Reinforcement Learning for Unstructured Driving

MS Robotics Student: Joseph Lowman

Faculty Supervisor: Professor Tulga Ersal

May 4, 2020

1 Abstract

Reinforcement learning is a recent method for solving unsupervised learning tasks without domain knowledge. This project applied a model-free reinforcement learning algorithm to solve a motion planning problem for autonomous driving in an unstructured environment. A 2D environment simulated the driving scenario, along with a reward function to define rewards for each state transition. A model-free, policy gradient algorithm, TD3, was evaluated on the driving environment. The algorithm learned policies to navigate from a randomized starting state to a 2D goal position. Videos of learned policies can be seen at `https://umich.box.com/s/04ef1bwx88j3n5fihm3nsecw3a8b2myv`. Code for the environment implementation is available at `https://github.com/jdlowman2/collision_imminent_env`. Code for the TD3 algorithm, forked from the original authors [1], is available at `https://github.com/jdlowman2/TD3`.

2 Introduction

Motion planning in autonomous driving requires extensive domain knowledge of vehicle dynamics and optimization control techniques such as model predictive control. An exciting alternative, reinforcement learning, proposes to solve the planning problem with a data driven approach. Several different reinforcement learning strategies have emerged in recent years. In the broadest sense, they can be classified as either model-based or model-free.

Model-based algorithms typically attempt to use networks to approximate system dynamics, the system controller, or both. Model-based algorithms attempt to train a deep network as a function approximator of the system dynamics, effectively predicting how the system will evolve given a set of inputs. Such a network can be computationally more efficient and accurate than traditional methods. Using this network, optimal control methods can be applied to the learned model of the world. Alternatively, known system dynamics can be employed to model how the system evolves, and the controller can be learned. Finally, a third approach is to learn both the system dynamics and the controller via deep networks acting as function approximators of each.

On the other side of the reinforcement learning spectrum are model-free reinforcement learning approaches. Rather than taking the traditional approach of modeling the system and applying a controller to the model, the model free approach attempts to directly model an action policy, which can predict the best action to take from each state within the environment. This approach does not explicitly model how the system states transition once given action inputs.

The most exciting advantage of the model free approach is the ability to create a useful controller without needing specialized domain knowledge for the particular problem at hand.

This research project aims to explore how model-free reinforcement learning can be applied to driving scenarios. Future work may compare the reinforcement learning method presented here with optimal controls methods.

2.1 Model-Free Reinforcement Learning

Within model-free reinforcement learning, two broad categories of algorithms are Q-learning and policy gradient methods. Q-learning algorithms attempt to model the 'Q value' of a particular state, which corresponds to the future expected reward from that state. Then, the agent can simply choose an action which will result in the maximum Q state. Alternatively, policy gradient methods attempt to learn an action policy explicitly by using gradient ascent.

A third, more recent method, called actor-critic, attempts to merge aspects of Q-learning and policy gradients. In essence, actor-critic methods employ two networks; one as the actor, which learns a policy, and the other network as the critic, which learns the Q value. One commonly used algorithm is deep deterministic policy gradients (DDPG). DDPG is particularly well suited for environments with continuous valued action spaces. DDPG is explained in more detail in section 4.2.

3 Background

Previous work developed MPC models for vehicle dynamics in collision imminent steering (CIS) maneuvers. Specifically, a nonlinear MPC model allowed for steering commands to avoid collisions along either straight or curved roads [2]. The method used Euler integration to optimize a trajectory over a finite time horizon. The objective function minimized time spent by the vehicle out of the initial lane, and the optimization subjected the vehicle to several constraints including steering angle and rate limits, tire slip constraints, and final state stability constraints.

Learned methods for driving have also seen some success in recent works. End to end learning methods, which map perception input directly to controller outputs, have demonstrated some success [3] [4].

One particularly relevant challenge to reinforcement learning is the choice and design of the reward function. In initial research involving Atari games, the reward function was relatively simple and straightforward, relating to score of the game. In motion planning for driving, it is less clear how to define the reward function and incorporate the most relevant aspects of the system. Strategies range from very simple, such as rewarding the final state, to more complex, such as incorporating multiple rewards into an engineered reward function [5].

Collision imminent scenarios have also been tested using game-engine based simulators (CARLA [6]) in order to accurately model many aspects of the environment [7]. The authors used a variational autoencoder and recurrent neural network to preprocess state data to a latent state space, before passing the latent variable to the actor critic networks. This approach has the benefit of realistic modeling, however it also requires extensive computational resources to render the environment and train networks through simulation. Our project proposes a 2D simulator to provide insight into the intricacies of reinforcement learning without the computational requirements of a realism simulator.

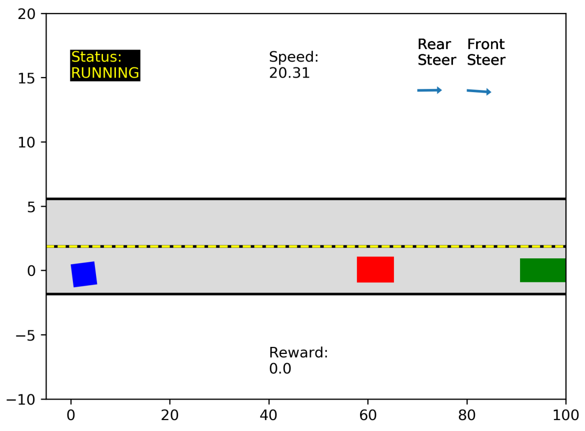

Figure 1: Environment for the task. The autonomous vehicle is represented in blue and starts at the left side of the environment. The goal region is designated with the green box. Throughout the simulation, the front and rear steering angles are represented by the arrows in the upper right corner. The red box represents a stationary obstacle. The reward for each timestep is displayed in the lower center.

4 Methods

4.1 Environment Simulator

A special purpose environment simulator was constructed to train the agent and evaluate performance. The environment was built using the OpenAI Gym framework. A visualization of the environment is shown in figure 1. Key design choices include design of the reward function, observation space, and action space. Each of these choices is explained in further detail in the following sections.

4.1.1 Nonlinear Vehicle Dynamics Model

The vehicle model consisted of a nonlinear 3 degree of freedom bicycle model which incorporated all wheel steering. The model specifies maximum steering angles of ±35 degrees and ±10 degrees for the front and rear wheels respectively. For each step of the environment, the vehicle motion was simulated using the Euler method $\mathbf{x_{t+1}} = \mathbf{x_t} + \Delta t \dot{\mathbf{x}}_\mathbf{t}$.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ \omega \\ \delta_f \\ \delta_r \end{bmatrix} \tag{1}$$

$$u = \begin{bmatrix} \dot{\delta_f} \\ \dot{\delta_r} \end{bmatrix}$$

$$\frac{\mathbf{dx}}{dt} = \begin{bmatrix} u\cos(\psi) - v\sin(\psi) \\ u\sin(\psi) + v\cos(\psi) \\ \omega \\ 0 \\ -u\omega + \frac{F_{y,f}\cos(\delta_f)+F_{y,r}\cos(\delta_r)}{m} \\ \frac{F_{y,f}\cos(\delta_f)l_f+F_{y,r}\cos(\delta_r)l_r}{I_{zz}} \\ \dot{\delta}_f \\ \dot{\delta}_r \end{bmatrix}$$

$$F_{y,-} = \mu F_{z,-}\sigma_{y,-}$$

$$\sigma_{y,-} = -\sin(C\arctan(B\frac{V_{y,-}}{V_{x,-}}))$$

$$V_{x,-} = u\cos(\delta_-) + (v + \omega l_-)\sin(\delta_-)$$
$$V_{y,-} = -u\sin(\delta_-) + (v + \omega l_-)\cos(\delta_-)$$

The parameters used for each constant in the vehicle and tire models are specified in the appendix, see table 3.

4.1.2 Action Space

The action space consisted of two dimensions, $a_1 \in [-1, 1]$ and $a_2 \in [-1, 1]$, corresponding to the front steering angle rate $\dot{\delta}_f$ and the rear steering angle rate $\dot{\delta}_r$. The environment incorporated a scaling factor such that the input actions were to scaled to maximum magnitude of 1.

$$\dot{\delta}_f = \frac{a_1 + 1}{2}(\dot{\delta}_{f,max} - \dot{\delta}_{f,min}) + \dot{\delta}_{f,min}$$

$$\dot{\delta}_r = \frac{a_2 + 1}{2}(\dot{\delta}_{r,max} - \dot{\delta}_{r,min}) + \dot{\delta}_{r,min}$$

Maximum steering rates were defined as ± 70 degrees per second and ± 35 degrees per second for the front and rear wheels respectively. Likewise, a maximum steering angle of ± 70 degrees and ± 35 degrees was enforced. The vehicle's speed, u, was not controllable by the agent and remained fixed within each episode, as specified by equation 1.

4.1.3 Observation Space

The observation space consisted of a column vector of state values. In each experiment, the first eight components were the eight components of the vehicle state (equation 1). Results using this setup, without obstacles, are plotted in figure 3. For testing with obstacles included, the x and y position of each obstacle were concatenated to the state vector, creating a ten dimensional state vector.

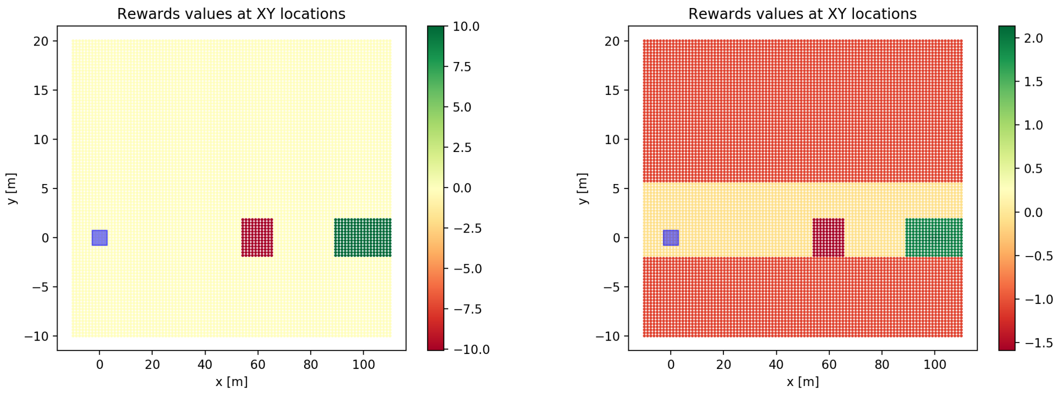

Figure 2: Visualization of two different reward functions. The vehicle starts at (0,0) and the goal is to get to (100,0). Each x, y point is colored according to the corresponding reward for that position. Left, a sparse reward function. Reward value for the vehicle in each (x, y) position throughout the environment are a small constant negative value. Reward for positions in collision with the obstacle are a large negative value, and rewards for positions within the goal region are a large positive value. Right, an expressive reward function. Positions in the goal region achieves 2.0 reward. Positions within the lane and not in collision with the obstacle achieves -0.1 reward. Collisions with the obstacle result in -1.5 reward. Leaving the boundaries of the lane results in -1.0 reward.

4.1.4 Reward Function

The reward function for the environment is a critical component which can greatly influence whether the algorithm succeeds or not. Many previous methods attempt to use "reward engineering" to achieve desirable behavior. This method can lead to dangerous agent actions, as peculiarities in the reward function can cause the agent to exploit undesirable behaviors ([8]). Alternatively, one can specify a sparse reward function, and only return a positive reward once the agent has achieved the goal state. However, sparse reward functions can reward the agent too infrequently, making it difficult to learn a policy without converging to local optima.

Two possibilities for reward function were considered for this driving environment. A sparse reward function specified a $+1.0$ reward for reaching the goal region, and a -0.1 penalty for each timestep not in the goal region. The positive reward for the goal region only considered the (x, y) components of the vehicle state and did not consider vehicle heading. Note that this reward function gives the agent the same reward whether the vehicle is within the drivable region or not. Therefore, the driving environment can be considered an unstructured environment. The agent may choose to navigate around the obstacle in either direction, and the road shown in figure 1 is merely a visualization tool.

A more expressive reward function was also tested. To encourage learning, the reward function was a superposition of four component functions. One corresponds to a positive reward in the goal region, one corresponds to a negative reward outside of the driveable region, one corresponds to a negative reward for collisions with the obstacle, and finally a constant, small negative constant is added to the total reward. The corresponding reward function map can be seen in figure 2.

4.1.5 Environment Initialization and Termination

The environment is initialized with the vehicle in a stochastic state with the following parameters.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ \omega \\ \delta_f \\ \delta_r \end{bmatrix} \in \begin{cases} [0, 10] \\ [0, 2.8] \\ [-0.087, 0.087] \\ [20, 30] \\ \{0\} \\ \{0\} \\ [-0.087, 0.087] \\ [-0.017, 0.017] \end{cases}$$

The obstacle and goal are initialized with positions

$$\mathbf{x_{obs}} = \begin{bmatrix} x = 55.0 \\ y = 0.0 \end{bmatrix}$$

$$\mathbf{x_{goal}} = \begin{bmatrix} x = 100.0 \\ y = 0.0 \end{bmatrix}$$

The environment termination conditions for each episode are defined as follows. The episode is terminated when the vehicle leaves the render area, analogous to driving very far away from the drivable area, or collides with the obstacle. The environment is not terminated when the vehicle enters the goal region, as we wish to discourage actions which briefly enter the goal region, then collide with an obstacle afterward.

4.2 Algorithms

In continuous action spaces, DDPG is a common choice for reinforcement learning [9]. Here, we used twin-delayed DDPG (TD3), an extension of DDPG, using the implementation provided by [1] at `https://github.com/sfujim/TD3`.

4.2.1 Deep Determinisitic Policy Gradients (DDPG)

Deep Determinisitic Policy Gradients (DDPG) uses an actor-critic framework, composed of two function approximators. The actor attempts to approximate the best action to take from a given state. The critic attempts to approximate the total future reward for a state action pair.

The full pseudocode of DDPG is attached in the appendix. Denote the actor network as $\mu(s|\theta^{\mu})$, the critic network as $Q(s, a|\theta^{Q})$, and the target networks as $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$ respectively.

During training, the algorithm samples sequences (s_i, a_i, r_i, s_{i+1}) from the environment, representing the starting state, action taken, ending state, and reward gained from the transition. Each sequence is added to a replay memory buffer. At regular intervals, a batch of sequences is removed from the memory buffer and used to update both actor and critic networks. When the buffer reaches maximum capacity, the oldest sequences are discarded.

The loss for the critic is computed by comparing the expected reward y_i from the target networks and using the mean squared error with the expected reward $Q(s_i, a_i|\theta^{Q})$ of the critic network. The critic is updated via gradient descent.

$$\begin{aligned} y_i &= r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1})|\theta^{\mu'})|\theta^{Q'}) \\ L_{critic} &= \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2, \end{aligned} \tag{2}$$

Here, Q represents the critic, Q' represents the target critic, μ represents the actor, and μ' represents the target actor.

The actor is updated using the gradients of the critic network $\nabla_a Q(s, a|\theta^Q)|$ and the gradient of the actor network $\nabla_{\theta^\mu}\mu(s|\theta^\mu)$. Since the objective is to maximize the reward obtained by the actor, the actor network is updated by gradient ascent.

$$\nabla_{\theta,\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{3}$$

4.2.2 Twin Delayed Deep Determinisitic Policy Gradients (TD3)

Twin Delayed Deep Determinisitic Policy Gradients (TD3) is a successor to DDPG ([1]). The critic network in DDPG has been shown to overestimate the value of state action pairs over time. TD3 adds an additional critic network and uses the critic network which produced minimum error with the actor for each update step of the critic networks. The full pseudocode of TD3 is included in the appendix. Note the critic network parameters are updated via the minimum loss computed by the twin critic networks.

4.3 Experimental Results

Testing was performed starting with as simple a setup as possible and progressively increasing the "difficulty" for the agent. In each experiment, a new actor critic network was trained, with no transfer of learned weights between experiments. Initial tests used an environment with no obstacles. In this case, the vehicle is initialized with a uniformly distributed, random heading in the range ± 10 radians. The training curve is shown in figure 3. Qualitatively, the agent learned to maintain roughly an average zero heading in order to reach the goal. However, it also displayed an oscillating behavior as the network chose actions on either extreme of the action space. To compensate, a steering rate limit was enforced in further experiments.

Next, an obstacle was added, and the obstacles x and y positions were concatenated into the state vector. Initial experiments used a larger range of vehicle states for environment initialization. However, initial experiments were unsuccessful. Subsequently, the range of vehicle states were limited to those specified in section 4.1.5.

Results from multiple experiments are shown in table 1. In fact, the sparse reward function typically resulted in an agent which was able to reach the goal more consistently. A visualization of vehicle trajectories is shown in figure 5.

4.4 Spread Environment

To test the environment further, the initialization vehicle states were expanded, in a "spread environment". The initial states of the vehicle were initialized uniformly from the ranges specified in equation 4. Results are shown in figure 6 and table 2. The agent achieved a lower average reward, as some initializations were much more difficult than in prior experiments. Additionally, initial states placing the vehicle near the boundaries of the environment resulted in a learned policy which

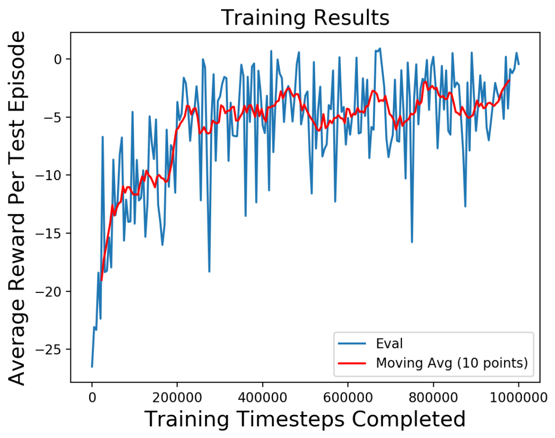

Figure 3: At every 5,000 timesteps of training, the agent was evaluated over 10 test episodes. This plot shows the resutls of TD3 trained in the environment simulator without obstacles and with the expressive reward function. The state vector consisted of the 8 vehicle states only. The agent learned to stay roughly within the drivable area, although it's trajectory was not smooth and tended to oscillate heading. The learned policy can be seen at `https://umich.box.com/s/2c0bxqzusuuz7tc78bfcrj0oqk5j3b9q`.

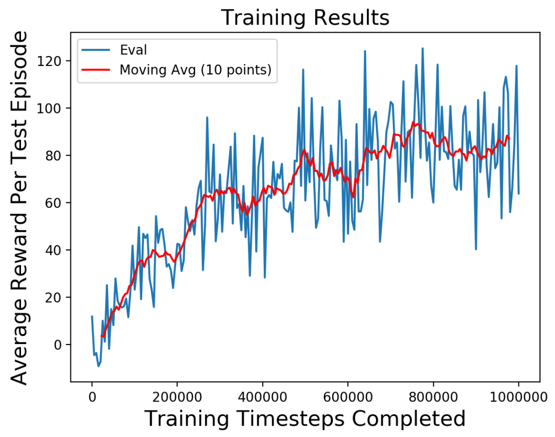

Figure 4: TD3 trained in environment simulator with one obstacle. The state vector consisted of the 8 vehicle states, the x position of the obstacle, and the y position of the obstacle. The sparse reward function was used.

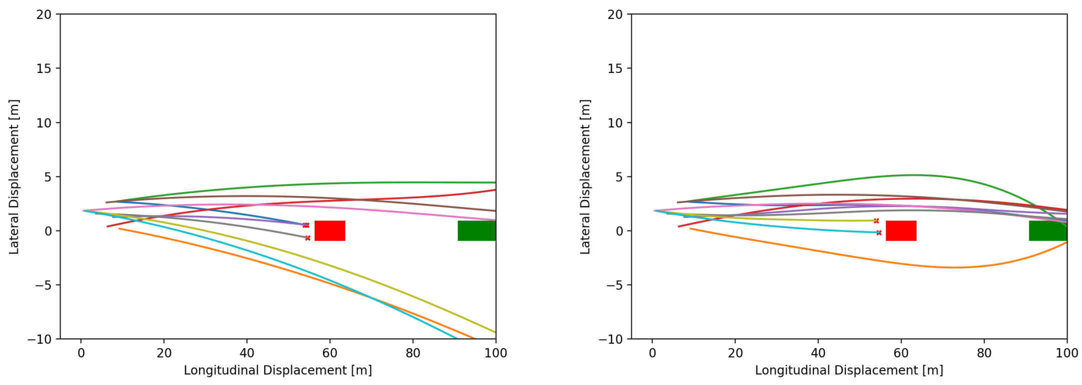

Figure 5: A visualization of 10 different paths taken by the vehicle. Left, an untrained agent. Right, the trained TD3 agent using the sparse reward function. Note that for initial conditions with negative heading, the agent chooses to navigate to the right of the obstacle, whereas for positive initial headings, the agent chooses to navigate to the left of the obstacle. Also, note not all initial conditions allow the vehicle to successfully avoid the obstacle, as the agent has no direct control over the velocity.

Algorithm	Reward	Vehicle Speed Range	Steering Rate Limits (front, rear)	Reached goal 500 episodes	Avg Reward (stdev) 500 episodes
TD3	Sparse	[10, 20]	±700, ±300	78.8%	109.21 (71.26)
TD3	NonSparse	[20, 30]	±700, ±300	50.4%	-5.81 (18.95)
TD3	Sparse	[20, 30]	±700, ±300	7.8%	0.57 (40.39)
DDPG	Sparse	[20, 30]	±700, ±300	57.0%	-4.57 (19.91)
TD3	Sparse	[20, 30]	±70, ±30	85.6%	88.44 (61.02)
TD3	NonSparse	[20, 30]	±70, ±30	55.0%	-0.67 (22.81)
Untrained	Sparse	[20, 30]	±70, ±30	2.0%	-8.19 (15.21)
Untrained	NonSparse	[20, 30]	±70, ±30	11.0%	-33.79 (30.47)

Table 1: Experiment details. In each experiment, one obstacle is present. Initial experiments allowed large steering rates and initialized the vehicle with slower speeds. Successive experiments increased the difficulty by increasing vehicle speed and reduced steering rate limits. The final two rows show baseline comparisons with untrained agents. The environment is initialized with a uniformly distributed vehicle state as noted in equation 4. Due to the initial state, some scenarios may not have a feasible solution where the vehicle is able to avoid the obstacle and enter the goal region.

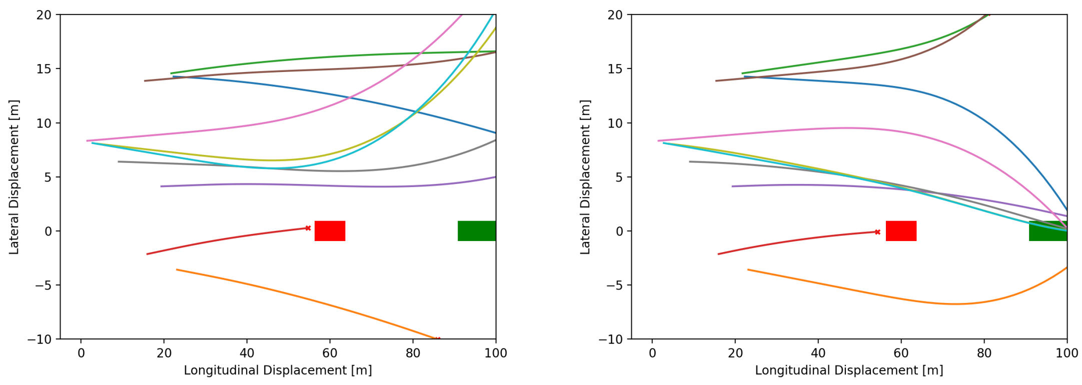

Figure 6: Vehicle paths when the initial state of the vehicle is uniformly distributed over a large area. Left, an untrained agent. Right, a trained agent using the sparse reward function. The trained agent learns to apply steering commands in order to pass through the goal region. Under some initial conditions, the agent has learned to terminate the episode by driving out of the environment (right, top paths, shown in green and brown, exiting at (80, 20)).

Algorithm	Reward	Vehicle Speed Range	Steering Rate Limits (front, rear)	Reached goal 500 episodes	Avg Reward (stdev) 500 episodes
TD3	Sparse	[20, 30]	±70, ±30	75.6%	54.6 (63.52)
Untrained	Sparse	[20, 30]	±70, ±30	5.4%	-5.56 (8.61)

Table 2: Experiment details for the spread environment. Similar to the results in table 1, some initial states may not have a feasible solution.

drove the vehicle out of the environment, terminating the episode (see figure 6).

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ \omega \\ \delta_f \\ \delta_r \end{bmatrix} \in \begin{cases} [0, 25] \\ [-5, 15] \\ [-0.087, 0.087] \\ [20, 30] \\ \{0\} \\ \{0\} \\ [-0.087, 0.087] \\ [-0.017, 0.017] \end{cases} \tag{4}$$

5 Discussion

The TD3 algorithm applied to the environment simulator produced an agent which was able to reach the goal position.

5.1 Reward Function Choice

The environment's reward function is crucial to the success of this method. The non-sparse reward function was quite restrictive in the actions it allowed the actor to take without penalty. For example, the actor was punished by the reward function if it took a path to the right of the obstacle, such as the orange path in the left plot of figure 5. On the other hand, the sparse reward function did not penalize leaving the road boundaries, and instead only rewarded reaching the goal region.

It may be possible to design a more instructive reward function to encourage exploration in promising regions of the environment, increase sample efficiency, and decrease overall training time. For example, a shortest path trajectory, without consideration of vehicle dynamics, may provide a low computational cost estimate of good policies.

5.2 Undesirable Learned Policies

Figures 5 and 6 show the paths the agent takes before and after training, using different initialization distributions of the vehicle state. Figure 6 demonstrates an interesting policy shown in the brown and green paths originating at approximately $(21, 15)$. These paths seem to show the agent exiting the environment quickly, thereby terminating the episode. One hypothesis for this behavior is the agent cannot reach the goal from these starting conditions. Therefore, the least expensive behavior is to avoid accumulating negative reward by leaving the environment. This highlights the importance of the simulation environment setup. The agent learns to exploit the simulation wherever possible, and this may or may not reflect real world conditions.

5.3 Driving Behavior

In many cases, the agent reached the goal with unconventional driving techniques, such as driving and sliding close to the obstacle. Clearly, these behaviors are troubling for real world applications, and a real world system cannot rely solely on the techniques presented here.

One critical downside to the model-free approach is the opacity of the actor network. After training, the actor network maps a high dimensional, continuous state space to an action. Small variations in the state space may produce drastically different actions.

6 Conclusion

Reinforcement learning is a powerful technique with enormous potential to generalize across many domains. Here, we demonstrated a environment setup for a driving scenario incorporating a non-linear vehicle dynamics model. We applied a model-free reinforcement learning algorithm and observed the agent learned to steer the vehicle to a goal position. However, this approach required an environment with a well designed action space, observation space, reward function, and initialization conditions. Additionally, the learned policies had no guarantee of safety and often demonstrated unusual driving behavior.

7 Future Work

The environment proposed here is limited in generalizability across different driving scenarios. Future work may attempt to extend the environment by incorporating a larger range of initialization parameters, multiple obstacles, and varying goal positions. Additionally, the environment may be extended to include an action space dimension corresponding to acceleration. The reward function may be modified to promote better driving behavior, such as penalizing threshold values of tire forces to prevent sliding maneuvers. Alternatively, the reward function may incorporate a heuristic, such as shortest path without dynamics, in order to better promote realistic driving behavior.

Safely applying reinforcement learning in motion planning is an active area of research, as the reinforcement learning paradigm requires exploration of the environment during training, which may lead to unsafe trajectories. Several recent works may be incorporated to improve safety. Alshiekh et al proposed a "shielding" system which monitors the agents actions and rejects unsafe actions[10]. Koller et al proposed a learning-based MPC method for probabilistically safe reinforcement learning [11]. Their method designated a safe region in which the system is understood and a safe controller can operate. The goal for the system was to learn a policy to operate outside of the safe region.

8 Acknowledgements

Thanks to Professor Tulga Ersal and John Wurts for insight and ideas into the problem.

References

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

[2] J. Wurts, J. L. Stein, and T. Ersal, "Minimum slip collision imminent steering in curved roads using nonlinear model predictive control," in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 3975–3980.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[4] K. Makantasis, M. Kontorinaki, and I. K. Nikolos, "A deep reinforcement-learning-based driving policy for autonomous road vehicles," *CoRR*, vol. abs/1907.05246, 2019. [Online]. Available: http://arxiv.org/abs/1907.05246

[5] A. Abels, D. M. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher, "Dynamic weights in multi-objective deep reinforcement learning," *arXiv preprint arXiv:1809.07803*, 2018.

[6] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[7] H. Porav and P. Newman, "Imminent collision mitigation with reinforcement learning and vision," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 958–964.

[8] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," 2016.

[9] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[10] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 6059–6066.

[12] J. C. Kegelman, L. K. Harbott, and J. C. Gerdes, "Insights into vehicle trajectories at the handling limits: analysing open data from race car drivers," *Vehicle System Dynamics*, vol. 55, no. 2, pp. 191–207, 2017. [Online]. Available: https://doi.org/10.1080/00423114.2016.1249893

[13] J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, "Improving the robustness of an mpc-based obstacle avoidance algorithm to parametric uncertainty using worst-case scenarios," *Vehicle System Dynamics*, vol. 57, no. 6, pp. 874–913, 2019.

[14] ——, "A nonlinear model predictive control formulation for obstacle avoidance in high-speed autonomous ground vehicles in unstructured environments," *Vehicle System Dynamics, International Journal of Vehicle Mechanics and Mobility*, vol. 56, pp. 853–882, 2018.

[15] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, 2017, pp. 908–918.

A Vehicle Parameters

Parameter	**Value**
Width	1.8 m
Length	4.8 m
Mass	2041 kg
I_{zz}	4964
L_f	1.56
L_r	1.64
$F_{z,f}$	51.40%
$F_{z,r}$	48.60%
μ	0.8
B	13
C	1.285
Δt	0.05

Table 3: Vehicle and tire parameters used in the environment simulation.

B Algorithm Pseudocode

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
**for** episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  **for** t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
        $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
    Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  **end for**
**end for**
```

Figure 7: Pseudocode of the DDPG algorithm from [9].

Algorithm 1 TD3

```
Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$
with random parameters $\theta_1$, $\theta_2$, $\phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
**for** $t = 1$ **to** $T$ **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
  Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

  Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  **if** $t$ mod $d$ **then**
    Update $\phi$ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  **end if**
**end for**
```

Figure 8: Pseudocode of the TD3 algorithm from [1].