too dominating.

3.1.3 Updating the Probability for Expert Demonstrations

In their experiments with GRI [4], Chekroun et al. use a constant probability p_{demo} for selecting expert demonstrations instead of exploring the environment. In AGRI this probability is designed to adapt to how well the agent performs when exploring the environment. The p_{demo} is initialized as $p_{initial}$, given as an input to the algorithm. After rolling out an RL episode the total reward over the episode $R_{episode}$ is calculated as seen in Algorithm 1. The episode reward is then compared to a success threshold $Thresh$ that indicates the total reward needed for the episode to be considered a success. $R_{episode}$ is then compared to the threshold for success. If $R_{episode}$ is greater than $Thresh$ the probability of rolling out an episode using the RL agent should be higher and vice versa. The update to p_{demo} is shown in Algorithm 1 as Δp_{demo}. In this thesis $Thresh$ have been set to 200, 20% of the theoretical maximum reward for RL agents.

AGRI is designed to favor the RL agent, in accordance with the underlying hypothesis. To do this it uses probability update clipping to make sure that Δp_{demo} does not become too large. Using $clip(\Delta p_{demo}, -1, 0.1)$ as the clip function p_{demo} is guaranteed to increase by a maximum of 0.1 each episode. A slowly increasing p_{demo} is desirable as it prevents p_{demo} from instantly becoming 1 after a single poor episode from a potentially otherwise well performing RL agent. On the other hand it does allow p_{demo} to instantly become 0 if the RL agent manages to achieve $R_{episode} = 2 \cdot Thresh$. A limit on the maximum decrease in p_{demo} would be counterintuitive to the RL favoring nature of AGRI. A clip function is also applied to p_{demo} in the end to ensure that it stays in the interval $[0, 1]$ at all times.

Another mechanism AGRI implements to ensure that RL rollouts will occasionally happen is a probability decay that reduces p_{demo}. Every time a rollout is collected from expert demonstrations a decay factor γ_p is multiplied with p_{demo}. γ_p a parameter defined as $0 < \gamma_p < 1$ that is given as an input to the algorithm, and ensures that p_{demo} does not get stuck at 1, even if the RL agent behaves poorly over a long period of time. In this thesis $\gamma_p = 0.9$ have been used.

3.2 Implementation and Technology

This section presents the technical details describing the Python implementation of the training pipeline. Section 3.2.1 presents the simulator before Section 3.2.2 explains how this is implemented in Python using a Gym environment. The remaining sub-sections describe the action- and observation space of the environment, the reward function design, and the design of the visual subsystem and policy network.

3.2.1 Choice of Simulator

CARLA [20] was chosen as the simulator for the experimental setup. The rationale for the choice is that out of all the simulators mentioned in Section 2.3, CARLA is by far the most well-documented one. It is also one of the most widely used simulators by other researchers, making it easier to compare results. Another major reason for using CARLA is that Chen et al. have made their dataset from [36] of prerecorded expert trajectories available for the public. The dataset includes RGB images from three cameras on the ego-vehicle from different angles as well as their respective semantic segmentation labels. Using this dataset instead of collecting everything from the beginning saves several days' worth of data collection.

3.2.2 OpenAI Gym Implementation

To interact with the CARLA simulator using the Python API an environment following the OpenAI Gym interface has been used. The reason for implementing the environment as a Gym environment is that it allows for Stable-Baselines3 implementations of the RL algorithms to be used to train the agents. This ensures that the reinforcement algorithms are correctly implemented and thoroughly tested. As mentioned in Section 2.2.1 a Gym environment requires a method *reset()* that returns the initial state of the environment and a method *step()* that takes an action as input and returns the new state, the reward and if the new state is a terminal state. When resetting the environment the decision to collect data using the current exploration policy or expert data is made. The decision is based on the current value of the RL probability. Resetting the environment for an episode of RL is referred to as *rl_reset()*, and resetting the environment for an episode of imitation learning is referred to as *imitation_reset()*. Similarly, when the environment is collecting an episode by exploration the *step()* method is referred to as *rl_step()*, and *imitation_step()* when collecting from the expert data.

imitation_reset() and imitation_step() When resetting the environment for collecting an episode from expert demonstrations an episode is loaded. The episode consists of 300 consecutive states and the actions selected by the expert in each state. Stable-Baselines3 offers no way to directly access the replay buffer during training. Therefore, the states and actions have to be added one by one through the *imitation_step()* method. Luckily this is relatively straightforward to implement. The environment is already implemented to keep track of the episode length, so all that has to be done is to return the i-th state-action pair from the dataset i-th episode step. As mentioned in Section 2.2.1, *step()* is expected to return a tuple (state, reward, done). The selected action is not a part of this tuple, so the action a selected by the expert agent in state s is therefore added as a component to the state instead. Section 3.2.4 explains how this is implemented. As explained in Section 3.1.1 a constant reward of 0.6 is returned for each step, and *imitation_step()* returns $done = True$ when the 300th step is returned.

rl_reset() When resetting the environment for a RL episode a random route is selected from the Scenario Runner route configuration. The configuration file used for the training is available in Appendix A.1. It contains ten different routes in Town 1, which is used as the training town. The Scenario Runner parses the configuration file and creates a list of route points that the agent should follow. The client then sets the map to Town 1 in the server and enables synchronous mode. This ensures that the server only produces a new frame in the simulation when it is told to so that it is time-discrete. The time between each time frame in the simulation is set to 1/20 seconds so that the simulation is running at 20 frames per second. This means that the sensors and cameras also produce signals at a rate of 20 frames per second. The Scenario Runner then creates a set of scenarios for the ego-vehicle to complete and spawns all necessary vehicles and pedestrians for the scenario.

rl_step() When applying an action to the environment the action is first translated from an action in the action space to a VehicleControl defined in the CARLA Python API. The client then sends the VehicleControl to the server together with a signal to move to the next time frame. The server performs one tick and returns new values for ego-vehicle sensors. The environment then calculates the reward according to the reward function explained in Section 3.2.5, and returns the tuple (state, reward, done).

3.2.3 Action Space

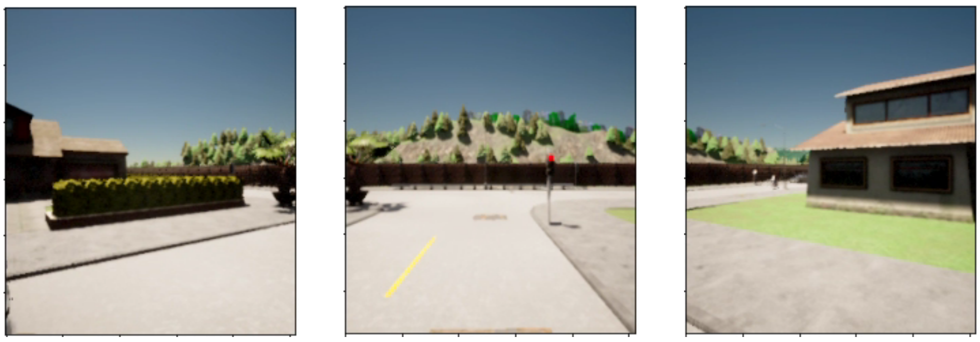
The environment comes in two flavors to support both methods that require discrete action spaces and methods that requires continuous action spaces. The discrete action space is implemented using 21 values for steering and 5 values for braking or throttling. The CARLA Python API defines the VehicleControl that controls a vehicle in the simulation. It takes as arguments *steer* in the interval $[-1, 1]$, and *throttle* and *brake* both in the interval $[0, 1]$. The 21 discrete values for steering are used to represent the following values for *steer*: $\{-1, -0.9, ..., 0.9, 1\}$. Similarly the 5 values for throttling and braking represents $\{0.5, 1\}$ for both *throttle* and *brake*, the last value is 0 which indicates no braking or throttling. In total this is $21 \times 5 = 105$ actions.

Continuous actions are easier to represent. The continuous action space is implemented as a vector with two elements $a = [x_1, x_2]$. Both elements are in the interval $x_1, x_2 \in [-1, 1]$. The first element in the vector, x_1, represents the value for *steer*. As *steer* expects a value in the interval $[-1, 1]$ it can be used directly. The second element, x_2, represent the control for the speed. Since *brake* and *throttle* both expects values in the interval $[0, 1]$ negative values are used to represent braking, so that if $x_2 > 0$ then $throttle = x_2, brake = 0$ else *throttle* $= 0, brake = x_2$. This approach ensures that the agent never brakes and throttles at the same time.

3.2.4 Observation Space

The observation space defines a standard for how the agent can expect to receive observations. The agent observes the environment through a set of sensors. The sensors used are three RGB cameras, one GNSS sensor, one IMU sensor, and one collision sensor. The collision sensor is only used for the reward function and for terminating the episode early and is not provided as an input to the agent policy. As mentioned in Section 2.2.1, complex environments such as autonomous driving environments require the use of sub-spaces to represent the different components of the environment state. In Gym environments, this is implemented as a dictionary space that is composed of simpler sub-spaces. In this thesis, there are six sub-spaces. The first three represent one RGB camera each, and the next three represent the vehicle velocity, a directional vector, and the expert demonstration action. A more detailed description of the sub-spaces is presented below. Vehicle velocity is omitted due to it being self-explanatory.

Visual Sub-Space The three sub-spaces representing the image from one RGB camera each can be seen as a combined visual sub-space. All images have sizes $H \times W \times C$ with $H = 288$ and $W = 256$ and $C = 3$. Section 3.2.4 depicts the three images that together make up the visual sub-space. All three cameras have a field of view (FoV) of 64 degrees. The left camera is rotated 60 degrees to the left and the right camera is rotated 60 degrees to the right. The center camera is faced directly ahead. The rationale for these exact measurements is based on the available dataset [36] from Chen et al. In order to use their dataset for training the visual encoder, explained in Section 3.2.6, the images have to be of the same sizes and with the same FoV.



(a) Left camera **(b)** Center camera **(c)** Right camera

Figure 3.1: The three visual inputs that the agent observes. All three are RGB images of size 288 × 256.

Directional Vector The agent is equipped with a GNSS sensor that provides geolocation data and an IMU that provides information about the vehicle rotation. By combining this information with the HD map and the route points provided by the Scenario Runner it is possible to create a normalized vector that points in the direction that the agent should drive. Geolocation data is provided as coordinates given in longitude and latitude. Using the HD map these coordinates can be translated into a xy-coordinate system, the same coordinate system used by the Scenario Runner. One possible approach is to use the xy-coordinates of the vehicle and route points directly as observations. However, this complicates learning as the coordinates are not normalized. If the current route point is at $[0,0]$ and the next route point is at $[0,1]$ it will look completely different than if the current route point is at $[100,100]$ and the next route point is at $[100,101]$. In both cases, the agent is tasked to drive towards a point that is 1 meter ahead, but the input to the agent is different. The same applies if the current route point is at $[0,0]$ and the next route point is 1 meter ahead at $[1,0]$. The location and rotation of the vehicle influence the representation for the same task. To handle this issue a normalized directional vector relative to the vehicle is used to point the vehicle in the correct direction.

The directional vector is calculated by first calculating the distance vector between the vehicle and the next route point. The distance vector is in a vector space with basis $\mathcal{B}_{world}$ that is independent of the vehicle rotation. This is undesirable, as rotating the vehicle 10 degrees to the left from the desired direction should rotate the directional vector 10 degrees to the right. The directional vector should therefore be relative to the vehicle, and not to the world coordinates. To handle this a change of basis is applied to the distance vector between the vehicle and the next route point. The basis is changed from $\mathcal{B}_{world}$ to $\mathcal{B}_{vehicle}$ so that the distance vector is now relative to the vehicle. This vector is then normalized by dividing with the L2 norm and now represents the directional vector. Figure 3.2 displays a visual representation of the directional vector for the vehicle while driving straight and while turning. Note how the vector directs the vehicle to turn left by pointing close to, but not directly at, the next route point.

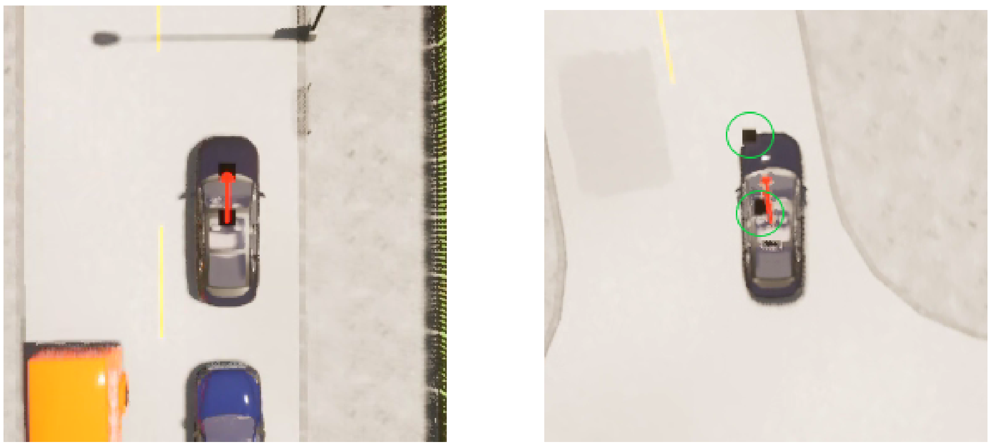


Figure 3.2: The images shows the directional vector that points to where the vehicle should drive as a red arrow. The two black squares (in the green circles in the right image) represent the current route point and the next route point.

Expert Actions As mentioned in Section 3.2.2 there is no way to insert the expert demonstrations directly into the replay buffer directly. Therefore, the expert actions have to be added as a component to the state in the *imitation_step()* method. This is implemented by defining a sub-space for actions in the observation space. As explained in Section 3.2.3 the environment comes in two flavors to handle both discrete and continuous actions. The same therefore has to be done with the observation sub-spaces for expert actions.

The observation sub-space for discrete action space is implemented as an integer in the interval $[-1, 104]$. The value -1 is used to indicate any action from the RL agent, and the remaining 105 actions are used to represent the discrete action values from the expert demonstrations. During the episode rollout, when the agent policy receives an observation containing an action value that is not -1 it simply selects the action instead of passing the state through the neural network. This way the action is stored correctly in the replay buffer together with the next state. If the action value in the observation sub-space is -1 then the policy will behave normally and pass the state to the neural network.

For the continuous action space a vector of three elements $a_{obs} = [x_1, x_2, x_3]$ is used to implement the observation sub-space. Here x_3 is added in addition to the two elements in the action space. $x_3 = 1$ indicates that x_1, x_2 comes from expert demonstrations and $x_3 = -1$ indicates that the system is collecting experience with the RL agent. The same way as with the discrete actions the agent policy will use expert demonstration actions directly, bypassing the neural network.

3.2.5 Reward Function Design

The reward function used for the RL agent is heavily inspired by the reward function Toromanoff et al. present in [28]. It uses the same three components desired speed, desired position, and desired rotation to calculate the reward so that

$$R = R_{speed} + R_{position} + R_{angular}. \tag{3.1}$$

To calculate R_{speed} the desired speed first needs to be calculated. The desired speed is defined, with a slight abuse of notation, as:

$$speed_{desired} = min\left(\begin{cases} 0\ m/s, & if \text{ red light ahead} \\ (d-5)/3\ m/s, & if \text{ vehicle ahead} \\ 30/3.6\ m/s & otherwise \end{cases}\right) \tag{3.2}$$

Here d is the distance to the leading vehicle. The distance is subtracted by 5 and divided by three so that the agent should keep a 3 seconds distance and stop 5 meters from the leading vehicle. If there is a red light affecting the vehicle it should stop, and otherwise, it should keep a speed of 30km/t (30/3.6 m/s). The speed reward is based on the relationship between the current speed and the desired speed. It is calculated as follows:

$$R_{speed} = \begin{cases} max(1 - abs\left(1 - \frac{10 \cdot speed_{current}}{speed_{desired}}\right), -1), & speed_{current} > speed_{desired} \\ \frac{speed_{current}}{speed_{desired}}, & speed_{current} < speed_{desired} \\ 0, & speed_{desired} < 1 \end{cases} \tag{3.3}$$

The speed reward component is in the interval $[-1, 1]$. Negative values first occur if the current speed is higher than 10% of the desired speed and is capped at -1. This is intended to learn the agent to not drive recklessly fast. If the agent is driving slower than the desired speed the speed reward is equal to the fraction of the desired speed that the agent is driving. To avoid the agent from seeking out and stopping in areas where the desired speed is zero R_{speed} is set to zero if the desired speed is less than 1 m/s.

The positional reward $R_{position}$ is based on the distance from the vehicle to the center of the lane. Let d here denote this distance and D_{max} denote the maximum distance the agent is allowed to be from the lane center before the episode terminates. The positional reward $R_{position}$ is then calculated as

$$R_{position} = -\left(\frac{d}{D_{max}}\right)^2. \tag{3.4}$$

The positional reward is in the interval $[-1, 0]$. It is 0 if the agent is exactly in the middle of the lane and -1 if the agent is 2 meters or more from the lane center. Similar to Toromanoff et al. in [28] the episode is terminated if the agent is further from the lane center than 2 meters so that $D_{max} = 2$. The positional reward in this thesis uses the square of the distance divided by the maximum allowed distance. The rationale for this is that the agent should be allowed to deviate from the lane center to some degree without receiving too much negative reward. Dividing by D_{max} ensures that the squared expression is in the interval $[0, 1]$. The reward is therefore less punishing for small deviations from the lane center than if a linear expression was used because $x \geq x^2$ for $x \in [0, 1]$.

The last reward component is the angular reward. It uses the waypoint API to calculate the angle in radians between the lane and the vehicle. The angular reward component is then calculated using the absolute value of the angle as follows:

$$R_{angular} = -\frac{angle}{\pi} \tag{3.5}$$

As the absolute value of the angle is at most π and at least 0 the angular reward is in the interval $[-1, 0]$. In addition to the three reward components, the agent is rewarded -5 for colliding with anything.

3.2.6 Visual Encoder

Similar to [4, 28, 36] a semantic segmentation encoder is trained like an autoencoder to predict the semantic segmentation labels. The encoder first transforms the input image into a lower-dimensional vector. This vector representation is later used as a part of the state representation. The decoder then uses the encoded vector representation to reconstruct the semantic segmentation labels. Figure 3.3 displays the overall architecture for the visual autoencoder with the encoder component, the encoded representation, and the decoder component.

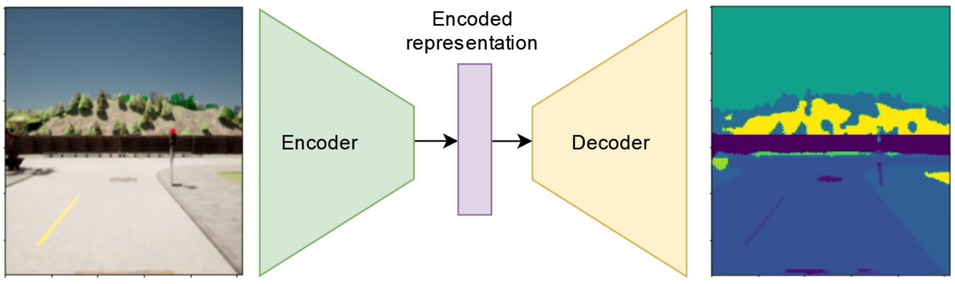


Figure 3.3: A visual representation of the semantic segmentation autoencoder. The encoder transforms the input image into a lower dimensional vector that is used by decoder to reconstruct the semantic segmentation labels in the image.

The visual encoder is trained in a supervised fashion using the dataset provided by Chen et al. [36] with a cross-entropy loss. Toromanoff et al. argue in [28] that augmenting the viewpoint of the images is important as the expert demonstration agent is always in the center of the lane, and that therefore, the dataset does not have enough variation in viewpoints to generalize well to the RL agent, whose images are much more varied. They do however use only a single center camera for training their model. The dataset by Chen et al. consists of images collected by three cameras placed at different angles. This is assumed to be enough variation so that the viewpoint augmentation is not needed. Images are instead augmented by randomly flipping the images horizontally, and randomly adding jitter and gaussian blur.

The backbone used in the semantic encoder is the ERFNet [39] created by Romera et al. Using the input sizes of $288 \times 256 \times 3$ the ERFNet outputs an encoded feature map of size $36 \times 32 \times 128$, where 128 is the channel dimension. Using ERFNet as the encoder backbone is partly based on the fact that it was used by Chen et al. in Learning From All Vehicles [36], the best performing driving agent on the CARLA leaderboard with an open-source implementation. The other main reason it was chosen is that it is one of the fastest semantic encoders available with state-of-the-art performance. The time for inference in the visual encoder is critical as this is the bottleneck in the entire training pipeline. During inference, this feature map is averaged over the channel dimension using adaptive average pooling to produce a vector of size 128. Wyk et al. explain adaptive average pooling: in [40] "*Adaptive average pooling is simply an average pooling operation that, given an input and output dimensionality, calculates the correct kernel size necessary to produce an output of the given dimensionality from the given input.*"

3.2.7 Policy Network

The policy network is responsible for mapping the inputs to actions. The RGB images are passed sequentially through the same visual encoder before the outputs are concatenated with the directional vector and the agent velocity. While training the RL agent the weights of the visual encoder are frozen as is [4, 28]. The resulting encoded state representation is a vector of size 387 that is then passed to the policy head. Figure 3.4 depicts an illustration of the policy network.

As will be explained in Section 3.3 both DQN [12] and TD3 [15] have been used as RL algorithms in this thesis. Both algorithms have a policy head that consists of two fully connected layers of sizes 400 and 300 respectively. In Stable-Baselines3 [13] this is the default for TD3, and the same architecture have therefore been used in the DQN architecture for comparability. The shape of the output action depends on the algorithm used. DQN outputs a discrete action while TD3 outputs a continuous action as described in Section 3.2.3.

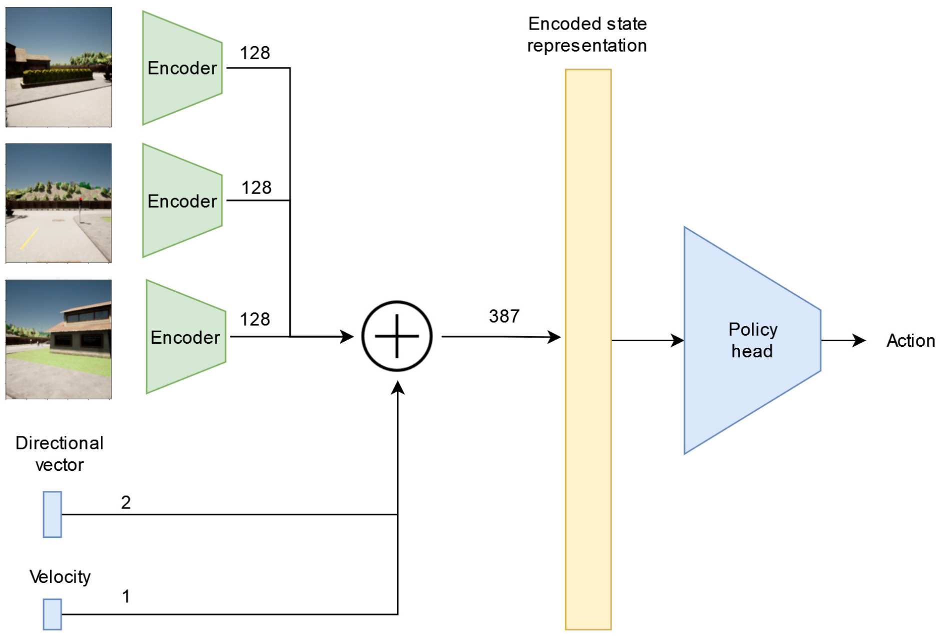


Figure 3.4: The policy network. The RGB images are encoded using the same visual encoder. The three outputs are then concatenated together with the directional vector and the velocity of the agent. The resulting vector is the encoded state representation of size 387. This vector is then passed to the policy head that outputs an action.

3.2.8 Expert Data Collection

The Scenario Runner, explained in Section 2.3.1, is used for collecting the expert demonstration dataset. The CARLA Python API comes with functionality for creating agents that can drive using an auto-pilot. The auto-pilot uses privileged information from the server about other vehicles and pedestrians to drive with a rule-based approach. This allows the agent to exhibit a near-perfect behavior most of the time. For unknown reasons the auto-pilot does however sometimes get stuck or collides with other vehicles. This phenomenon is also reported by Chekroun et al. in [4] where they estimate that $\sim 10\%$ of the expert demonstrations are noisy. The privileged agent is equipped with the same set of sensors as the RL agent, mentioned in Section 3.2.4. The resulting dataset consists of a total of 54,434 samples collected from ten different routes in Town 1 with the same weather. Using episode lengths of 300 for expert demonstrations this totals 181 episodes of expert data.

3.3 Experiments

This section presents the experiments that have been conducted. Section 3.3.1 present the experiments used to test AGRI. Both the value-based approach and the policy gradient approach are presented. Section 3.3.2 presents how the first experiments are repeated, using a kinematic bicycle model to augment the encoded state representation.

3.3.1 Experiment 1a and 1b: Using AGRI with Value Based Methods and Policy Based Methods

The first part of experiment 1 is to test AGRI using a value-based approach with discrete action space. DQN have been chosen as the RL algorithm for this as it is the most commonly used in similar research [4, 28]. The second part of experiment 1 is to test AGRI using a policy gradient approach with continuous action space. TD3 have been used for this task. The reason for choosing TD3 is that it is the most recent of the three off-policy algorithms provided by Stable-Baselines3, and from [15] it also seems to perform better than DDPG in general. The most important hyperparameters and specifications are given below:

Training length Due to a limited amount of time the agent is trained for only 500,000 time steps in the environment. The simulation runs at approximately 3 frames per second so training a model takes up to 46 hours. Models are trained for at most 100,000 time steps before the model and replay buffer is saved and the training is paused. Training the models for longer tended to cause the CARLA server to freeze. Restarting the training by saving and loading the model and replay buffer every 100,000 time steps helped prevent this freezing.

Buffer size The buffer size is set to only 10,000 samples. The small size is due to memory limitations, as observations stored in the replay buffer consist of three relatively large RGB images. Chapter 5 discusses this issue further.

Discount factor The discount factor γ is set to 0.9. The default from Stable-Baselines3 is 0.99, but early experimenting indicated that $\gamma = 0.9$ helped the agent learn faster. A hypothesis is that as the agent is trained for relatively few iterations it helps learning by focusing more on immediate rewards than rewards further into the future.

Experiment 1a: Training with DQN The behavior network is updated by gradient descent after every episode. The update is done by sampling a mini-batch from the replay buffer and doing one step of gradient descent for every step the agent has taken in the environment. A mini-batch has a size of 16 samples, so if the agent reaches the maximum episode length of 1,000 steps it is trained for a total of 1,000 steps of gradient descent on 16,000 samples. The target network is updated every 2,000 steps by copying the weights of the behavior network. The learning rate is set to a constant of 0.0001 and the optimizer used by default is the Adam optimizer [41]. The DQN implementation uses an ϵ greedy policy that selects a random action instead of the action with the highest predicted value with probability $p = \epsilon$. In the beginning, the model selects actions mostly at random, with an initial exploration rate set to $\epsilon_{initial} = 0.9$. The exploration rate decreases as the agent explore the environment and after 90,000 steps it reaches the minimum value of $\epsilon_{minimum} = 0.05$.

Experiment 1b: Training with TD3 To properly compare the results from **RQ1** and **RQ2** it is desirable to use similar specifications for TD3 as for DQN. The same mini-batch size, learning rate, and optimizer are therefore used. The behavior network for TD3 is also updated every episode, similar to the experiment with DQN. The delay for updating the policy is set to two episodes. If agents reach the maximum of 1,000 steps per RL episode the update will be at approximately the same interval as the target update in experiment 1a. The rest of the hyperparameters are set as the defaults from Stable-Baselines3. This includes $\sigma = 0.2$ and $c = 0.5$ for σ and c from Equation (2.25).

3.3.2 Experiment 2a and 2b: Augmenting the State Representation With a Kinematic Bicycle Model

The kinematic bicycle model [34] is a mathematical model of vehicle dynamics. Polack et al. [34] shows how the model can be written as:

$$
\begin{aligned}
\dot{x} &= v \cdot cos(\theta + \beta) \\
\dot{y} &= v \cdot sin(\theta + \beta) \\
\dot{v} &= a \\
\dot{\theta} &= \frac{v}{r_b} sin(\beta) \\
tan(\beta) &= \frac{r_b}{f_b + r_b} \tan(\phi)
\end{aligned}
\tag{3.6}
$$

The same notation used by Chen et al. in World on Rails [33] is used. $\dot{x}, \dot{y}, \dot{v}, \dot{\theta}$ represents the change in position in the x-direction and y-direction, change in velocity, and change in rotation of the vehicle body respectively. r_b, f_b, and ϕ represent the location of the rear and front wheelbases, and the front wheel steering angle respectively. β will then be the slip angle at the center of gravity of the vehicle. Some of these variables are unknown for the vehicle in CARLA. Chen et al. solved this in World on Rails [33] by implementing the model as a very simple parameterized neural network. Their implementation and trained model is used in this thesis to represent the vehicle dynamics of the vehicle.

To answer **RQ3** almost the exact same experiments as experiments 1a and 1b are repeated. The only difference is that the encoded state representation explained in Section 3.2.6 is augmented with the outputs from a kinematic bicycle model. This is to add knowledge about the vehicle dynamics to the state representation. A pretrained model is used with frozen weights, similar to the visual encoder. The weights used for the model are from Chen et al. and their work with World on Rails [33]. It takes as input the current vehicle position, current rotation, current velocity, and action and predicts the next location, rotation, and velocity if the agent executes the action.

For each of the 105 discrete actions, the kinematic bicycle model calculates the predicted outputs. To do this the input velocity is replicated 105 times and represented as a vector. It is desirable for the output from the bicycle model to be relative to the vehicle, similar to the directional vector. The vector representations for locations and rotations can therefore be constant and equal to zero as if the vehicle was starting in origin in the coordinate system with zero rotation. For each of the actions in the discrete action space, the bicycle model produces a vector of size 4 to represent the predicted location, rotation, and velocity. Before being passed to the bicycle model the actions are translated from discrete integers to a vector representation as described in Section 3.2.3 with three elements representing *steer*, *throttle* and *brake*. Doing this for all 105 actions produces a vector of size $4 \times 105 = 420$ that is flattened and concatenated together with the encoded image representations, the directional vector, and the velocity. The resulting state representation is then a vector of size $3 \times 128 + 2 + 1 + 4 \times 105 = 807$. Figure 3.5 illustrates the policy network with the kinematic bicycle model.

The underlying hypothesis for using the kinematic bicycle model is that it enriches the state representation by adding an understanding of how the vehicle behaves to input actions. The same discrete actions are used as input for the continuous action space policy. It would be impossible to apply the bicycle model to the infinitely large continuous action space. An assumption is that the continuous agent will be able to learn from the augmented state representation. In theory, any discretization of the action space should be eligible to use as input to the bicycle model as it is not directly used as output, but merely as a part of the state representation.

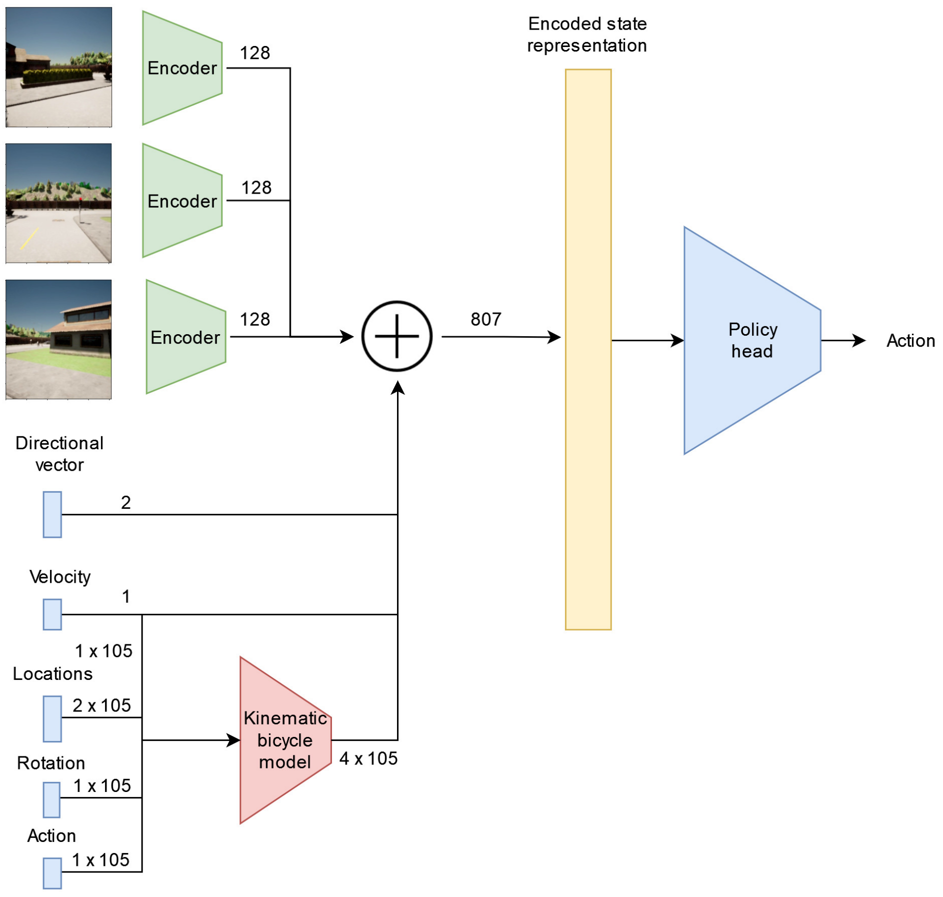


Figure 3.5: An augmentation of the encoded state representation. In addition to the encoded images, velocity, and directional vector a kinematic bicycle model has been added. The kinematic bicycle model uses the vehicle velocity together with constant predefined actions to calculate the relative location and rotation of the vehicle by applying each action.

Chapter 4

Results

This chapter presents the results obtained from the experiments conducted in this thesis. Section 4.1 presents the intermediate results obtained from training the semantic segmentation encoder before Section 4.2, 4.3, 4.4 and 4.5 presents the results from training the models described in Section 3.3. In the end Section 4.6 presents the results from evaluating the agents in an unseen environment.

4.1 Visual Encoder Results

Figure 4.1 and Figure 4.2 displays input images and their predicted labels. In Figure 4.1 the left image is the RGB input, the center image is the ground truth semantic labels and the right image is the predicted labels. The image is sampled from a separate dataset that was collected with the sole purpose to be used for evaluation. It is not part of the Learning From All Vehicles dataset [36] and has therefore never been seen before by the visual encoder. In Figure 4.2 images are sampled from the dataset containing expert demonstrations. Therefore, it does not contain any ground truth labels. However, it is still possible to visually observe the consistency in the predictions.

The results from training the visual encoder are best evaluated by analyzing the reconstructions. 1,000 images from a custom collected dataset were used in the evaluation. The reconstructions are evaluated using mean accuracy and the intersection over union (IoU) as the metrics. When evaluated on the 1,000 images the visual encoder achieved a mean accuracy of 0.9525 and a mean IoU of 0.9172. The mean accuracy indicates that on average 95.25% of the pixels in the image are correctly classified, and the high IoU score indicates that the visual encoder does not achieve the high accuracy by being overly confident in predicting classes of large sizes. Simply predicting "road" for the bottom half of the image and "sky" for the top half of the image would most likely result in a relatively high accuracy score, but the IoU would be incredibly low. Therefore, it is reasonable to believe that the visual encoder is able to represent the state of the visual input to a satisfying degree.

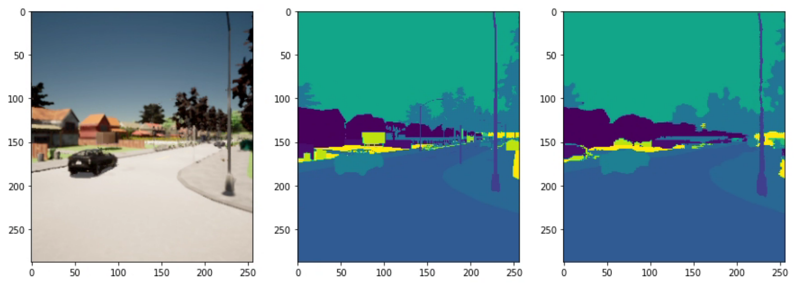


Figure 4.1: The figure displays the results of training the visual encoder. The left image is the RGB input to the visual encoder, the center image is the visual representation of the ground truth semantic labels, and the right image is the predicted ground truth semantic labels. The input image is sampled from a custom collected dataset and was not part of the training data.

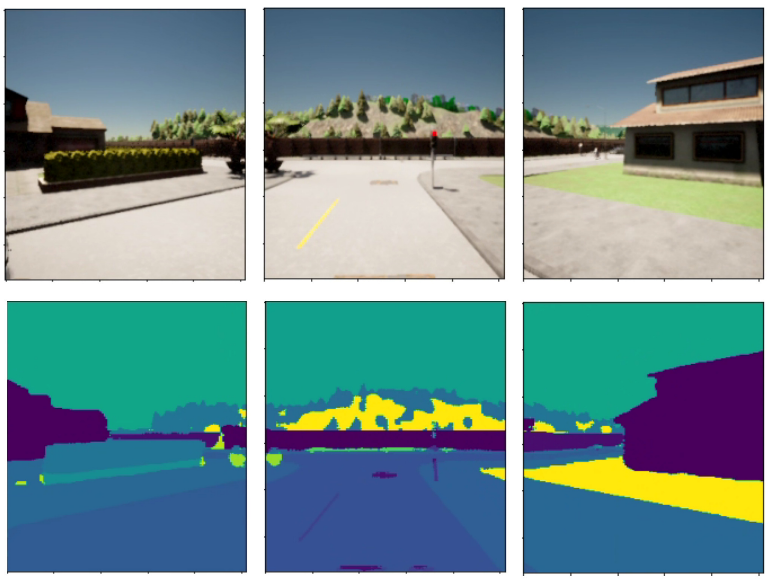


Figure 4.2: The figure displays the reconstructed labels predicted by the trained visual autoencoder. The inputs are sampled from the expert demonstration dataset and therefore does not have a ground truth image available.

4.2 Experiment 1a: Using AGRI With DQN

Figure 4.3 displays all the results from training the autonomous driving agent using AGRI together with the results from training the same agent using vanilla RL as a baseline. Figure 4.3a and 4.3c displays the episode reward and the value loss for the vanilla RL agent, and Figure 4.3b, 4.3d and 4.3e displays the reward, loss and proportion of imitation learning episodes for the AGRI agent. All plots use the number of time steps as the horizontal axis, and all plots are smoothed using a smoothing factor of 0.8 in Tensorboard[1].

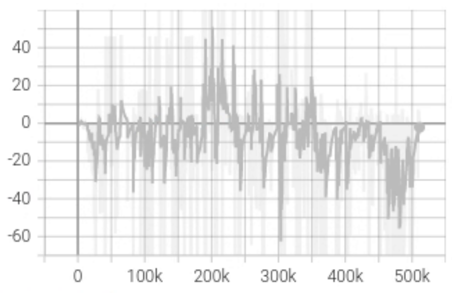
Based on the rewards plotted in Figure 4.3a and 4.3b it is safe to say that the agent did not learn to consistently score a high reward, neither for the vanilla RL agent or the agent trained with AGRI. The vanilla RL agent tends to achieve points closer to zero compared to the AGRI agent. A hypothesis is that this agent more quickly learns to stand still, to not receive a negative reward, or to drive straight forward for a short period of time before causing a collision or driving off the road.

The AGRI agent will not learn to stand still the same way, as this would lead it to be constantly trained on the expert demonstration dataset. A large partition of the expert demonstration data consists of states where the expert is driving forward, and training mostly on this data should in theory lead to the agent learning to drive forward in some way. From the reward plot, it is evident that the agent did not learn to stand still, but it did not learn any optimal policy either. Visually inspecting the agent when it is training gives an indication that it learns to drive forward for a short period of time, but then stops for unknown reasons. Due to the design of the reward function the agent receives a negative reward in every time step until the episode times out unless it is perfectly aligned in the lane center. The AGRI agent is therefore rewarded less than the vanilla RL agent that goes off the road after driving forward. This is one of the flaws in the reward function and is discussed further in Chapter 5. The low rewards achieved by the AGRI agent are reflected in the proportion of imitation learning episodes in Figure 4.3e. Approximately 60% of all episodes are rolled out using imitation learning, and the imitation learning decay explained in Section 3.1.2 is the only mechanism that enables the agent to occasionally explore with RL.

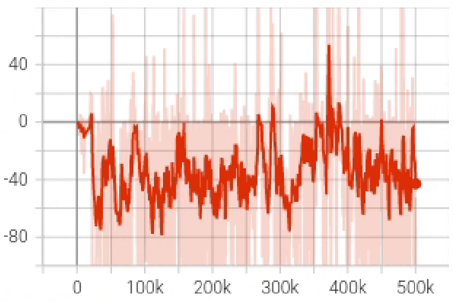
Both the plots for the value losses in Figure 4.3c and 4.3d further supports the claim that the agents are not able to learn an optimal policy. For the vanilla RL agent the loss has a slight downward trend at the end, but the very low values for the loss could indicate that the agent is stuck in a local optimum. Agents can at best achieve a reward of 1 in a single time step. The discounted value of the first state should then be approximately 10, using a discount factor for future values

[1]Tensorboard (`https://www.tensorflow.org/tensorboard`) uses exponential moving average for smoothing. A smoothing factor of 0.8 means an exponential moving average where data at the current time step is weighted by 0.2 and the exponential average at the previous time step is weighted by 0.8.

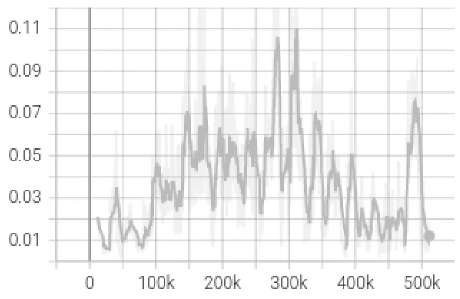
of 0.9 and a maximum of 1,000 time steps per episode. Value losses that are in the order of 0.1 together with rewards that are mostly less than zero could therefore indicate that the agents are in a local optimum. The value loss for the AGRI agent has an upward trend, indicating that it might be able to learn a way out from the local minima over time. However, it was not able to do so during the 500,000 time steps used for training.



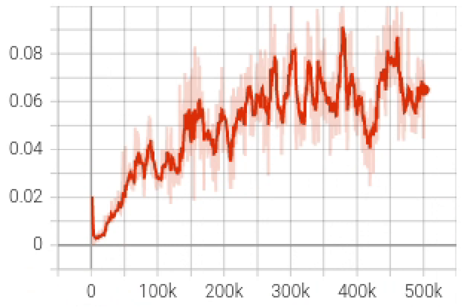
(a) Vanilla RL episode reward.



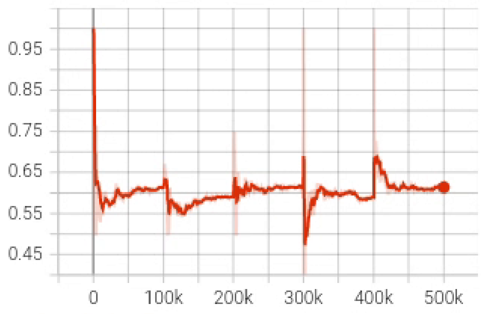
(b) AGRI episode reward.



(c) Vanilla RL DQN value loss.



(d) AGRI DQN value loss.



(e) The proportion of episode rollouts using imitation learning. 1 indicates only imitation learning and 0 indicates only RL.

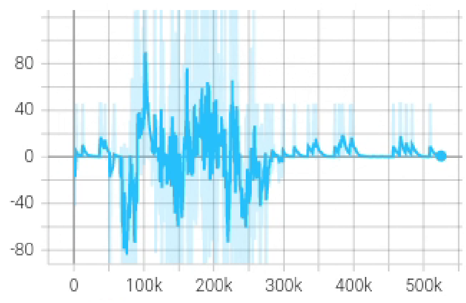
Figure 4.3: The results from training an agent using a discrete action space with AGRI are displayed in the red plots. The gray plots are from an agent trained with the exact same hyperparameters using vanilla RL. All subplots have the number of time steps as the horizontal axis.

4.3 Experiment 1b: Using AGRI With TD3

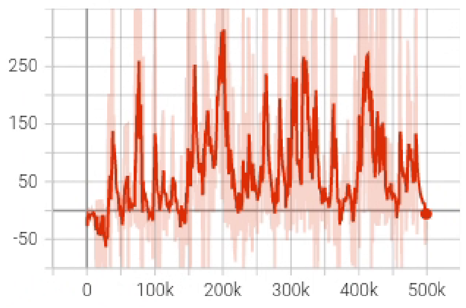
Similar to the first experiment a vanilla RL agent is trained as a baseline to compare AGRI in an environment with a continuous action space. Figure 4.4a, 4.4c, and 4.4e shows the RL agent reward, actor loss and critic loss respectively. Figure 4.4b, 4.4d, 4.4f and 4.4g shows the AGRI agent reward, actor loss, critic loss and proportion of imitation learning episodes. The horizontal axis and the smoothing factor is equal to that in Section 4.2.

Unlike in the first experiment, the agents in the second experiment manage to achieve some higher rewards. Based on the reward and losses for the vanilla RL agent it seems like the model becomes unstable at approximately 250,000 time steps. Until this point, the reward is consistently considerably higher than for the discrete approaches, but the reward goes to zero as the actor and critic losses come back to lower values. At this point, the RL agent is stuck in a local optimum where it has learned to stand still and almost always receives a reward of zero. Occasionally it manages to achieve some more reward, but this is rare compared to how often it stands still.

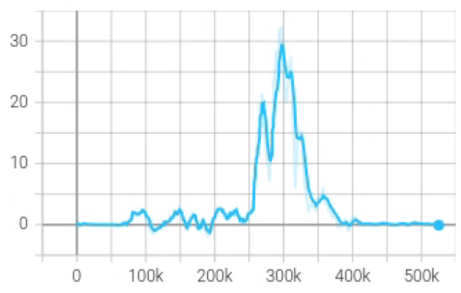
On the other hand, the AGRI agent manages to achieve a high reward with some consistency. It is however not able to fully converge, as the actor loss oscillates around zero. For high absolute values of the actor loss the rewards tends to be lower, as seen in Figure 4.4b and 4.4d. The critic network does seem to be able to converge to some degree, but as Figure 4.4f shows it does not manage to stabilize completely. Nevertheless, the AGRI agent does achieve the highest total rewards by a great margin. Due to the smoothing of the plots, the maximum values for the rewards are not visible, but for the vanilla RL agent the maximum reward during training was 338.9, whereas the AGRI agent achieved a maximum of 599.7. Due to the inconsistencies in reward the proportion of imitation learning never converges to zero, but instead reaches a minimum at approximately 30% imitation learning (The dips in Figure 4.4g around 100k and 200k time steps are caused by the resetting of the training every 100k time step as explained in Section 3.3.1. The logging of the metric for the imitation proportion is not saved and is therefore highly sensitive to the first few episodes. It does however become representative after some episodes).



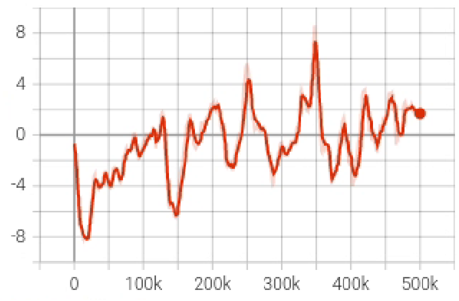
(a) Vanilla RL episode reward.



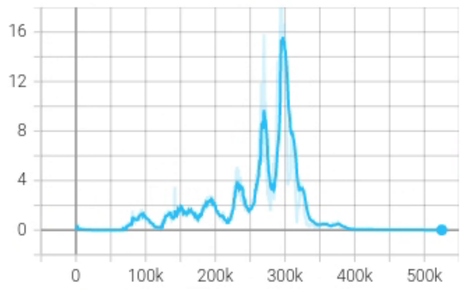
(b) AGRI episode reward.



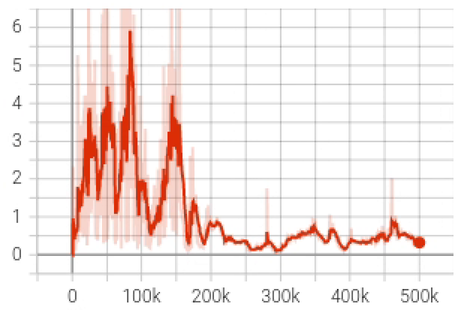
(c) Vanilla RL actor loss.



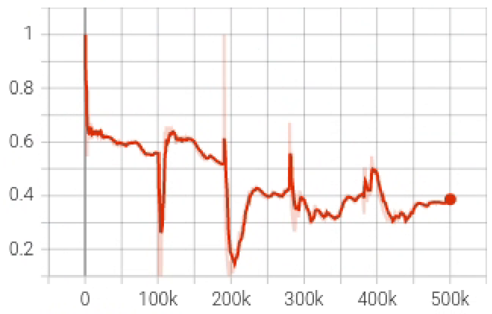
(d) AGRI actor loss.



(e) Vanilla RL critic loss.



(f) AGRI critic loss.



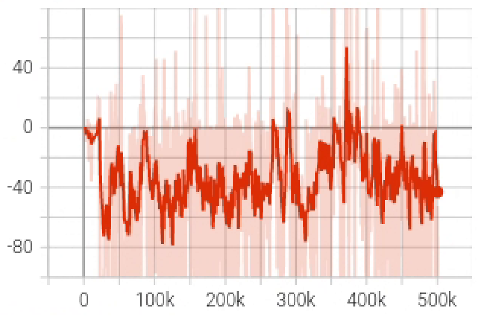
(g) The proportion of episode rollouts using imitation learning. 1 indicates only imitation learning and 0 indicates only RL.

Figure 4.4: The results from training an agent using a continuous action space with AGRI are displayed in the red plots. The blue plots are the results from an agent trained with the exact same hyperparameters using vanilla RL. All subplots have the number of time steps as the horizontal axis.

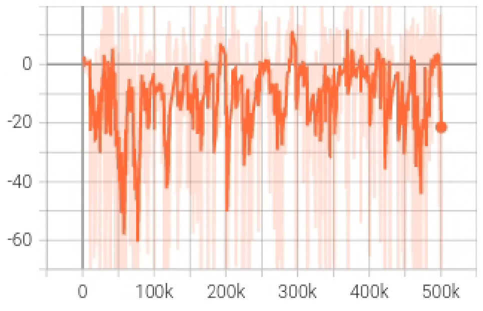
4.4 Experiment 2a: Augmenting the State in the Value Based Approach

As explained in Section 3.3.2 the second experiment is to test if the state representation can be augmented using the outputs of a kinematic bicycle model. Figure 4.5 displays the results from augmenting the state in the discrete action space environment. The left column of plots displays the discrete AGRI agent trained in the first experiment as the baseline. The right column displays the results from training the same agent with the augmented state representation, all else kept equal. Similarly Figure 4.6 compares the results from using a kinematic model with AGRI in the right column to the AGRI agent trained in the second experiment.

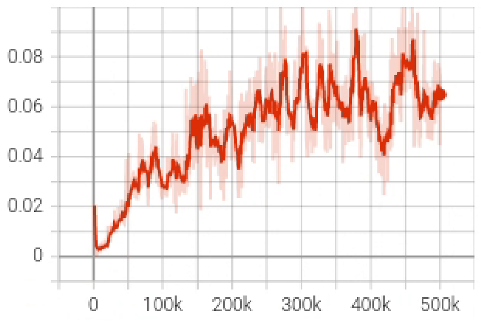
Based on the rewards in Figure 4.5a and 4.5b there is little evidence that using a kinematic bicycle model does help the agent to learn. The rewards for the kinematic agent might be slightly shifted up compared to the agent trained in the first experiment, but it still rarely manages to complete episodes with a total reward greater than zero. Both the losses and the imitation proportions are also very similar without any significant differences. The training results for the discrete action space environment indicate that augmenting the state with the kinematic bicycle model has little to no effect on the training performance.



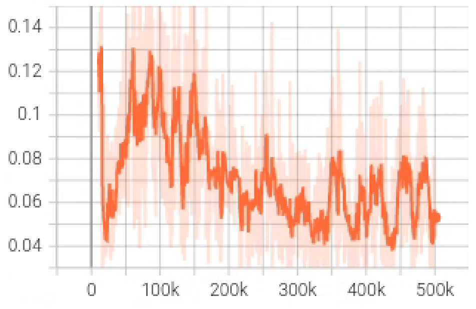
(a) Discrete AGRI episode reward.



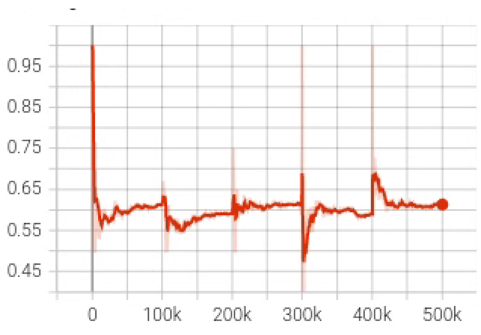
(b) Discrete Kinematic AGRI episode reward.



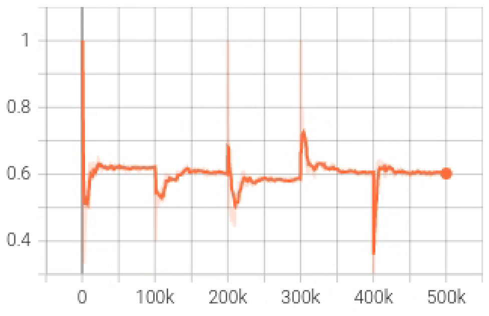
(c) Discrete AGRI DQN value loss.



(d) Discrete Kinematic AGRI DQN value loss.



(e) Discrete AGRI imitation proportion.



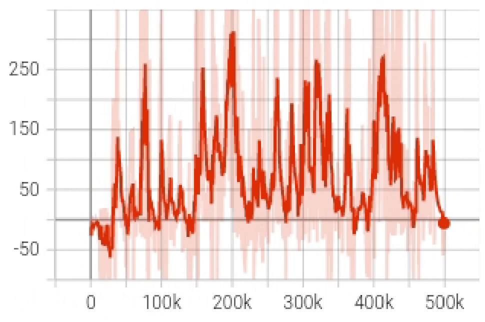
(f) Discrete Kinematic AGRI imitation proportion.

Figure 4.5: The results from training an agent using AGRI in the discrete action space environment with a kinematic bicycle model to augment the state representation.

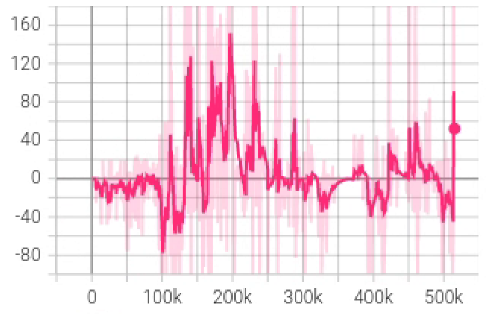
4.5 Experiment 2b: Augmenting the State in the Policy Gradient Approach

The results for the experiment in the continuous environment, displayed in Figure 4.6, also indicate that the kinematic bicycle model does not increase the performance of the agent. On the contrary, comparing the rewards in Figure 4.6a and 4.6b indicates that adding the kinematic bicycle model makes it harder for the agent to learn. Without the bicycle model, the agent is able to achieve higher rewards, more frequently, than with the bicycle model. This may be a result of the assumption from Section 3.3.2, that the continuous agent will be able to learn from the augmented state representation even if a discrete action space is used as a basis for the augmentation.

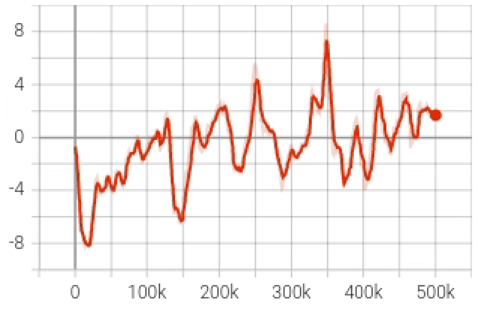
Both losses in Figure 4.6d and 4.6f also indicates that the model trained with the kinematic bicycle model performs worse than the baseline. This also goes for the imitation proportion, where the kinematic AGRI does not get lower than approximately 60% imitation learning. This could be an indication that the augmented state representation from the bicycle model adds more noise to the state representation than it enriches the state. Nevertheless, the kinematic AGRI agent does learn to some extent. Based on the training rewards it is still the second-best performing agent trained in all the experiments.



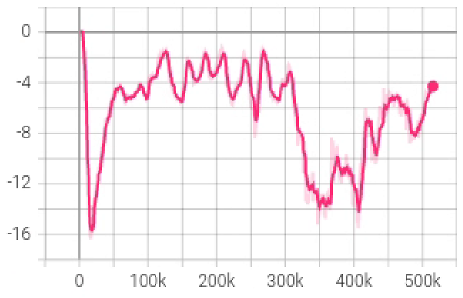
(a) Continuous AGRI episode reward.



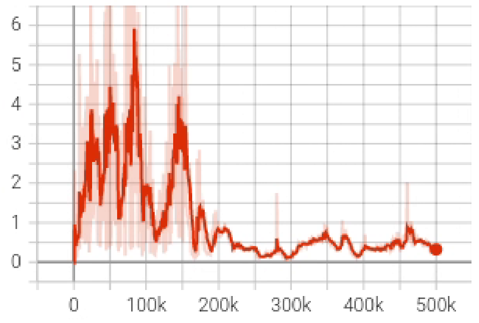
(b) Continuous kinematic AGRI episode reward.



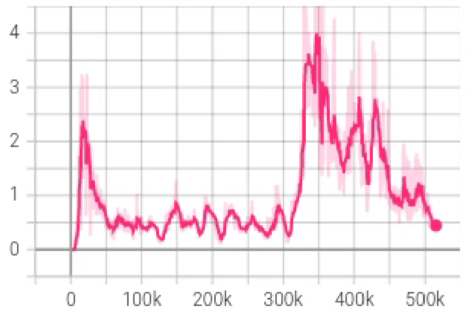
(c) Continuous AGRI actor loss.



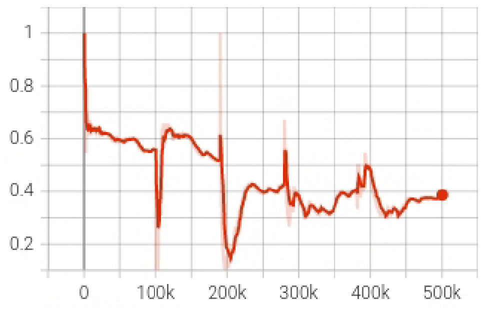
(d) Continuous kinematic AGRI actor loss.



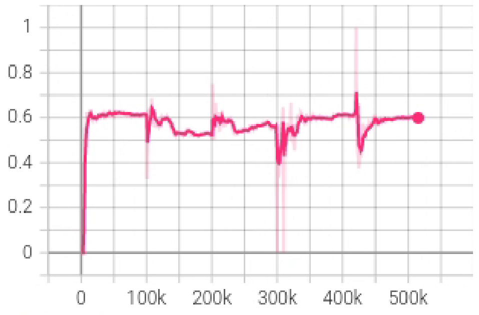
(e) Continuous AGRI critic loss.



(f) Continuous kinematic AGRI critic loss.



(g) Continuous AGRI imitation proportion.



(h) Continuous kinematic AGRI imitation proportion.

Figure 4.6: The results from training an agent using AGRI in the continuous action space environment with a kinematic bicycle model to augment the state representation.

4.6 Results in the Evaluation Environment

The previous sections have discussed how the agents trained in the experiments have performed in the training environment. To see if the agents are able to generalize, and not simply learn the training environment, they have to be evaluated in a different environment. During training, the agents were only exploring the world of Town 1 in CARLA. The evaluation will therefore be done using Town 4 so that the agents have never seen the environment before. The agents are evaluated by completing ten routes in the environment. The configuration file for the routes used in the evaluation is available in Appendix A.2.

4.6.1 Evaluation Metric

When evaluating the agents the flaw in the reward function mentioned in Section 4.2 became very clear in scenarios where the agents have to stop and wait for the vehicle in front. With the current reward design the agents have to stop very close to the lane center and at the exact right angle to not get negative rewards for several time steps at a time. This makes it likely that the agent will reach the lower threshold of -100 total reward, and the episode terminates. The result of this flaw in the reward function design is that the agents that do not move for the entire episode receive a total reward of 0, whereas the agents that drive reasonably well could very well be rewarded -100.

To solve this problem, a different metric is used to evaluate the agent performance in the evaluation environment. The new metric is simply how many points along the route that the agent manages to pass in an episode. Route points are approximately 1 meter apart, and the reward can therefore almost be viewed as the distance traveled. Colliding is still treated as a terminal event and will stop the episode early. The terminal event of straying more than 2 meters from the lane center is slightly relaxed, to see if agents can recover from their mistakes. Therefore, the episode does not terminate before the agent has been more than 2 meters from the lane center for more than 4 seconds.

4.6.2 Evaluation Episode Length

As one time step in the environment corresponds to 1/20 seconds in the simulator, an episode length of 1,000 time steps corresponds to 50 seconds of driving. The evaluation town, Town 4, features highways and routes that are much longer than the routes in the training town. To better evaluate how well the agents perform, the maximum episode length used in the evaluation is set to 4,000 time steps, corresponding to 200 seconds of driving. For most of the agents trained this is enough time to either crash, leave the lane for a long enough time or get stuck standing still.

4.6.3 A Note on Time Steps Used in Training

Note that some of the result plots in Section 4.2 and Section 4.3 have plots that go slightly longer than to 500,000 time steps. This is related to the problems with the CARLA server freezing, and when training in intervals of 100,000 time steps from the last checkpoint before the failure some agents were trained slightly longer than 500,000. However, all evaluations are completed with agents trained for exactly 500,000 time steps, as checkpoints are saved every 10,000 time steps as a backup.

4.6.4 Evaluation Results

The mean, maximum and minimum episode rewards, as well as the mean episode length from the evaluations, are displayed in Table 4.1. Both agents trained in the environment with continuous action space using AGRI greatly outperform all the other agents by approximately two orders of magnitude. The results indicate that the agents trained in experiments 1a and 2a did not manage to learn a working policy at all. They either stand completely still or leave the lane almost immediately and are outperformed by the vanilla RL baseline.

On the other hand, the agents trained in experiment 1b and 2b does manage to learn policies that can drive longer distances. As displayed in Table 4.1 the AGRI agent trained in experiment 1b outperforms all the other on all metrics. The highest mean reward indicates that the agent consistently drives longer than the other agents. The highest maximum reward indicates that it performs best at peak performance, and the highest minimum reward indicates that it is better at its worst than the other agents at their worst. A video of this agent driving in the evaluation environment can be found here: `https://youtu.be/Cmje6RrnWCo`. The bird view camera in the video is added to provide an overview only and is not a part of the observation space.

The AGRI agent trained with the kinematic bicycle model achieves only approximately 60% of the performance compared to the agent from experiment 1b on most metrics. However, it still performs more than 10 times that of the third-best agent trained. This could be an indication that AGRI might be a valid method for increasing performance when training RL agents, whereas augmenting the state representation with a kinematic bicycle model seems to add more noise to the state than it adds value.

Table 4.1: Rewards obtained in the evaluation environment using the evaluation reward

	Mean Reward	Max Reward	Min Reward	Mean Episode Length
DQN RL	14.90	59	0.00	145.10
DQN AGRI	1.90	7.00	0.00	32.90
DQN Kinematic AGRI	6.30	7.00	5.00	88.40
TD3 RL	0.00	0.00	0.00	101.00
TD3 AGRI	**770.20**	**1,338.00**	**122.00**	**2,252.20**
TD3 Kinematic AGRI	380.90	1,054.00	117.00	1,466.10

Chapter 5

Discussion

This section discusses the results obtained from the experiments. Section 5.1 discusses the results from conducting experiment 1a and answers the research question **RQ1** based on the results. Section 5.2 then discusses the results from experiment 1b and answers **RQ2** before Section 5.3 discusses the results from experiment 2a and 2b and answers **RQ3**. In the end Section 5.4 discusses shortcomings of the implementations in this thesis.

5.1 Experiment 1a

When evaluating the agents trained using AGRI with a value-based RL algorithm it is clear that AGRI does not improve learning at all. On the contrary, the results displayed in Table 4.1 show that the vanilla RL approach is able to achieve a higher mean reward, and a higher maximum reward, than both the other AGRI agents trained with the same RL algorithm.

A question that arises is how the agents trained with AGRI using the policy gradient approach outperformed the AGRI agents using the value-based approach with such a great margin. One explanation could be that the discretization of the action space is not fine-grained enough. In this thesis, 21 values have been used to represent steering, allowing the agent to turn in steps of 10% in either direction or go straight. 5 values have been used to control the speed, leaving 2 values each for braking and throttling and 1 value for neither. Similar works, such as Toromanoff et al. [28] and Chekroun et al. [4], have used 27 values for steering and 2 or 3 values for throttling or braking. It does however seems unlikely that this difference should make such an impact on performance.

Another possible could have been that the learning rate was too high or too low for the agent to properly learn. However, the most probable cause of the large difference between the value-based approaches and the policy gradient approaches lies in the updating of the target network in the DQN implementation. By default, the target network in the DQN implementation is updated by copying the weights of the behavior network. This differs significantly from the updates of the value networks in the TD3 implementation where the target network weights are updated by adding the behavior network weights multiplied by 0.005. The target networks in the DQN implementation are therefore completely changed every 2,000 time step, whereas the TD3 implementation gradually updates the target networks. Sadly this difference was noticed too late to train the agents again for experiments 1a and 2a.

To draw a conclusion from the results, and answer the first research question, **RQ1**, the results from experiment 1a indicate that using AGRI with a value-based approach does not improve learning.

5.2 Experiment 1b

When evaluating the agents trained using AGRI with a policy gradient RL algorithm it is clear that AGRI outperforms the vanilla RL implementation by a great margin. The video linked to in Section 4.6.4 shows that the agent is able to follow lanes reasonably well, slow down to not run into leading vehicles, and change lanes. However, as can be seen in the video the agent does not at all consider red traffic lights when driving, and it does oscillate a lot around the lane center. The agent trained in experiment 1b is still the best out of all the agents trained, and it does learn to follow lanes to a satisfactory degree considering the low number of time steps used for the training. With the 500,000 time steps used for training at 20 frames per second, the agent has experienced just under 7 hours of simulated driving.

To draw a conclusion from the results, and answer the second research question, **RQ2**, the results from experiment 1b indicate that using AGRI with a policy gradient approach does indeed improve learning.

5.3 Experiment 2a and 2b

The results from evaluating the trained agent from experiment 2a and 2b agrees well with the training results from Section 4.4 and Section 4.5. The DQN implementation in this experiment suffers from the same drawbacks of the target update explained in Section 5.1 that could be the explanation for why the DQN implementations failed to learn. For the TD3 the AGRI agent trained without the kinematic bicycle model performs significantly better. In both Figure 4.6 and Table 4.1, the agent without the bicycle model achieves a consistently higher reward as well as a higher maximum reward. The results give no indication that the bicycle model improves learning in any way, but rather that it adds noise to the state representation and makes learning more difficult.

To draw a conclusion from the results, and answer the third research question,

RQ3, the results from experiments 2a and 2b indicate that using a kinematic bicycle model to augment the state representation does not improve learning at all.

5.4 Shortcomings

5.4.1 Flaws in the Reward Function

As mentioned in Section 4.2 and Section 4.6 the reward function used to train the agents has design flaws that should be corrected if used in further studies. The reward should be designed in a way so that the agent does not receive negative rewards when waiting behind a leading vehicle or waiting for a red light. An approach similar to that of Chen et al. in [33] where they reward the agent 0.01 times the speed reward in areas where the desired speed is zero. They also ignore orientation and distance to lane center in these areas. In this thesis, the speed reward for such areas has been set to zero, but the lateral component and angular component are still considered. Using zero as the speed reward component in all areas where the desired speed is zero leads to scenarios where the maximum episode reward could be zero, for example, if the agent spawns in front of a red light. With the current reward design, this type of scenario gives the exact same reward as spawning anywhere and not moving at all. This is not desirable as the agents then have no way to differentiate between when to stand still and when to not. Even worse could be if the agent spawns a few meters in front of the red light. By not ignoring the lateral and angular reward components when the desired speed is zero the agent could be rewarded -60 for stopping 0.5 meters away from the lane center and staying there until the episode times out. These flaws in the design function could be one of the reasons why the agents fail to converge. Especially the continuous AGRI agent that learns to somewhat consistently achieve high rewards could suffer from this design.

5.4.2 Gradient Steps and Replay Buffer Size

Another factor that could be crucial in the learning process for the agents is the replay buffer size. The replay buffer size used in this thesis is incredibly small for such a complex task. In combination with the small buffer size, a high number of gradient steps have been used. It is highly likely that this could have caused the models to overfit to recent observations. If the agent manages to complete ten RL episodes in a row where the episode length is the maximum then the replay buffer will be completely switched out. Furthermore, agents are trained for one gradient step for every step taken in the environment. This means that for every episode with a length of 1,000 steps the weights of the network will be updated 1,000 times. Recall from Section 3.3.1 that the mini-batch size used in 16. This means that a total of 16,000 samples will be sampled from the replay buffer during an update of the weights and that there are bound to be duplicate samples in the training data. This could possibly be the source of over-fitting and could be an explanation for why the continuous AGRI agent tended to have an oscillating reward and actor loss.

A solution to the over-fitting problem could be to reduce the number of gradient steps per update. A challenge is that a fixed number of gradient steps would then have to be performed at each weight update. This could potentially lead to over-fitting if the agent completes a series of very short episodes. Similarly, the number of gradient steps could be too small, so there would exist samples in the replay buffer that was never sampled before being replaced. This would be bad for the sample efficiency.

A better solution would be to increase the buffer size. The problem with the current implementation is that the RGB images are stored directly in the replay buffer. Toromanoff et al. present a solution in [28] where they only store the encoded vector representation of the state. This method could be applicable to this thesis as well, by storing all images as their encoded vectors of length 128. This would reduce the size of each image by a factor of approximately 500 which in turn should enable the replay buffer size to be increased by at least two orders of magnitude.

To implement this into the current solution all the RGB images would have to be encoded and stored in a new dataset of expert demonstrations. The visual encoder would also have to be moved from a part of the policy network to a part of the environment. The states emitted by the environment would then be in the form of the encoded vector state representation depicted in Figure 3.4. In theory, moving this part of the policy network to the environment should not make any difference, as it does not have any weights that should be updated. By doing this the learning process should also experience an increase in speed in terms of frames per second. In the current solution, samples from RL rollouts have to pass through the visual encoder in order to select an action, and again during the gradient steps. By moving the encoder to the environment, the samples would only have to be encoded a single time, when emitting the state.

Despite the design flaws in the reward function, the small replay buffer size, and the relatively large amount of gradient steps when updating the policy, AGRI still shows some evidence that it is able to learn a policy that can drive a vehicle in an urban setting. The video linked to in Section 4.6 shows how the AGRI agent from experiment 1b is able to follow the lane, change lanes, and follow a leading vehicle for a while. These results are from training for only 500,000 time steps. The fact that the vanilla RL agent only learned to stand still while the AGRI agents learn to follow vehicles could indicate that AGRI might be a more robust method for training RL agents.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presents AGRI, an adaptive extension to GRI presented by Chekroun et al. [4]. AGRI utilizes the replay buffer of off-policy RL algorithms to mix expert demonstrations into the RL learning process. Experimental results indicate that AGRI works to some extent, and could possibly be a robust method for training RL agents.

Research question 1 seeks to find out if AGRI can be used together with a value-based off-policy RL algorithm. DQN was chosen as the RL algorithm for the experiments. The experiments fail to provide any results that indicate that an agent trained using AGRI is able to learn better than a standard DQN RL approach.

Research question 2 seeks to find out if AGRI can be used together with a policy gradient RL algorithm with a continuous action space. TD3 was chosen as the RL algorithm for the experiments. The results from the experiments indicate that the agent trained with AGRI performs significantly better than the baseline agent trained with TD3 in a standard RL approach. The agent learns to follow lanes and leading vehicles to some degree, with a relatively low number of interactions with the environment.

Research question 3 seeks to find out if the representation of the environment state can be augmented using a kinematic bicycle model and whether this can improve the performance of AGRI. The results from the experiments indicate this is not the case. By adding the augmentations from the bicycle model, the performance of the agents is lower than without the augmentations.

6.2 Future Work

In the current implementations GRI [4] and AGRI are only compatible with off-policy RL algorithms, as they rely on the replay buffer of the algorithms. In future work, it would be interesting to see an extension to GRI and AGRI that is able to use on-policy algorithms as well. This should in theory be possible to implement. The forward pass of off-policy algorithms returns an action and the expected value of the action. On-policy algorithms, on the other hand, return the action, value, and the log-probability of selecting the action given the current state under the current policy. In order to use GRI and AGRI together with an on-policy, all that has to be done is to calculate the log probability of selecting the expert demonstration under the current policy.

One drawback of using on-policy RL approaches is that the sample efficiency is low as they do not use replay buffers. Instead, they only store the states between updates in a rollout buffer. A second drawback is that the expert demonstration observations also have to be passed through the policy network to calculate the policy distribution of the state. This will most likely slow down the frame rate of the learning process, although this can be mitigated by encoding the state representations of the expert demonstrations in advance as discussed in Chapter 5.

Bibliography

[1] N. C. for Statistics and Analysis, *Early estimate of motor vehicle traffic fatalities for the first half (january–june) of 2021*, 2021.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, 'Mastering the game of go with deep neural networks and tree search,' *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, ISSN: 1476-4687. DOI: `10.1038/nature16961`. [Online]. Available: `https://doi.org/10.1038/nature16961`.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski and S. Zhang, 'Dota 2 with large scale deep reinforcement learning,' *CoRR*, vol. abs/1912.06680, 2019. arXiv: `1912.06680`. [Online]. Available: `http://arxiv.org/abs/1912.06680`.

[4] R. Chekroun, M. Toromanoff, S. Hornauer and F. Moutarde, 'GRI: general reinforced imitation and its application to vision-based autonomous driving,' *CoRR*, vol. abs/2111.08575, 2021. arXiv: `2111.08575`. [Online]. Available: `https://arxiv.org/abs/2111.08575`.

[5] R. Bellman, R. Corporation and K. M. R. Collection, *Dynamic Programming*, ser. Rand Corporation research study. Princeton University Press, 1957, ISBN: 9780691079516. [Online]. Available: `https://books.google.no/books?id=wdtoPwAACAAJ`.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, 'Asynchronous methods for deep reinforcement learning,' *CoRR*, vol. abs/1602.01783, 2016. arXiv: `1602.01783`. [Online]. Available: `http://arxiv.org/abs/1602.01783`.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, 'Proximal policy optimization algorithms,' *CoRR*, vol. abs/1707.06347, 2017. arXiv: `1707.06347`. [Online]. Available: `http://arxiv.org/abs/1707.06347`.

[9] J. Schulman, S. Levine, P. Moritz, M. I. Jordan and P. Abbeel, 'Trust region policy optimization,' *CoRR*, vol. abs/1502.05477, 2015. arXiv: `1502.05477`. [Online]. Available: `http://arxiv.org/abs/1502.05477`.

[10] D. Chen, B. Zhou, V. Koltun and P. Krähenbühl, *Learning by cheating*, 2019. DOI: `10.48550/ARXIV.1912.12294`. [Online]. Available: `https://arxiv.org/abs/1912.12294`.

[11] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, 'Deterministic policy gradient algorithms,' in *International conference on machine learning*, PMLR, 2014, pp. 387–395.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, 'Human-level control through deep reinforcement learning,' *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 1476-4687. DOI: `10.1038/nature14236`. [Online]. Available: `https://doi.org/10.1038/nature14236`.

[13] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus and N. Dormann, 'Stable-baselines3: Reliable reinforcement learning implementations,' *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: `http://jmlr.org/papers/v22/20-1364.html`.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, *Continuous control with deep reinforcement learning*, 2015. DOI: `10.48550/ARXIV.1509.02971`. [Online]. Available: `https://arxiv.org/abs/1509.02971`.

[15] S. Fujimoto, H. van Hoof and D. Meger, 'Addressing function approximation error in actor-critic methods,' *CoRR*, vol. abs/1802.09477, 2018. arXiv: `1802.09477`. [Online]. Available: `http://arxiv.org/abs/1802.09477`.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, 'Openai gym,' *CoRR*, vol. abs/1606.01540, 2016. arXiv: `1606.01540`. [Online]. Available: `http://arxiv.org/abs/1606.01540`.

[17] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup and D. Meger, 'Deep reinforcement learning that matters,' *CoRR*, vol. abs/1709.06560, 2017. arXiv: `1709.06560`. [Online]. Available: `http://arxiv.org/abs/1709.06560`.

[18] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,' *CoRR*, vol. abs/1801.01290, 2018. arXiv: `1801.01290`. [Online]. Available: `http://arxiv.org/abs/1801.01290`.

[19] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel and W. Zaremba, 'Hindsight experience replay,' *CoRR*, vol. abs/1707.01495, 2017. arXiv: 1707.01495. [Online]. Available: http://arxiv.org/abs/1707.01495.

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López and V. Koltun, 'CARLA: an open urban driving simulator,' *CoRR*, vol. abs/1711.03938, 2017. arXiv: 1711.03938. [Online]. Available: http://arxiv.org/abs/1711.03938.

[21] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié and C. Guionneau, 'Torcs, the open racing car simulator,' 2015.

[22] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Mozeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky and S. Kim, 'LGSVL simulator: A high fidelity simulator for autonomous driving,' *CoRR*, vol. abs/2005.03778, 2020. arXiv: 2005.03778. [Online]. Available: https://arxiv.org/abs/2005.03778.

[23] S. Ross, G. J. Gordon and J. A. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning*, 2010. DOI: 10.48550/ARXIV.1011.0686. [Online]. Available: https://arxiv.org/abs/1011.0686.

[24] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[25] K. He, X. Zhang, S. Ren and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[26] F. Codevilla, M. Müller, A. López, V. Koltun and A. Dosovitskiy, *End-to-end driving via conditional imitation learning*, 2017. DOI: 10.48550/ARXIV.1710.02410. [Online]. Available: https://arxiv.org/abs/1710.02410.

[27] F. Codevilla, E. Santana, A. M. López and A. Gaidon, *Exploring the limitations of behavior cloning for autonomous driving*, 2019. DOI: 10.48550/ARXIV.1904.08980. [Online]. Available: https://arxiv.org/abs/1904.08980.

[28] M. Toromanoff, É. Wirbel and F. Moutarde, 'End-to-end model-free reinforcement learning for urban driving using implicit affordances,' *CoRR*, vol. abs/1911.10868, 2019. arXiv: 1911.10868. [Online]. Available: http://arxiv.org/abs/1911.10868.

[29] F. Codevilla, M. Müller, A. Dosovitskiy, A. M. López and V. Koltun, 'End-to-end driving via conditional imitation learning,' *CoRR*, vol. abs/1710.02410, 2017. arXiv: 1710.02410. [Online]. Available: http://arxiv.org/abs/1710.02410.

[30] M. Toromanoff, É. Wirbel and F. Moutarde, 'Is deep reinforcement learning really superhuman on atari?' *CoRR*, vol. abs/1908.04683, 2019. arXiv: `1908.04683`. [Online]. Available: `http://arxiv.org/abs/1908.04683`.

[31] A. Sauer, N. Savinov and A. Geiger, 'Conditional affordance learning for driving in urban environments,' *CoRR*, vol. abs/1806.06498, 2018. arXiv: `1806.06498`. [Online]. Available: `http://arxiv.org/abs/1806.06498`.

[32] F. Codevilla, E. Santana, A. M. López and A. Gaidon, 'Exploring the limitations of behavior cloning for autonomous driving,' *CoRR*, vol. abs/1904.08980, 2019. arXiv: `1904.08980`. [Online]. Available: `http://arxiv.org/abs/1904.08980`.

[33] D. Chen, V. Koltun and P. Krähenbühl, 'Learning to drive from a world on rails,' *CoRR*, vol. abs/2105.00636, 2021. arXiv: `2105.00636`. [Online]. Available: `https://arxiv.org/abs/2105.00636`.

[34] P. Polack, F. Altché, B. d'Andréa-Novel and A. de La Fortelle, 'The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?' *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 812–818, 2017.

[35] M. Tan and Q. V. Le, 'Efficientnet: Rethinking model scaling for convolutional neural networks,' *CoRR*, vol. abs/1905.11946, 2019. arXiv: `1905.11946`. [Online]. Available: `http://arxiv.org/abs/1905.11946`.

[36] D. Chen and P. Krähenbühl, 'Learning from all vehicles,' in *CVPR*, 2022.

[37] H. Wu, W. Su and Z. Liu, 'Pid controllers: Design and tuning methods,' in *2014 9th IEEE Conference on Industrial Electronics and Applications*, 2014, pp. 808–813. DOI: `10.1109/ICIEA.2014.6931273`.

[38] B. Jaeger. 'Expert drivers for autonomous driving.' (2021), [Online]. Available: `https://kait0.github.io/files/master_thesis_bernhard_jaeger.pdf`.

[39] E. Romera, J. M. Álvarez, L. M. Bergasa and R. Arroyo, 'Erfnet: Efficient residual factorized convnet for real-time semantic segmentation,' *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018. DOI: `10.1109/TITS.2017.2750080`.

[40] G. J. van Wyk and A. S. Bosman, 'Evolutionary neural architecture search for image restoration,' *CoRR*, vol. abs/1812.05866, 2018. arXiv: `1812.05866`. [Online]. Available: `http://arxiv.org/abs/1812.05866`.

[41] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: `10.48550/ARXIV.1412.6980`. [Online]. Available: `https://arxiv.org/abs/1412.6980`.

Appendix A

Additional Material

A.1 Route Configuration File

Code listing A.1: The route configuration file used for training

```xml
<?xml version="1.0" encoding="UTF-8"?>
<routes>
   <route id="0" town="Town01">
      <weather
         cloudiness="0"
         precipitation="0"
         precipitation_deposits="0"
         wind_intensity="0"
         sun_azimuth_angle="0"
         sun_altitude_angle="70"
         fog_density="0"
         fog_distance="0"
         wetness="0"
      />
      <waypoint pitch="360.0" roll="0.0" x="338.7027893066406" y="
          226.75003051757812" yaw="269.9790954589844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="321.98931884765625" y="
          194.67242431640625" yaw="179.83230590820312" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="283.6903991699219" y="
          194.78451538085938" yaw="179.83230590820312" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="108.0505142211914" y="
          195.29856872558594" yaw="179.83230590820312" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.40200805664062" y="210.57827758789062"
           yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.41706848144531" y="309.6344299316406"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="75.58748626708984" y="326.3004455566406
          " yaw="180.0352020263672" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="14.334035873413086" y="
          326.2628173828125" yaw="180.0352020263672" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="1.8717632293701172" y="
          299.4347229003906" yaw="269.8846435546875" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="1.612621784210205" y="
          170.71238708496094" yaw="269.8846435546875" z="0.0" />
```

```
    <waypoint pitch="360.0" roll="0.0" x="1.3654530048370361" y="
        47.93744659423828" yaw="269.8846435546875" z="0.0" />
  </route>
  <route id="1" town="Town01">
    <waypoint pitch="0.0" roll="0.0" x="121.72344970703125" y="59.17844009399414"
         yaw="0.0670308843255043" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="249.13783264160156" y="59.3275032043457"
        yaw="0.0670308843255043" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="315.4411315917969" y="59.40507507324219"
        yaw="0.0670308843255043" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="334.6475524902344" y="75.26239013671875"
        yaw="89.9791030883789" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="334.6621398925781" y="115.21650695800781"
         yaw="89.9791030883789" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="312.11029052734375" y="
        129.5294189453125" yaw="180.18728637695312" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="254.0718994140625" y="
        129.33969116210938" yaw="180.18728637695312" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="130.1896514892578" y="128.9347381591797
        " yaw="180.18728637695312" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="104.88580322265625" y="
        128.85202026367188" yaw="180.18728637695312" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="92.3873062133789" y="113.84988403320312
        " yaw="269.99127197265625" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="92.3824462890625" y="81.85515594482422"
         yaw="269.99127197265625" z="0.0" />
  </route>
  <route id="2" town="Town01">
    <waypoint pitch="0.0" roll="0.0" x="53.05508041381836" y="-2.35788631439209"
        yaw="-179.970947265625" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-2.673415422439575" y="28.63454818725586"
         yaw="89.88465118408203" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-2.3427786827087402" y="192.8701171875"
        yaw="89.88465118408203" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-2.1942241191864014" y="266.6610107421875
        " yaw="89.88465118408203" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-2.1050660610198975" y="310.9482727050781
        " yaw="89.88465118408203" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="24.25780487060547" y="330.2689208984375"
        yaw="0.03519752621650696" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="67.7093276977539" y="330.29559326171875"
        yaw="0.03519752621650696" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="103.49156951904297" y="330.31756591796875
        " yaw="0.035137325525283813" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="271.3299560546875" y="330.4205017089844"
        yaw="0.035137325525283813" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="316.4518127441406" y="330.4482421875" yaw
        ="0.03531792759895325" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="376.9873352050781" y="330.485595703125"
        yaw="0.03531792759895325" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="396.4393310546875" y="290.8396301269531"
        yaw="-90.04281616210938" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="396.38623046875" y="219.68142700195312"
        yaw="-90.04266357421875" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="396.3192443847656" y="129.7147979736328"
        yaw="-90.04266357421875" z="0.0" />
  </route>
  <route id="3" town="Town01">
    <waypoint pitch="360.0" roll="0.0" x="315.2674560546875" y="
        1.7750924825668335" yaw="0.029052734375" z="0.0" />
```

```
      <waypoint pitch="0.0" roll="0.0" x="334.6253356933594" y="14.335249900817871"
           yaw="89.9791030883789" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="315.8530578613281" y="55.40555191040039
          " yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="226.33297729492188" y="
          55.300819396972656" yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="173.5809326171875" y="
          55.239105224609375" yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="157.98867797851562" y="
          39.896236419677734" yaw="269.930908203125" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="141.9262237548828" y="-2.3128161430358887
          " yaw="-179.970947265625" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="109.39694213867188" y="-2.329313039779663
          " yaw="-179.970947265625" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.37610626220703" y="40.17418670654297"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.38069152832031" y="70.33818054199219"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.38651275634766" y="108.64129638671875"
           yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.391845703125" y="143.68722534179688"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.39682006835938" y="176.42515563964844"
           yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.4019775390625" y="210.35337829589844"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.41710662841797" y="309.8704833984375"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="75.55514526367188" y="326.3004150390625
          " yaw="180.0352020263672" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="12.632027626037598" y="
          326.26177978515625" yaw="180.0352020263672" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="1.8649307489395142" y="
          296.0408020019531" yaw="269.8846435546875" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="1.4228647947311401" y="76.4553451538086
          " yaw="269.8846435546875" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="1.3183270692825317" y="
          24.528728485107422" yaw="269.8846435546875" z="0.0" />
   </route>
   <route id="4" town="Town01">
      <waypoint pitch="360.0" roll="0.0" x="56.90949630737305" y="1.644068956375122
          " yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="108.8623046875" y="1.6704164743423462"
          yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="134.96644592285156" y="
          1.6836549043655396" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="153.95574951171875" y="12.585290908813477
          " yaw="89.93091583251953" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="153.98809814453125" y="39.41703414916992"
           yaw="89.93091583251953" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="139.60507202148438" y="
          55.19935607910156" yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="108.625244140625" y="55.16311264038086"
           yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.3819351196289" y="78.5218505859375"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.38763427734375" y="116.0037612915039"
          yaw="89.99128723144531" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="88.3920669555664" y="145.12017822265625"
          yaw="89.99128723144531" z="0.0" />
```

```
     <waypoint pitch="0.0" roll="0.0" x="88.39686584472656" y="176.69561767578125"
          yaw="89.99128723144531" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="182.24264526367188" y="199.08143615722656
         " yaw="-0.16769057512283325" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="310.5677185058594" y="198.7058563232422"
         yaw="-0.16769057512283325" z="0.0" />
  </route>
  <route id="5" town="Town01">
     <waypoint pitch="0.0" roll="0.0" x="334.7254638671875" y="288.90679931640625"
          yaw="89.9791030883789" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="334.7344970703125" y="313.657958984375"
         yaw="89.9791030883789" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="357.1101379394531" y="330.47332763671875"
          yaw="0.03531792759895325" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="396.45916748046875" y="317.3710021972656"
          yaw="-90.04281616210938" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="396.43365478515625" y="283.24346923828125
         " yaw="-90.04281616210938" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="396.318115234375" y="128.203857421875"
         yaw="-90.04266357421875" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="396.2600402832031" y="50.238136291503906"
          yaw="-90.04265594482422" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="396.2330017089844" y="13.925765991210938"
          yaw="-90.04265594482422" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="368.76617431640625" y="
         -2.1977765560150146" yaw="-179.970947265625" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="308.4601135253906" y="-2.228360176086426"
          yaw="-179.970947265625" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="207.614501953125" y="-2.279503107070923"
         yaw="-179.970947265625" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="173.9556427001953" y="-2.296572685241699"
          yaw="-179.970947265625" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="153.95753479003906" y="14.065252304077148
         " yaw="89.93091583251953" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="153.9872283935547" y="38.697792053222656"
          yaw="89.93091583251953" z="0.0" />
     <waypoint pitch="360.0" roll="0.0" x="109.57969665527344" y="
         55.16423034667969" yaw="180.06703186035156" z="0.0" />
     <waypoint pitch="360.0" roll="0.0" x="92.3765869140625" y="43.30881881713867"
          yaw="269.99127197265625" z="0.0" />
     <waypoint pitch="360.0" roll="0.0" x="92.37274169921875" y="18.01288414001465
         " yaw="269.99127197265625" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="68.05789947509766" y="-2.3502776622772217
         " yaw="-179.970947265625" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="14.081366539001465" y="
         -2.3776514530181885" yaw="-179.970947265625" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="-2.6820547580718994" y="
         24.343191146850586" yaw="89.88465118408203" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="-2.4851901531219482" y="
         122.13080596923828" yaw="89.88465118408203" z="0.0" />
     <waypoint pitch="0.0" roll="0.0" x="-2.1895012855529785" y="269.0069580078125
         " yaw="89.88465118408203" z="0.0" />
  </route>
  <route id="6" town="Town01">
     <waypoint pitch="360.0" roll="0.0" x="92.41790771484375" y="315.1663513183594
         " yaw="269.99127197265625" z="0.0" />
     <waypoint pitch="360.0" roll="0.0" x="92.41033172607422" y="
         265.31756591796875" yaw="269.99127197265625" z="0.0" />
     <waypoint pitch="360.0" roll="0.0" x="92.40457153320312" y="
         227.42677307128906" yaw="269.99127197265625" z="0.0" />
```

```
      <waypoint pitch="360.0" roll="0.0" x="92.40277099609375" y="
          215.60093688964844" yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="219.9288330078125" y="198.97113037109375"
           yaw="-0.16769057512283325" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="313.20977783203125" y="198.6981201171875"
           yaw="-0.16769057512283325" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="338.6866455078125" y="
          182.45599365234375" yaw="269.9790954589844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="338.6753234863281" y="
          151.41770935058594" yaw="269.9790954589844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="320.4845275878906" y="
          129.55679321289062" yaw="180.18728637695312" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="199.1981658935547" y="
          129.16030883789062" yaw="180.18728637695312" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="129.8414764404297" y="128.93359375" yaw
          ="180.18728637695312" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="107.4002685546875" y="
          128.86024475097656" yaw="180.18728637695312" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="92.38561248779297" y="
          102.70710754394531" yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="92.381103515625" y="73.05748748779297"
          yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="114.37248992919922" y="59.16984176635742"
           yaw="0.0670308843255043" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="135.82371520996094" y="59.19493865966797"
           yaw="0.0670308843255043" z="0.0" />
   </route>
   <route id="7" town="Town01">
      <waypoint pitch="360.0" roll="0.0" x="157.9880828857422" y="39.39617156982422
          " yaw="269.930908203125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="157.96221923828125" y="
          17.952861785888672" yaw="269.930908203125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="253.0749053955078" y="
          1.7435522079467773" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="315.1820983886719" y="
          1.7750492095947266" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="348.85223388671875" y="
          1.7921247482299805" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="379.3345031738281" y="
          1.8075834512710571" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.24493408203125" y="
          29.937896728515625" yaw="89.95734405517578" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.2692565917969" y="
          62.618507385253906" yaw="89.95734405517578" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.34820556640625" y="
          168.6100311279297" yaw="89.95733642578125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.4200439453125" y="265.0082702636719
          " yaw="89.95718383789062" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.4576721191406" y="315.3871765136719
          " yaw="89.95718383789062" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="361.8758850097656" y="326.4762878417969
          " yaw="180.0353240966797" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="299.8201599121094" y="326.4380187988281
          " yaw="180.0353240966797" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="269.32763671875" y="326.4192810058594"
          yaw="180.03514099121094" z="0.0" />
   </route>
   <route id="8" town="Town01">
      <waypoint pitch="360.0" roll="0.0" x="204.92605590820312" y="
          326.3797912597656" yaw="180.03514099121094" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="111.30892944335938" y="
          326.3223571777344" yaw="180.03514099121094" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="92.4170913696289" y="309.8084716796875"
           yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="92.41275787353516" y="
          281.28570556640625" yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="92.40253448486328" y="214.0068359375"
          yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="92.39788818359375" y="
          183.42884826660156" yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="92.39239501953125" y="
          147.30715942382812" yaw="269.99127197265625" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="105.45079803466797" y="132.8538818359375"
           yaw="0.1872929185628891" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="214.47857666015625" y="133.2102813720703"
           yaw="0.1872929185628891" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="312.8659362792969" y="133.53189086914062"
           yaw="0.1872929185628891" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="334.67816162109375" y="159.21560668945312
          " yaw="89.9791030883789" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="334.68341064453125" y="173.56646728515625
          " yaw="89.9791030883789" z="0.0" />
   </route>
   <route id="9" town="Town01">
      <waypoint pitch="360.0" roll="0.0" x="338.6842956542969" y="
          176.00216674804688" yaw="269.9790954589844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="338.6741638183594" y="
          148.27407836914062" yaw="269.9790954589844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="338.6495666503906" y="80.78645324707031
          " yaw="269.9790954589844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="320.7290344238281" y="55.4112548828125"
           yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="212.8487548828125" y="55.2850456237793"
           yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="173.32044982910156" y="
          55.238800048828125" yaw="180.06703186035156" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="157.9892120361328" y="
          40.340248107910156" yaw="269.930908203125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="157.95950317382812" y="
          15.695076942443848" yaw="269.930908203125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="173.07179260253906" y="
          1.702979564666748" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="280.5867614746094" y="
          1.7575045824050903" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="312.7760314941406" y="
          1.7738289833068848" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="349.441162109375" y="1.7924233675003052
          " yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="380.9141540527344" y="
          1.8083845376968384" yaw="0.029052734375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.23345947265625" y="
          14.538103103637695" yaw="89.95734405517578" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.3465576171875" y="
          166.41212463378906" yaw="89.95733642578125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.4117431640625" y="
          253.89410400390625" yaw="89.95718383789062" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="392.44500732421875" y="
          298.43646240234375" yaw="89.95718383789062" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="378.6864318847656" y="
          326.48663330078125" yaw="180.0353240966797" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="365.12506103515625" y="326.478271484375
          " yaw="180.0353240966797" z="0.0" />
   </route>
</routes>
```

A.2 Route Configuration File for Evaluation

Code listing A.2: The route configuration file used for evaluation

```
<?xml version="1.0" encoding="UTF-8"?>
<routes>
   <route id="30" town="Town04">
      <weather
         cloudiness="0"
         precipitation="0"
         precipitation_deposits="0"
         wind_intensity="0"
         sun_azimuth_angle="0"
         sun_altitude_angle="70"
         fog_density="0"
         fog_distance="0"
         wetness="0"
      />
      <waypoint pitch="0.0" roll="0.0" x="-488.22320556640625" y="339.6849670410156
          " yaw="59.5145149230957" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-475.5334167480469" y="358.6210021972656"
           yaw="52.83040237426758" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-447.5931701660156" y="388.4106140136719"
           yaw="40.839210510253906" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-419.9382019042969" y="408.3812255859375"
           yaw="30.82954216003418" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-390.7762145996094" y="422.646728515625"
          yaw="21.30451011657715" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-353.0600891113281" y="433.14752197265625
          " yaw="9.811640739440918" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-321.3542785644531" y="436.00048828125"
          yaw="0.4719550609588623" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-277.26324462890625" y="
          435.85406494140625" yaw="-0.2081967145204544" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-226.70909118652344" y="435.6703796386719
          " yaw="-0.2081967145204544" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-177.30760192871094" y="
          434.55560302734375" yaw="-5.559072494506836" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-146.08168029785156" y="429.1661376953125
          " yaw="-14.025965690612793" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-112.87530517578125" y="417.9194030761719
          " yaw="-23.39567756652832" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-84.46420288085938" y="403.0145568847656"
           yaw="-31.968528747558594" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-58.0605583190918" y="383.5847473144531"
          yaw="-40.728450775146484" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-29.442157745361328" y="
          353.67669677734375" yaw="-51.796146392822266" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="54.57451629638672" y="269.7650451660156"
          yaw="-40.916812896728516" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="82.24029541015625" y="248.6325225830078"
          yaw="-33.83201217651367" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="133.7696990966797" y="221.35757446289062"
           yaw="-21.953279495239258" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="219.6721954345703" y="192.3584747314453"
          yaw="-20.596675872802734" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="321.0372009277344" y="146.99412536621094"
           yaw="-30.665882110595703" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="382.2953796386719" y="92.96318054199219"
          yaw="-56.819374084472656" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="413.94622802734375" y="
          -240.84771728515625" yaw="267.1632080078125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="409.15411376953125" y="
          -272.8075866699219" yaw="255.7819061279297" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="394.5356750488281" y="
          -310.2859802246094" yaw="241.60153198242188" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="361.81060791015625" y="
          -352.4405212402344" yaw="222.7532958984375" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="322.1484680175781" y="
          -379.4797668457031" yaw="205.8142547607422" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="271.0467834472656" y="
          -394.52667236328125" yaw="186.9999237060547" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="145.00303649902344" y="
          -395.58477783203125" yaw="175.4221954345703" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="84.82112121582031" y="
          -379.7251892089844" yaw="155.05113220214844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="53.213504791259766" y="
          -360.7630920410156" yaw="143.0278778076172" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="15.176793098449707" y="
          -315.29376220703125" yaw="123.47056579589844" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="-13.497654151916504" y="
          -219.7641357421875" yaw="90.8860778808594" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="-15.270705223083496" y="
          161.37664794921875" yaw="89.71086883544922" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="-15.517499923706055" y="
          235.0567626953125" yaw="94.42162322998047" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="-66.07498931884766" y="
          347.8439636230469" yaw="133.86752319335938" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="-141.9580841064453" y="
          395.2472229003906" yaw="162.14744567871094" z="0.0" />
   </route>
   <route id="31" town="Town04">
      <waypoint pitch="0.0" roll="0.0" x="-281.5526428222656" y="-87.47563934326172
          " yaw="176.10842895507812" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-310.5595397949219" y="-86.0473861694336"
           yaw="-187.61404418945312" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-331.85467529296875" y="
          -80.33483123779297" yaw="-202.41879272460938" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-354.71966552734375" y="-66.363037109375"
           yaw="-220.4357452392578" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-381.8287048339844" y="
          -23.544204711914062" yaw="-254.88758850097656" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-439.4989929199219" y="7.347505569458008"
           yaw="-190.1195068359375" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-480.9691467285156" y="26.524372100830078
          " yaw="-219.5145721435547" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-502.7483215332031" y="52.914344787597656
          " yaw="-241.42083740234375" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-513.7513427734375" y="100.86363983154297
          " yaw="90.3575210571289" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-514.5786743164062" y="201.4623565673828"
           yaw="91.7344741821289" z="0.0" />
```

```
<waypoint pitch="0.0" roll="0.0" x="-504.91351318359375" y="303.23046875" yaw
    ="71.28516387939453" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-471.06451416015625" y="
    364.29425048828125" yaw="50.71378707885742" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-427.3395690917969" y="403.7365417480469"
     yaw="33.39055252075195" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-362.9612731933594" y="431.170654296875"
    yaw="12.770912170410156" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-289.670166015625" y="435.899169921875"
    yaw="-0.2081967145204544" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-197.3670196533203" y="435.56378173828125
    " yaw="-0.2081967145204544" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-126.55042266845703" y="423.2880859375"
    yaw="-19.473011016845703" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-90.14501190185547" y="406.43914794921875
    " yaw="-30.197683334350586" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="-41.47032928466797" y="367.6764221191406"
     yaw="-46.86736297607422" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="16.131141662597656" y="233.97682189941406
    " yaw="-86.51721954345703" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="15.703251838684082" y="57.05411911010742"
     yaw="-90.28913116455078" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="15.309857368469238" y="-32.79617691040039
    " yaw="-90.22486114501953" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="14.847967147827148" y="-150.4875030517578
    " yaw="-90.22486114501953" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="27.290651321411133" y="
    -170.50123596191406" yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="112.77010345458984" y="
    -170.01467895507812" yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="144.0137481689453" y="-169.85960388183594
    " yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="187.89918518066406" y="-169.5870361328125
    " yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="211.6175079345703" y="-169.45204162597656
    " yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="243.9209442138672" y="-169.26815795898438
    " yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="268.7220153808594" y="-169.1269989013672"
     yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="300.8354187011719" y="-168.9442138671875"
     yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="323.29046630859375" y="
    -168.81639099121094" yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="338.3078918457031" y="-168.7309112548828"
     yaw="0.3261248767375946" z="0.195892333984375" />
<waypoint pitch="0.0" roll="0.0" x="351.9974060058594" y="-179.7596435546875"
     yaw="-89.22856140136719" z="0.00433349609375" />
<waypoint pitch="0.0" roll="0.0" x="352.63641357421875" y="-229.4485626220703
    " yaw="-91.26260375976562" z="0.00433349609375" />
<waypoint pitch="0.0" roll="0.0" x="340.038818359375" y="-250.28768920898438"
     yaw="-178.60302734375" z="0.00433349609375" />
<waypoint pitch="0.0" roll="0.0" x="324.9431457519531" y="-250.18927001953125
    " yaw="179.60549926757812" z="0.00433349609375" />
<waypoint pitch="0.0" roll="0.0" x="302.8276672363281" y="-250.03700256347656
    " yaw="179.60549926757812" z="0.00433349609375" />
<waypoint pitch="0.0" roll="0.0" x="268.8157043457031" y="-249.8028106689453"
     yaw="179.60549926757812" z="0.00433349609375" />
<waypoint pitch="0.0" roll="0.0" x="244.80979919433594" y="
    -249.63751220703125" yaw="179.60549926757812" z="0.00433349609375" />
```

```
      <waypoint pitch="0.0" roll="0.0" x="216.93577575683594" y="-249.4456024169922
           " yaw="179.60549926757812" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="204.1916961669922" y="
           -261.2143249511719" yaw="271.3144226074219" z="0.01959228515625" />
      <waypoint pitch="360.0" roll="0.0" x="204.96435546875" y="-294.8885192871094"
            yaw="271.3144226074219" z="0.01959228515625" />
      <waypoint pitch="0.0" roll="0.0" x="215.9167938232422" y="-307.83258056640625
           " yaw="0.5897840261459351" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="245.5916748046875" y="-307.527099609375"
           yaw="0.5897840261459351" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="268.7578125" y="-307.28863525390625" yaw=
           "0.5897840261459351" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="306.52886962890625" y="-290.6662292480469
           " yaw="65.18749237060547" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="311.8023986816406" y="-260.4535827636719"
            yaw="90.51094055175781" z="0.0043487548828125" />
   </route>
   <route id="32" town="Town04">
      <waypoint pitch="360.0" roll="0.0" x="269.0704650878906" y="
           -246.3044891357422" yaw="359.6054992675781" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="300.98529052734375" y="
           -246.5242462158203" yaw="359.6054992675781" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="315.2895812988281" y="
           -258.9996643066406" yaw="270.51092529296875" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="312.9399719238281" y="
           -283.4756164550781" yaw="253.852294921875" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="287.4882507324219" y="
           -310.2721862792969" yaw="189.25515747070312" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="266.4151306152344" y="
           -310.81292724609375" yaw="180.58978271484375" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="230.984619140625" y="
           -310.99237060546875" yaw="180.58978271484375" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="258.725967845524" y="-280.241984548755"
            yaw="90.0648851519824" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="201.725967845524" y="-280.241984548755"
            yaw="-90.0648851519824" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="230.984619140625" y="-246.0105660541485
           " yaw="0.02145987658843" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="258.725967845524" y="-280.241984548755"
            yaw="90.0648851519824" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="230.984619140625" y="
           -310.99237060546875" yaw="180.58978271484375" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="214.57583618164062" y="
           -311.3465576171875" yaw="180.58978271484375" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="191.778076171875" y="-311.1783142089844"
           yaw="-179.8567352294922" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="160.05184936523438" y="-311.2235412597656
           " yaw="-181.35772705078125" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="127.77545928955078" y="-305.0201721191406
           " yaw="-200.40090942382812" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="96.31536102294922" y="-286.0783996582031"
            yaw="-221.70245361328125" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="77.39625549316406" y="-263.79034423828125
           " yaw="-237.07952880859375" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="63.925846099853516" y="
           -234.94784545898438" yaw="-252.85215759277344" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="58.790828704833984" y="
           -202.98590087890625" yaw="-268.8935241699219" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="58.69148254394531" y="-188.51019287109375
           " yaw="90.35357666015625" z="0.033660888671875" />
      <waypoint pitch="0.0" roll="0.0" x="71.52420043945312" y="-170.24945068359375
          " yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="112.69795989990234" y="-170.0150909423828
          " yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="131.4148712158203" y="
          -185.9712371826172" yaw="272.4569396972656" z="0.0386962890625" />
      <waypoint pitch="360.0" roll="0.0" x="137.35049438476562" y="
          -211.70291137695312" yaw="-63.98918151855469" z="0.0386962890625" />
      <waypoint pitch="360.0" roll="0.0" x="151.6375732421875" y="
          -230.22340393066406" yaw="-40.845550537109375" z="0.0386962890625" />
      <waypoint pitch="360.0" roll="0.0" x="182.2176513671875" y="
          -244.37359619140625" yaw="-8.816879272460938" z="0.0386962890625" />
      <waypoint pitch="360.0" roll="0.0" x="191.0310516357422" y="
          -245.1036834716797" yaw="358.4046936035156" z="0.0386962890625" />
      <waypoint pitch="0.0" roll="0.0" x="200.0151824951172" y="-231.7708740234375"
           yaw="91.3144302368164" z="0.01959228515625" />
      <waypoint pitch="0.0" roll="0.0" x="198.86997985839844" y="-181.860595703125"
           yaw="91.3144302368164" z="0.01959228515625" />
      <waypoint pitch="0.0" roll="0.0" x="211.7692108154297" y="-169.451171875" yaw
          ="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="245.3657989501953" y="-169.2599334716797"
           yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="268.75653076171875" y="
          -169.12680053710938" yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="301.767822265625" y="-168.93890380859375"
           yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="324.3397216796875" y="-168.81040954589844
          " yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="340.23541259765625" y="
          -168.71994018554688" yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="357.42059326171875" y="-168.6221160888672
          " yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="368.80682373046875" y="
          -168.55731201171875" yaw="0.3261248767375946" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="381.82220458984375" y="
          -153.25997924804688" yaw="90.59859466552734" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="381.6919860839844" y="-140.7970428466797"
           yaw="90.59859466552734" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="384.9174499511719" y="-114.50049591064453
          " yaw="90.59859466552734" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="384.2105712890625" y="-34.1280632019043"
          yaw="90.43910217285156" z="0.0" />
      <waypoint pitch="0.3238201141357422" roll="0.0" x="361.8473815917969" y="
          5.333199977874756" yaw="149.4435272216797" z="0.03513186052441597" />
      <waypoint pitch="1.9857581853866577" roll="0.0" x="307.6686096191406" y="
          10.060099601745605" yaw="-179.02207946777344" z="1.3211320638656616" />
      <waypoint pitch="3.3505096435546875" roll="0.0" x="255.2800750732422" y="
          9.165844917297363" yaw="-179.02207946777344" z="3.7611031532287598" />
      <waypoint pitch="2.8235442638397217" roll="0.0" x="182.79576110839844" y="
          7.928563594818115" yaw="-179.02207946777344" z="7.966650009155273" />
      <waypoint pitch="0.0" roll="0.0" x="49.19037628173828" y="6.282506942749023"
          yaw="-179.76736450195312" z="11.0" />
      <waypoint pitch="-0.5447480082511902" roll="0.0" x="-1.444108247756958" y="
          6.076910018920898" yaw="-179.76736450195312" z="10.926908493041992" />
      <waypoint pitch="-0.9749510884284973" roll="0.0" x="-63.26222229003906" y="
          5.862168312072754" yaw="-179.9231719970703" z="9.915364265441895" />
      <waypoint pitch="-2.9702091217041016" roll="0.0" x="-178.98451232910156" y="
          5.70700740814209" yaw="-179.9231719970703" z="5.817442893981934" />
      <waypoint pitch="-2.0595204830169678" roll="0.0" x="-263.76251220703125" y="
          5.593338966369629" yaw="-179.9231719970703" z="1.7683030366897583" />
      <waypoint pitch="-0.6504369378089905" roll="0.0" x="-331.0777587890625" y="
          5.503083229064941" yaw="-179.9231719970703" z="0.1763741374015808" />
      <waypoint pitch="0.0" roll="0.0" x="-432.02093505859375" y="6.334071159362793
          " yaw="-185.31605529785156" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-493.97271728515625" y="39.71971893310547
          " yaw="-231.3241424560547" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-513.587158203125" y="91.11449432373047"
          yaw="-266.8977966308594" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-513.947265625" y="132.26893615722656"
          yaw="90.3575210571289" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-514.2728271484375" y="184.4401397705078"
           yaw="90.3575210571289" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-515.2552490234375" y="238.67782592773438
          " yaw="89.86629486083984" z="0.0" />
   </route>
   <route id="33" town="Town04">
      <waypoint pitch="0.0" roll="0.0" x="-63.0432243347168" y="387.7452697753906"
          yaw="-38.99549102783203" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-32.63063049316406" y="357.6307678222656"
           yaw="-50.440120697021484" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="10.684922218322754" y="270.6761169433594"
           yaw="-76.6004638671875" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="16.52663803100586" y="220.31808471679688"
           yaw="-90.16564178466797" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="16.234933853149414" y="162.4143524169922"
           yaw="-90.28913116455078" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="15.550390243530273" y="26.762535095214844
          " yaw="-90.28913116455078" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="15.254679679870605" y="
          -46.855560302734375" yaw="-90.22486114501953" z="0.0" />
      <waypoint pitch="361.47802734375" roll="0.0" x="67.48360443115234" y="
          -103.71863555908203" yaw="358.3492431640625" z="0.2963566184043884" />
      <waypoint pitch="363.54254150390625" roll="0.0" x="97.12774658203125" y="
          -100.31896209716797" yaw="20.55169677734375" z="1.7024706602096558" />
      <waypoint pitch="365.3463134765625" roll="0.0" x="118.11686706542969" y="
          -86.13691711425781" yaw="47.54095458984375" z="3.877535581588745" />
      <waypoint pitch="365.0263366699219" roll="0.0" x="131.61622619628906" y="
          -58.393516540527344" yaw="80.56587982177734" z="7.1716814041137695" />
      <waypoint pitch="363.0312805175781" roll="0.0" x="128.23634338378906" y="
          -28.46866798400879" yaw="112.11688995361328" z="9.519408226013184" />
      <waypoint pitch="-0.44407668709754944" roll="0.0" x="1.3972853422164917" y="
          6.088447570800781" yaw="-179.76736450195312" z="10.951427459716797" />
      <waypoint pitch="-0.9741529822349548" roll="0.0" x="-65.08020782470703" y="
          5.8597307205200195" yaw="-179.9231719970703" z="9.884441375732422" />
      <waypoint pitch="359.7007141113281" roll="0.0" x="-106.3203125" y="
          -9.66769027709961" yaw="226.91018676757812" z="9.322182655334473" />
      <waypoint pitch="357.0876159667969" roll="0.0" x="-119.02806854248047" y="
          -36.12523651123047" yaw="261.7791748046875" z="8.397171974182129" />
      <waypoint pitch="356.02716064453125" roll="0.0" x="-111.6511459350586" y="
          -72.12218475341797" yaw="-59.072662353515625" z="5.675800323486328" />
      <waypoint pitch="356.02716064453125" roll="0.0" x="-72.83853912353516" y="
          -98.28777313232422" yaw="-8.899246215820312" z="2.005282402038574" />
      <waypoint pitch="359.74945068359375" roll="0.0" x="-35.542240142822266" y="
          -89.2466049194336" yaw="36.93992614746094" z="0.0046467469073832035" />
      <waypoint pitch="360.0" roll="0.0" x="-16.205305099487305" y="
          -36.5977668762207" yaw="89.77513885498047" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="-16.017396926879883" y="
          11.281970977783203" yaw="89.77513885498047" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="-15.820234298706055" y="
          52.47965621948242" yaw="89.71086883544922" z="0.0" />
      <waypoint pitch="361.2682800292969" roll="0.0" x="-60.99004364013672" y="
          144.97706604003906" yaw="172.96588134765625" z="0.3752487599849701" />
      <waypoint pitch="364.0442199707031" roll="0.0" x="-118.65277862548828" y="
          117.16047668457031" yaw="240.92144775390625" z="3.8155317306518555" />
      <waypoint pitch="364.77117919921875" roll="0.0" x="-125.39179992675781" y="
          87.23468017578125" yaw="273.736083984375" z="6.545107841491699" />
      <waypoint pitch="360.97479248046875" roll="0.0" x="-63.682159423828125" y="
          37.36163330078125" yaw="0.0768280029296875" z="9.908937454223633" />
      <waypoint pitch="0.0" roll="0.0" x="15.606971740722656" y="37.97494888305664"
           yaw="-90.28913116455078" z="11.0" />
      <waypoint pitch="357.4396057128906" roll="0.0" x="170.7868194580078" y="
          39.22816467285156" yaw="0.9779205322265625" z="8.505694389343262" />
      <waypoint pitch="357.51171875" roll="0.0" x="287.8409423828125" y="
          41.22623825073242" yaw="0.9779205322265625" z="2.074390411376953" />
      <waypoint pitch="359.01605224609375" roll="0.0" x="346.86383056640625" y="
          42.00389099121094" yaw="356.32611083984375" z="0.32436490058898926" />
      <waypoint pitch="360.0" roll="0.0" x="412.2002258300781" y="
          -32.668495178222656" yaw="270.4390869140625" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="412.56622314453125" y="
          -80.42717742919922" yaw="270.4390869140625" z="0.0" />
   </route>
   <route id="34" town="Town04">
      <waypoint pitch="360.0" roll="0.0" x="92.20408630371094" y="
          -382.9563903808594" yaw="157.67535400390625" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="44.082176208496094" y="
          -353.39971923828125" yaw="139.20782470703125" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="28.456111907958984" y="
          -338.1231384277344" yaw="132.0880584716797" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-28.244382858276367" y="
          -247.3603057861328" yaw="123.9988021850586" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-97.86443328857422" y="-167.5721893310547
          " yaw="139.3140869140625" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-139.92556762695312" y="
          -132.6236572265625" yaw="148.55484008789062" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-197.8251953125" y="-102.70037841796875"
          yaw="158.62039184570312" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-234.66213989257812" y="
          -91.12406158447266" yaw="166.48809814453125" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-279.4051818847656" y="-84.12016296386719
          " yaw="175.71859741210938" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-309.8734130859375" y="-82.607666015625"
          yaw="-187.45741271972656" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-352.29095458984375" y="
          -63.83389663696289" yaw="-220.2904052734375" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-379.75689697265625" y="
          -17.014066696166992" yaw="-258.9152526855469" z="0.0" />
      <waypoint pitch="360.1035461425781" roll="0.0" x="-357.1685791015625" y="
          30.017763137817383" yaw="359.631591796875" z="0.004470161162316799" />
      <waypoint pitch="361.1860656738281" roll="0.0" x="-305.5273132324219" y="
          33.53736877441406" yaw="0.0768280029296875" z="0.5864595770835876" />
      <waypoint pitch="362.97021484375" roll="0.0" x="-204.52964782714844" y="
          33.67278289794922" yaw="0.0768280029296875" z="4.4951276779174805" />
      <waypoint pitch="361.34271240234375" roll="0.0" x="-127.03149852145" y="
          37.30177307128906" yaw="0.0768280029296875" z="9.137267112731934" />
      <waypoint pitch="360.9741516113281" roll="0.0" x="-65.13790893554688" y="
          37.35968017578125" yaw="0.0768280029296875" z="9.88417911529541" />
      <waypoint pitch="357.6973571777344" roll="0.0" x="120.90387725830078" y="
          61.45814514160156" yaw="56.92566680908203" z="9.838208198547363" />
      <waypoint pitch="355.5294494628906" roll="0.0" x="126.76893615722656" y="
          106.36044311523438" yaw="107.34747314453125" z="6.799232482910156" />
   <waypoint pitch="355.8213806152344" roll="0.0" x="97.20584106445312" y="
        140.24691772460938" yaw="154.8564453125" z="2.48572039604187" />
   <waypoint pitch="357.9312438964844" roll="0.0" x="66.20530700683594" y="
        145.62423706054688" yaw="183.5865936279297" z="0.6092644333839417" />
   <waypoint pitch="0.0" roll="0.0" x="15.716876029968262" y="59.7540283203125"
        yaw="-90.28913116455078" z="0.0" />
   <waypoint pitch="0.0" roll="0.0" x="15.451249122619629" y="3.2311558723449707
        " yaw="-90.22486114501953" z="0.0" />
   <waypoint pitch="0.0" roll="0.0" x="14.921006202697754" y="
        -131.87681579589844" yaw="-90.22486114501953" z="0.0" />
   <waypoint pitch="0.0" roll="0.0" x="14.814301490783691" y="-159.0655059814453
        " yaw="-90.22486114501953" z="0.0" />
   <waypoint pitch="0.0" roll="0.0" x="25.810285568237305" y="
        -170.50965881347656" yaw="0.3261248767375946" z="0.195892333984375" />
   <waypoint pitch="0.0" roll="0.0" x="115.74958038330078" y="
        -169.99771118164062" yaw="0.3261248767375946" z="0.195892333984375" />
   <waypoint pitch="0.0" roll="0.0" x="145.0629119873047" y="-169.86502075195312
        " yaw="0.3261248767375946" z="0.195892333984375" />
   <waypoint pitch="0.0" roll="0.0" x="187.36929321289062" y="
        -169.59005737304688" yaw="0.3261248767375946" z="0.195892333984375" />
   <waypoint pitch="0.0" roll="0.0" x="211.6328582763672" y="-169.4519500732422"
         yaw="0.3261248767375946" z="0.195892333984375" />
   <waypoint pitch="0.0" roll="0.0" x="243.6725616455078" y="-169.2695770263672"
         yaw="0.3261248767375946" z="0.195892333984375" />
   <waypoint pitch="360.0" roll="0.0" x="254.8811492919922" y="
        -154.82557678222656" yaw="90.17674255371094" z="0.01958465576171875" />
   <waypoint pitch="360.0" roll="0.0" x="254.80921936035156" y="
        -131.50787353515625" yaw="90.17674255371094" z="0.01958465576171875" />
   <waypoint pitch="0.0" roll="0.0" x="271.760498046875" y="-118.60427856445312"
         yaw="0.9240430593490601" z="0.01959228515625" />
   <waypoint pitch="0.0" roll="0.0" x="299.6679382324219" y="-118.15415954589844
        " yaw="0.9240430593490601" z="0.01959228515625" />
   <waypoint pitch="0.0" roll="0.0" x="310.4628601074219" y="-110.24256896972656
        " yaw="90.51094055175781" z="0.0043487548828125" />
   <waypoint pitch="0.0" roll="0.0" x="310.24847412109375" y="-86.20258331298828
        " yaw="90.51094055175781" z="0.0043487548828125" />
   <waypoint pitch="0.0" roll="0.0" x="330.2210693359375" y="-64.07766723632812"
         yaw="0.9061377644538879" z="0.0043487548828125" />
   <waypoint pitch="0.0" roll="0.0" x="364.35577392578125" y="-65.0423355102539"
         yaw="-0.7268505692481995" z="0.0043487548828125" />
   <waypoint pitch="0.0" roll="0.0" x="380.8219909667969" y="-48.67954635620117"
         yaw="90.43910217285156" z="0.0" />
   <waypoint pitch="0.0" roll="0.0" x="380.6858825683594" y="-30.922266006469727
        " yaw="90.43910217285156" z="0.0" />
   <waypoint pitch="0.5015116930007935" roll="0.0" x="357.5417785644531" y="
        7.521181106567383" yaw="156.6797637939453" z="0.08426664024591446" />
   <waypoint pitch="1.3669906854629517" roll="0.0" x="331.4211730957031" y="
        10.465546607971191" yaw="-179.02207946777344" z="0.6260725855827332" />
</route>
<route id="35" town="Town04">
   <waypoint pitch="3.307222366333008" roll="0.0" x="203.8810272216797" y="
        8.288481712341309" yaw="-179.02207946777344" z="6.838401794433594" />
   <waypoint pitch="2.4017601013183594" roll="0.0" x="164.40867614746094" y="
        7.61470365524292" yaw="-179.02207946777344" z="8.805212020874023" />
   <waypoint pitch="1.288453459739685" roll="0.0" x="115.87562561035156" y="
        6.786262035369873" yaw="-179.02207946777344" z="10.368358612060547" />
   <waypoint pitch="0.0" roll="0.0" x="17.27475929260254" y="6.152915954589844"
        yaw="-179.76736450195312" z="11.0" />
   <waypoint pitch="-0.9736899137496948" roll="0.0" x="-66.13497161865234" y="
        5.8583173751831055" yaw="-179.9231719970703" z="9.8665132522583" />
```

```
    <waypoint pitch="357.8592834472656" roll="0.0" x="-116.99535369873047" y="
        -27.572500228881836" yaw="251.48211669921875" z="8.826952934265137" />
    <waypoint pitch="356.02716064453125" roll="0.0" x="-118.04573822021484" y="
        -56.33713912963867" yaw="-76.82113647460938" z="6.974219799041748" />
    <waypoint pitch="356.02716064453125" roll="0.0" x="-86.76951599121094" y="
        -94.17179870605469" yaw="-24.02081298828125" z="3.111525058746338" />
    <waypoint pitch="356.1885681152344" roll="0.0" x="-60.678245544433594" y="
        -98.7884521484375" yaw="4.63446044921875" z="1.075364112854004" />
    <waypoint pitch="360.0" roll="0.0" x="-16.0787410736084" y="
        -4.348906517028809" yaw="89.77513885498047" z="0.0" />
    <waypoint pitch="360.45977783203125" roll="0.0" x="-42.43091583251953" y="
        138.98097229003906" yaw="151.22486877441406" z="0.04931524023413658" />
    <waypoint pitch="361.35125732421875" roll="0.0" x="-62.9927864074707" y="
        145.18467712402344" yaw="175.197021484375" z="0.42595288157463074" />
    <waypoint pitch="363.2038879394531" roll="0.0" x="-105.59686279296875" y="
        132.7899627685547" yaw="219.33187866210938" z="2.394624948501587" />
    <waypoint pitch="364.825439453125" roll="0.0" x="-125.36922454833984" y="
        94.58512878417969" yaw="265.9417419433594" z="5.865746021270752" />
    <waypoint pitch="361.6607971191406" roll="0.0" x="-111.92537689208984" y="
        55.14680480957031" yaw="311.796875" z="8.730497360229492" />
    <waypoint pitch="360.97528076171875" roll="0.0" x="-62.548316955566406" y="
        37.363155364990234" yaw="0.0768280029296875" z="9.92823314666748" />
    <waypoint pitch="360.0" roll="0.0" x="14.9048433303833" y="37.643550872802734
        " yaw="0.232635498046875" z="11.0" />
    <waypoint pitch="358.1361999511719" roll="0.0" x="115.05550384521484" y="
        54.051361083984375" yaw="46.485382080078125" z="10.216599464416504" />
    <waypoint pitch="356.49639892578125" roll="0.0" x="129.1477508544922" y="
        85.6916732788086" yaw="85.49930572509766" z="8.395012855529785" />
    <waypoint pitch="354.87432861328125" roll="0.0" x="113.65818786621094" y="
        128.47520446777344" yaw="133.9757843017578" z="4.349025249481201" />
    <waypoint pitch="357.5226745605469" roll="0.0" x="72.28358459472656" y="
        145.66053771972656" yaw="177.21603393554688" z="0.8736939430236816" />
    <waypoint pitch="359.6270446777344" roll="0.0" x="42.331844329833984" y="
        138.30975341796875" yaw="210.4820556640625" z="0.019801577553153038" />
    <waypoint pitch="0.0" roll="0.0" x="15.397431373596191" y="
        -10.481854438781738" yaw="-90.22486114501953" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="14.942656517028809" y="
        -126.36024475097656" yaw="-90.22486114501953" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="14.825342178344727" y="
        -156.25233459472656" yaw="-90.22486114501953" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="27.33547019958496" y="-170.5009765625"
        yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="95.40255737304688" y="-170.24444580078125
        " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="144.3073637005684" y="-222.2315674215825"
         yaw="-45.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="204.7617985159424" y="-276.6031597621564"
         yaw="-90.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="187.361479254546" y="-310.924575364244275
        " yaw="-180.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="58.93156874521454" y="
        -310.924575364244275" yaw="-180.3261248767375946" z="0.195892333984375" /
        >
    <waypoint pitch="0.0" roll="0.0" x="95.40255737304688" y="-170.24444580078125
        " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="113.82496643066406" y="-170.0086669921875
        " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="138.85244750976562" y="-169.8662109375"
        yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="187.37423706054688" y="
         -169.59002685546875" yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="210.88375854492188" y="
         -169.45620727539062" yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="243.1909942626953" y="-169.27232360839844
         " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="271.02838134765625" y="
         -169.11387634277344" yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="299.91827392578125" y="
         -168.94943237304688" yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="339.91497802734375" y="
         -168.81283569335938" yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="360.0" roll="0.0" x="328.2840576171875" y="
         -121.1314568559897" yaw="180.44322204589844" z="0.004344940185546875" />
    <waypoint pitch="360.0" roll="0.0" x="314.561482117812" y="-143.754693469346"
          yaw="-90.269681378458341" z="0.004344940185546875" />
    <waypoint pitch="360.0" roll="0.0" x="331.01798362149" y="-246.731653219784"
         yaw="0.17484318769354348" z="0.004344940185546875" />
    <waypoint pitch="360.0" roll="0.0" x="328.2840576171875" y="
         -180.2013702392578" yaw="242.44322204589844" z="0.004344940185546875" />
  </route>
  <route id="36" town="Town04">
    <waypoint pitch="0.0" roll="0.0" x="14.701272964477539" y="
         -187.86557006835938" yaw="-90.22486114501953" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="14.630420684814453" y="
         -224.15390014648438" yaw="271.5786437988281" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="31.320476531982422" y="
         -287.68646240234375" yaw="-62.1405029296875" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="65.48259735107422" y="-330.2546081542969"
          yaw="-40.3635139465332" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="102.10064697265625" y="
         -352.94903564453125" yaw="-23.2144775390625" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="140.73130798339844" y="-363.4803466796875
         " yaw="-7.283715724945068" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="176.6475372314453" y="-364.4976806640625"
          yaw="0.5013986825942993" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="190.8989715576172" y="-364.38360595703125
         " yaw="0.4299219846725464" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="202.90049743652344" y="-350.5193786621094
         " yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="202.1409149169922" y="-324.4151916503906"
          yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="191.3997802734375" y="-311.17926025390625
         " yaw="-179.8567352294922" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="164.50732421875" y="-311.24652099609375"
         yaw="-179.8567352294922" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="137.1165771484375" y="-307.9787292480469"
          yaw="-194.74749755859375" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="116.63600158691406" y="
         -300.07989501953125" yaw="-207.43331909179688" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="92.745849609375" y="-282.73663330078125"
         yaw="-224.52279663085938" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="74.62584686279297" y="-259.2897644042969"
          yaw="-239.69004821777344" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="62.53457260131836" y="-230.06149291992188
         " yaw="-255.3617401123047" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="58.682777404785156" y="
         -187.09957885742188" yaw="90.35357666015625" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="78.54766082763672" y="-170.2493133544922"
          yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="111.11359405517578" y="
        -170.02410888671875" yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="360.0" roll="0.0" x="131.49044799804688" y="
        -187.73252868652344" yaw="272.4569396972656" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="133.5595703125" y="-201.670654296875"
        yaw="-74.61013793945312" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="140.02346801757812" y="
        -216.57327270507812" yaw="-58.49299621582031" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="151.96360778808594" y="
        -230.5032958984375" yaw="-40.44239807128906" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="180.46920776367188" y="
        -244.0763397216797" yaw="-10.48089599609375" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="190.38192749023438" y="
        -245.08560180664062" yaw="358.4046936035156" z="0.0386962890625" />
    <waypoint pitch="0.0" roll="0.0" x="200.07064819335938" y="
        -234.18809509277344" yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="198.90110778808594" y="
        -183.21690368652344" yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="198.4252166748047" y="-160.47689819335938
        " yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="214.66400146484375" y="
        -125.70246124267578" yaw="36.59318161010742" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="231.2375946044922" y="-119.35476684570312
        " yaw="5.320570468902588" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="245.8983917236328" y="-119.02140808105469
        " yaw="0.9240430593490601" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="258.3119201660156" y="-132.37623596191406
        " yaw="-89.82325744628906" z="0.01958465576171875" />
    <waypoint pitch="0.0" roll="0.0" x="258.39239501953125" y="
        -158.46783447265625" yaw="-89.82325744628906" z="0.01958465576171875" />
    <waypoint pitch="0.0" roll="0.0" x="258.46453857421875" y="
        -181.85963439941406" yaw="-89.82325744628906" z="0.01958465576171875" />
    <waypoint pitch="0.0" roll="0.0" x="258.6319580078125" y="-236.132568359375"
        yaw="-89.82325744628906" z="0.01958465576171875" />
    <waypoint pitch="0.0" roll="0.0" x="258.70123291015625" y="-258.5882263183594
        " yaw="-89.82325744628906" z="0.01958465576171875" />
    <waypoint pitch="0.0" roll="0.0" x="258.8244323730469" y="-298.52032470703125
        " yaw="-89.82325744628906" z="0.01958465576171875" />
    <waypoint pitch="0.0" roll="0.0" x="267.0066223144531" y="-307.306640625" yaw
        ="0.5897840261459351" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="282.8596496582031" y="-307.1434631347656"
         yaw="0.5897840261459351" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="301.0977783203125" y="-299.2485656738281"
         yaw="46.956443786621094" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="308.9264831542969" y="-284.6018371582031"
         yaw="71.66845703125" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="311.7925720214844" y="-259.35113525390625
        " yaw="90.51094055175781" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="311.5842590332031" y="-235.9899139404297"
         yaw="90.51094055175781" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="311.1050720214844" y="-182.2567596435547"
         yaw="90.51094055175781" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="310.90240478515625" y="
        -159.53497314453125" yaw="90.51094055175781" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="310.64276123046875" y="
        -130.41705322265625" yaw="90.51094055175781" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="325.8652038574219" y="-117.73162078857422
        " yaw="0.9240430593490601" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="336.6689147949219" y="-117.63641357421875
        " yaw="-4.299356460571289" z="0.01959228515625" />
```

```
      <waypoint pitch="-0.07674998790025711" roll="0.0" x="351.3386535644531" y="
          -130.84422302246094" yaw="-88.98917388916016" z="0.007313055917620659" />
      <waypoint pitch="0.0" roll="0.0" x="351.7012634277344" y="-157.76431274414062
          " yaw="-89.22856140136719" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="352.003173828125" y="-180.1864471435547"
          yaw="-89.22856140136719" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="352.1387634277344" y="-235.35081481933594
          " yaw="-98.3767318725586" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="342.57342529296875" y="-249.9364471435547
          " yaw="-165.61737060546875" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="324.3822021484375" y="-250.18540954589844
          " yaw="179.60549926757812" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="302.6083984375" y="-250.03549194335938"
          yaw="179.60549926757812" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="268.7098388671875" y="-249.8020782470703"
           yaw="179.60549926757812" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="245.00079345703125" y="-249.6388397216797
          " yaw="179.60549926757812" z="0.00433349609375" />
      <waypoint pitch="0.0" roll="0.0" x="214.6555633544922" y="-249.42990112304688
          " yaw="179.60549926757812" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="204.12197875976562" y="
          -258.1758117675781" yaw="271.3144226074219" z="0.01959228515625" />
      <waypoint pitch="360.0" roll="0.0" x="205.01510620117188" y="
          -297.1004333496094" yaw="271.3144226074219" z="0.01959228515625" />
      <waypoint pitch="360.0" roll="0.0" x="205.64422607421875" y="
          -324.51934814453125" yaw="271.3144226074219" z="0.01959228515625" />
      <waypoint pitch="360.0" roll="0.0" x="206.39923095703125" y="
          -350.42425537109375" yaw="271.3144226074219" z="0.01959228515625" />
      <waypoint pitch="0.0" roll="0.0" x="230.66529846191406" y="-364.1372985839844
          " yaw="-0.29530856013298035" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="260.695556640625" y="-363.8970031738281"
          yaw="4.1477370262146" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="287.33489990234375" y="-359.1738586425781
          " yaw="15.96041202545166" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="313.7850036621094" y="-348.3807067871094"
           yaw="28.436067581176758" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="342.71429443359375" y="-327.1446533203125
          " yaw="44.12626647949219" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="364.6312561035156" y="-299.2043762207031"
           yaw="59.65094757080078" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="375.8125" y="-274.61480712890625" yaw="
          71.44500732421875" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="382.434814453125" y="-240.22972106933594"
           yaw="86.75253295898438" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="385.9430236816406" y="-212.6571502685547"
           yaw="90.59859466552734" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="385.6279296875" y="-182.50074768066406"
          yaw="90.59859466552734" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="385.1250305175781" y="-134.36874389648438
          " yaw="90.59859466552734" z="0.0" />
   </route>
   <route id="37" town="Town04">
      <waypoint pitch="360.0" roll="0.0" x="220.9058380126953" y="
          -395.6593017578125" yaw="180.42991638183594" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="159.40386962890625" y="
          -396.1457824707031" yaw="180.11590576171875" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="137.2083282470703" y="-394.78564453125"
           yaw="172.87075805664062" z="0.0" />
      <waypoint pitch="360.0" roll="0.0" x="104.40274810791016" y="
          -387.4465637207031" yaw="161.90866088867188" z="0.0" />
```

```
<waypoint pitch="360.0" roll="0.0" x="79.4221420288086" y="-377.1005554199219
    " yaw="153.096435546875" z="0.0" />
<waypoint pitch="360.0" roll="0.0" x="43.836021423339844" y="
    -353.1868896484375" yaw="139.1018829345703" z="0.0" />
<waypoint pitch="360.0" roll="0.0" x="24.01766014099121" y="
    -333.01483154296875" yaw="129.88455200195312" z="0.0" />
<waypoint pitch="360.0" roll="0.0" x="-16.338682556152344" y="
    -224.6736053466797" yaw="456.06732177734375" z="0.0" />
<waypoint pitch="360.0" roll="0.0" x="-16.847301483154297" y="
    -200.18057250976562" yaw="89.77513885498047" z="0.0" />
<waypoint pitch="360.0" roll="0.0" x="-16.218891143798828" y="
    -40.059322357177734" yaw="89.77513885498047" z="0.0" />
<waypoint pitch="360.0" roll="0.0" x="-15.852235794067383" y="
    46.137962341308594" yaw="89.71086883544922" z="0.0" />
<waypoint pitch="360.98297119140625" roll="0.0" x="-54.19522476196289" y="
    143.67225646972656" yaw="165.29368591308594" z="0.2254064530134201" />
<waypoint pitch="362.3365783691406" roll="0.0" x="-87.07882690429688" y="
    142.7154083251953" yaw="197.0497283935547" z="1.2736387252807617" />
<waypoint pitch="363.457763671875" roll="0.0" x="-110.15067291259766" y="
    128.603515625" yaw="225.85433959960938" z="2.7891640663146973" />
<waypoint pitch="364.16278076171875" roll="0.0" x="-119.98950958251953" y="
    114.59873962402344" yaw="243.9669647216797" z="4.0424885749816895" />
<waypoint pitch="364.1700744628906" roll="0.0" x="-124.50845336914062" y="
    80.44613647460938" yaw="281.0916748046875" z="7.1322197914123535" />
<waypoint pitch="360.9730224609375" roll="0.0" x="-67.70280456542969" y="
    37.35624313354492" yaw="0.0768280029296875" z="9.840595245361328" />
<waypoint pitch="360.5543212890625" roll="0.0" x="-1.8418172597885132" y="
    37.57555389404297" yaw="0.232635498046875" z="10.924320220947266" />
<waypoint pitch="355.69207763671875" roll="0.0" x="127.71321868896484" y="
    102.9642333984375" yaw="103.72856140136719" z="7.094720840454102" />
<waypoint pitch="354.87432861328125" roll="0.0" x="109.84391021728516" y="
    132.0736083984375" yaw="139.3603057861328" z="3.835606098175049" />
<waypoint pitch="357.2054138183594" roll="0.0" x="77.34449005126953" y="
    145.40267944335938" yaw="176.0394744873047" z="1.111791968345642" />
<waypoint pitch="359.8039245605469" roll="0.0" x="40.10015869140625" y="
    136.9214630126953" yaw="213.28746032714844" z="0.0054727462120354176" />
<waypoint pitch="0.0" roll="0.0" x="15.446653366088867" y="2.0601918697357178
    " yaw="-90.22486114501953" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="15.27511978149414" y="-41.64741516113281"
     yaw="-90.22486114501953" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="11.423349380493164" y="
    -131.27297973632812" yaw="-90.22486114501953" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="11.226259231567383" y="
    -187.49221801757812" yaw="-90.22486114501953" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="11.076330184936523" y="
    -220.64308166503906" yaw="270.18170166015625" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="38.74594497680664" y="-300.171875" yaw="
    -56.37700271606445" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="84.09489440917969" y="-343.6820983886719"
     yaw="-31.25214958190918" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="124.4841079711914" y="-360.4506530761719"
     yaw="-13.841785430908203" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="152.728759765625" y="-364.50787353515625"
     yaw="-2.5068819522857666" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="187.6028594970703" y="-364.4083251953125"
     yaw="0.4299219846725464" z="0.0" />
<waypoint pitch="0.0" roll="0.0" x="202.83053588867188" y="-352.47021484375"
    yaw="91.3144302368164" z="0.01959228515625" />
<waypoint pitch="0.0" roll="0.0" x="202.17324829101562" y="-325.8245849609375
    " yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="200.6520233154297" y="-259.5257873535156"
         yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="200.14462280273438" y="-237.4120635986328
        " yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="198.89111328125" y="-182.7812042236328"
        yaw="91.3144302368164" z="0.01959228515625" />
    <waypoint pitch="0.0" roll="0.0" x="212.9224090576172" y="-169.44461059570312
        " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="244.10043334960938" y="-169.2671356201172
        " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="269.11859130859375" y="
        -169.12474060058594" yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="300.999267578125" y="-168.9432830810547"
        yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="340.3591613769531" y="-168.71923828125"
        yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="357.4131774902344" y="-168.62216186523438
        " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="368.7809753417969" y="-168.55746459960938
        " yaw="0.3261248767375946" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="381.7611389160156" y="-147.41366577148438
        " yaw="90.59859466552734" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="381.3944396972656" y="-112.31800842285156
        " yaw="90.59859466552734" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="381.0716247558594" y="-81.25092315673828"
         yaw="-269.5501708984375" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="349.1683349609375" y="
        -68.27950286865234" yaw="178.3131103515625" z="0.0043487548828125" />
    <waypoint pitch="360.0" roll="0.0" x="332.59405517578125" y="
        -67.64382934570312" yaw="177.70645141601562" z="0.0043487548828125" />
    <waypoint pitch="360.0" roll="0.0" x="315.1619567871094" y="
        -77.60230255126953" yaw="246.20465087890625" z="0.0043487548828125" />
    <waypoint pitch="360.0" roll="0.0" x="313.9056701660156" y="
        -103.81141662597656" yaw="270.51092529296875" z="0.0043487548828125" />
    <waypoint pitch="360.0" roll="0.0" x="314.1311340332031" y="
        -133.0957489013672" yaw="270.51092529296875" z="0.0043487548828125" />
  </route>
  <route id="38" town="Town04">
    <waypoint pitch="360.0" roll="0.0" x="64.83004760742188" y="
        -224.6131134033203" yaw="-77.751708984375" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="71.2993392944336" y="-244.9078826904297
        " yaw="-66.88722229003906" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="82.44536590576172" y="
        -265.03167724609375" yaw="-55.15098571777344" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="96.07714080810547" y="
        -281.0929260253906" yaw="-43.79193115234375" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="117.77859497070312" y="
        -296.72784423828125" yaw="-27.750274658203125" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="133.2962646484375" y="
        -303.2240295410156" yaw="-17.680862426757812" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="153.1439666748047" y="
        -307.3028869628906" yaw="-5.5453338623046875" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="170.66632080078125" y="
        -307.7311096191406" yaw="0.1432647705078125" z="0.033660888671875" />
    <waypoint pitch="360.0" roll="0.0" x="190.39112854003906" y="
        -307.6817626953125" yaw="0.1432647705078125" z="0.033660888671875" />
    <waypoint pitch="0.0" roll="0.0" x="217.03329467773438" y="-307.8210754394531
        " yaw="0.5897840261459351" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="245.18458557128906" y="-307.5003967285156
        " yaw="0.5897840261459351" z="0.0043487548828125" />
```

```
      <waypoint pitch="0.0" roll="0.0" x="268.8760681152344" y="-307.28741455078125
          " yaw="0.5897840261459351" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="285.8556213378906" y="-306.9682922363281"
           yaw="6.765690326690674" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="303.4971008300781" y="-296.243896484375"
          yaw="55.82583236694336" z="0.0043487548828125" />
      <waypoint pitch="0.0" roll="0.0" x="311.79425048828125" y="-259.5401916503906
          " yaw="90.51094055175781" z="0.0043487548828125" />
      <waypoint pitch="360.0" roll="0.0" x="325.0581359863281" y="
          -246.6899871826172" yaw="359.6054992675781" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="339.20684814453125" y="
          -246.7873992919922" yaw="359.6054992675781" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="347.8764343261719" y="
          -238.92654418945312" yaw="76.22891235351562" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="349.05889892578125" y="
          -221.4833221435547" yaw="90.77143859863281" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="348.5255126953125" y="
          -181.8705596923828" yaw="90.77143859863281" z="0.00433349609375" />
      <waypoint pitch="360.0" roll="0.0" x="324.3228454589844" y="
          -172.31056213378906" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="302.01007080078125" y="
          -172.4375762939453" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="267.8661804199219" y="
          -172.63192749023438" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="242.79788208007812" y="
          -172.7746124267578" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="212.1085968017578" y="
          -172.9492950439453" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="190.2705078125" y="-173.07359313964844"
           yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="146.2179718017578" y="
          -173.35848999023438" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="116.8371810913086" y="
          -173.4915771484375" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="76.47642517089844" y="
          -173.74407958984375" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="360.0" roll="0.0" x="27.645009994506836" y="
          -173.999267578125" yaw="180.3261260986328" z="0.195892333984375" />
      <waypoint pitch="0.0" roll="0.0" x="14.69900894165039" y="-188.4425048828125"
           yaw="-90.22486114501953" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="14.615015983581543" y="-209.8443145751953
          " yaw="-90.22486114501953" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="50.487098693847656" y="-320.7240905761719
          " yaw="-47.19378662109375" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="83.29607391357422" y="-347.2858581542969"
           yaw="-30.792631149291992" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="125.97801208496094" y="-364.4034729003906
          " yaw="-12.913924217224121" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="163.1409912109375" y="-368.1159973144531"
           yaw="0.5013986825942993" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="249.90109252929688" y="-367.7364501953125
          " yaw="-0.29530856013298035" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="287.49481201171875" y="
          -359.12799072265625" yaw="16.03290367126465" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="336.909423828125" y="-332.44757080078125"
           yaw="40.69894790649414" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="374.4433898925781" y="-278.4915466308594"
           yaw="69.65299987792969" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="382.3780517578125" y="-241.1703643798828"
           yaw="86.34180450439453" z="0.0" />
```

```
    <waypoint pitch="0.0" roll="0.0" x="382.4445495605469" y="-212.8241729736328"
          yaw="90.59859466552734" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="382.1245422363281" y="-182.1951904296875"
          yaw="90.59859466552734" z="0.0" />
    <waypoint pitch="360.0" roll="0.0" x="368.3731689453125" y="
         -172.0313720703125" yaw="180.3261260986328" z="0.195892333984375" />
    <waypoint pitch="360.0" roll="0.0" x="359.9306640625" y="-172.10787963867188"
          yaw="180.3261260986328" z="0.195892333984375" />
    <waypoint pitch="360.0" roll="0.0" x="339.1204833984375" y="
         -172.2149658203125" yaw="180.3261260986328" z="0.195892333984375" />
    <waypoint pitch="360.0" roll="0.0" x="325.8786315917969" y="
         -172.3017120361328" yaw="180.3261260986328" z="0.195892333984375" />
    <waypoint pitch="0.0" roll="0.0" x="310.8879699707031" y="-157.91519165039062
         " yaw="90.51094055175781" z="0.0043487548828125" />
    <waypoint pitch="0.0" roll="0.0" x="310.6393737792969" y="-133.03883361816406
         " yaw="90.51094055175781" z="0.0043487548828125" />
    <waypoint pitch="360.0" roll="0.0" x="301.86016845703125" y="
         -121.61925506591797" yaw="180.92404174804688" z="0.01959228515625" />
    <waypoint pitch="360.0" roll="0.0" x="269.64727783203125" y="
         -122.1388168334961" yaw="180.92404174804688" z="0.01959228515625" />
    <waypoint pitch="360.0" roll="0.0" x="243.0809326171875" y="
         -122.5027847290039" yaw="180.92404174804688" z="0.01959228515625" />
    <waypoint pitch="360.0" roll="0.0" x="231.92315673828125" y="
         -122.80830383300781" yaw="184.61492919921875" z="0.01959228515625" />
    <waypoint pitch="360.0" roll="0.0" x="201.91207885742188" y="
         -161.14862060546875" yaw="269.77020263671875" z="0.01959228515625" />
    <waypoint pitch="360.0" roll="0.0" x="187.185791015625" y="
         -173.09115600585938" yaw="180.3261260986328" z="0.195892333984375" />
    <waypoint pitch="360.0" roll="0.0" x="141.87583923339844" y="
         -173.34906005859375" yaw="180.3261260986328" z="0.195892333984375" />
    <waypoint pitch="360.0" roll="0.0" x="131.3639678955078" y="
         -184.7850341796875" yaw="272.4569396972656" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="137.26539611816406" y="
         -211.52774047851562" yaw="-64.18177795410156" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="156.52223205566406" y="
         -234.0313262939453" yaw="-35.032196044921875" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="190.98350524902344" y="
         -245.10235595703125" yaw="358.4046936035156" z="0.0386962890625" />
    <waypoint pitch="360.0" roll="0.0" x="214.16030883789062" y="
         -245.92640686035156" yaw="359.6054992675781" z="0.00433349609375" />
    <waypoint pitch="360.0" roll="0.0" x="243.9140167236328" y="
         -246.1312713623047" yaw="359.6054992675781" z="0.00433349609375" />
    <waypoint pitch="360.0" roll="0.0" x="271.29522705078125" y="
         -246.2991485595703" yaw="359.6054992675781" z="0.00433349609375" />
 </route>
 <route id="39" town="Town04">
    <waypoint pitch="0.0" roll="0.0" x="-257.4617919921875" y="-90.17411804199219
         " yaw="171.1091766357422" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-296.2134704589844" y="-86.86602783203125
         " yaw="179.0170135498047" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-351.7880859375" y="-68.75138092041016"
         yaw="-217.90353393554688" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-381.4906005859375" y="
         -24.760557174682617" yaw="-254.04220581054688" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-384.76947021484375" y="
         -3.012847900390625" yaw="-268.8101501464844" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-398.76190185546875" y="5.784707546234131
         " yaw="179.631591796875" z="0.0" />
    <waypoint pitch="0.0" roll="0.0" x="-430.2335205078125" y="6.185691833496094"
          yaw="-184.1747283935547" z="0.0" />
```

```
      <waypoint pitch="0.0" roll="0.0" x="-503.7024841308594" y="54.71406936645508"
           yaw="-242.7170867919922" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-513.7704467773438" y="103.92790985107422
          " yaw="90.3575210571289" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-515.2421875" y="242.3034210205078" yaw="
          89.47163391113281" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-506.9309997558594" y="296.92578125" yaw=
          "73.2252197265625" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-472.6380920410156" y="362.345458984375"
          yaw="51.44786071777344" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-428.0704650878906" y="403.25244140625"
          yaw="33.64748001098633" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-375.0228576660156" y="428.02935791015625
          " yaw="16.424245834350586" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-332.56890869140625" y="435.5860290527344
          " yaw="3.7612531185150146" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-292.7345886230469" y="435.9103088378906"
           yaw="-0.2081967145204544" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-181.4010772705078" y="434.91448974609375
          " yaw="-4.462084770202637" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-130.54518127441406" y="424.6565246582031
          " yaw="-18.345731735229492" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-78.3988037109375" y="399.085693359375"
          yaw="-33.8978271484375" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-33.48869705200195" y="358.66290283203125
          " yaw="-50.08180618286133" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="15.592141151428223" y="240.9547119140625"
           yaw="-84.6488037109375" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="16.422504425048828" y="199.58425903320312
          " yaw="-90.28913116455078" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="15.254642486572266" y="-46.86491394042969
          " yaw="-90.22486114501953" z="0.0" />
      <waypoint pitch="362.50775146484375" roll="0.0" x="82.86859893798828" y="
          -103.56114959716797" yaw="5.0680694580078125" z="0.8531252145767212" />
      <waypoint pitch="365.38226318359375" roll="0.0" x="118.45911407470703" y="
          -85.75933837890625" yaw="48.07890319824219" z="3.9298620223999023" />
      <waypoint pitch="364.364990234375" roll="0.0" x="132.3380584716797" y="
          -48.34086227416992" yaw="91.20783233642578" z="8.078770637512207" />
      <waypoint pitch="361.8727111816406" roll="0.0" x="119.09281158447266" y="
          -13.37026309967041" yaw="130.28089904785156" z="10.349198341369629" />
      <waypoint pitch="0.0" roll="0.0" x="25.08895492553711" y="6.18464469909668"
          yaw="-179.76736450195312" z="11.0" />
      <waypoint pitch="-1.580212950706482" roll="0.0" x="-286.66015625" y="
          5.562638282775879" yaw="-179.9231719970703" z="1.041011929512024" />
      <waypoint pitch="-0.03770020976662636" roll="0.0" x="-360.4718933105469" y="
          5.538497447967529" yaw="179.631591796875" z="0.0005925323348492384" />
      <waypoint pitch="0.0" roll="0.0" x="-400.5914611816406" y="5.796472072601318"
           yaw="179.631591796875" z="0.0" />
      <waypoint pitch="0.0" roll="0.0" x="-435.8338623046875" y="6.7710137367248535
          " yaw="-187.75843811035156" z="0.0" />
   </route>
   </routes>
```

Aleksander Scherman Olsen

NTNU

Kunnskap for en bedre verden