navTD3: A Novel Deep Deterministic Reinforcement Learning Method for Autonomous Vehicle Navigation

Dogan C. Cicek, Furkan B. Mutlu, Enes Duran, and Suleyman S. Kozat, *Senior Member, IEEE*

***Abstract*— Current state-of-the-art deterministic deep reinforcement learning (RL) algorithms that work on continuous action space may fail in problem-specific tasks such as autonomous driving. In this work, we investigate why these algorithms perform poorly and introduce methods to remedy the identified problems. We present an end-to-end deep RL algorithm, navTD3, for autonomous driving tasks. To make our approach more applicable for different types of autonomous vehicles and perceive the dynamics of the vehicle that our algorithm navigates, navTD3 controls the throttle rather than velocity. Moreover, classic deep RL methods struggle, especially at the beginning of the training on autonomous driving tasks. In our algorithm, we develop a new exploration process to cover the state and action space of the task correctly and present a novel regularization technique called Tanh Regularization to prevent gradients from vanishing in the early stages of the training. We verify navTD3 on an autonomous ground vehicle equipped with an IMU sensor, three depth cameras, and Bird's Eye View map in a simulated office environment. We compare our approach with state-of-the-art deep deterministic RL algorithms, Deep Deterministic Policy Gradients(DDPG), and Twin Delayed Deep Deterministic Policy Gradients(TD3). navTD3 tackles the problems that DDPG and TD3 suffer and outperforms DDPG and TD3 in terms of the final performance, sample efficiency, and generalization ability.**

***Index Terms*— Autonomous agents, reinforcement learning, collision avoidance, autonomous vehicle navigation, deep learning methods**

I. Introduction

Delegating the task of driving to machines has been an interest of humankind for decades. With the prowess of the learning algorithms on various control tasks [1], [2], the research on this problem is oriented more towards machine learning. One important and widely studied application area of autonomous driving is indoor navigation for office-like environments [3], [4].

Indoor navigation is challenging due to highly dense environments such as houses and offices. The need for models of fine-grained precision becomes evident in those cases. Therefore, the learning algorithms should be able to perceive the complex environment and extract the essential cues simultaneously for the navigation task. Another complication originates from the nature of the driving task. Since driving includes numerous sub-tasks such as path planning, obstacle avoidance, localization, etc., the model's performance depends heavily on performance of the agent on sub-tasks. Instead of creating a comprehensive training data set to solve sub-tasks simultaneously, they can be solved simultaneously by utilizing Reinforcement Learning (RL) [3], [4]. At that point, the reward-shaping process plays a huge role in tackling each sub-task successfully since optimizing the given reward configuration must lead to the desired policy. In practice, it is often arduous to find the proper reward setting to solve these tasks. Also, sparse reward signals deteriorate the learning abilities [4].

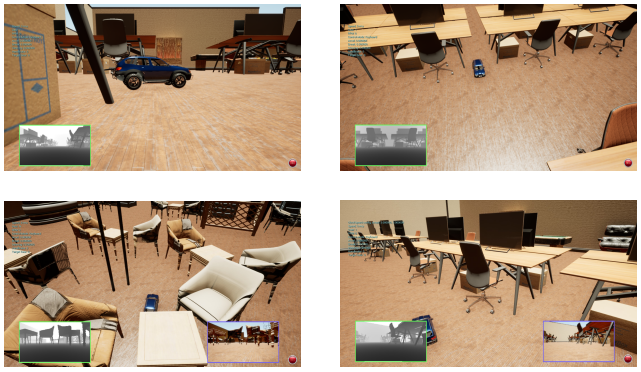


Fig. 1: Examples from the scene. Sub-windows display current depth camera sensor output

Recently, several works have demonstrated the effectiveness of the RL agents in continuous control tasks [3]–[6]. One of the research trends in RL is utilizing it for autonomous driving tasks, which can be defined as a particular deterministic continuous control task. This area prompted wider usage of RL algorithms, not only benefiting in controlling the vehicles but also regulating the vehicle-to-vehicle interactions and ride-sharing systems [7]–[9]. One of the main reasons for the wide usage of the RL algorithms is decreasing the dependency of the learning process on the dataset. RL algorithms do not require expert demonstrations to obtain a decent policy [10].

In this paper, we present navTD3, an end-to-end deep RL method for indoor navigation tasks. Our method is suitable for hardware deployment. We summarize our main contributions as follows:

- In contrast to the recent approaches [11], [12], our algorithm controls acceleration rather than the velocity of the vehicle. By doing so, we enable our algorithm to learn the vehicle's dynamics. Also, we annihilate the necessity of a component that adjusts the vehicle's velocity with respect to the controlled vehicle.

The first two authors contributed equally to this work. Dogan C. Cicek, Furkan B. Mutlu, and Suleyman S. Kozat are with Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey. Enes Duran is with Department of Computer Science, University of Tübingen, Tübingen, Germany(e-mail: cicek@ee.bilkent.edu.tr, mutlu@ee.bilkent.edu.tr, enes.duran@student.uni-tuebingen.de, kozat@ee.bilkent.edu.tr)

- We propose a novel regularization technique, Tanh Regularization, to prevent the agent from selecting actions from the extreme points of the action-space of the task and balance the action distribution of the transitions stored in the replay buffer.
- We propose a novel feature representation for observation space. Using cyclic features increases the generalization performance and learning pace of our algorithm.
- We design a realistic simulation environment that we use to provide different types of navigation tasks to the agents and assess the performance of our algorithm.

The paper is organized as follows: We first give the previous related works on autonomous driving and background information. We describe our algorithm and give details about reward shaping and training procedure. Lastly, we share the results of a virtual environment along with the evaluation.

II. Related Works

A. Learning Based Autonomous Driving Methods

Learning-based algorithms are recently become highly popular to solving autonomous driving tasks. These algorithms propose directly learning a mapping function from high dimensional input space to low dimensional action space [13]. Learning-based algorithms may address the autonomous driving task as an end-to-end learning problem. End-to-end learning approaches aim to enable function approximators to learn the intermediate feature representations by means of no human interaction. Through this, function approximators learn which features are important for the downstream task and pay greater attention to those features [14]. In addition, by eliminating human interaction, another source of bias is eliminated. One prominent example of end-to-end learning is learning a driving policy by training neural networks via gradient descent family algorithms. The Imitation Learning(IL) approach is one of the most popular subsets of learning-based algorithms to solve autonomous driving tasks [15]–[17]. Imitation Learning aims to imitate an expert by observing interactions with the environment while performing the task. The most well-known application of the IL is behavioral cloning [15], [16]. It formulates the IL task as a supervised learning task [17]. Given a dataset of expert demonstrations, behavioral cloning trains a driving policy that ideally mimics the expert policy in every state [16], [17]. The first applied case of behavioral cloning is ALVINN [15]. The hardware-deployed model takes a flattened gray-scale image as input and outputs the discrete steering angle to drive the vehicle. This work has shown the potency of learning in autonomous driving tasks and has drawn interest in the field. However, the model is too simple for solving the autonomous driving problem and only works for secluded roads. Consequently, employing depth information merely as gray-scaled images hinders the agent to learn a policy involving long-term plans and is not sufficient to solve complex tasks. Therefore, providing more information by adding another source of data is necessary as the given task is getting more complex. Hence, in our work, we utilize the Bird's Eye View (BEV), which can be defined as a top-down view of a part of the 3D scene. This approach is widely used in previous learning-based autonomous driving methods due to its effectiveness in autonomous driving tasks [18]–[21].

B. Reinforcement Learning

RL is a paradigm of machine learning where the agents learn to maximize their expected cumulative reward through trials and errors by interacting with the environment [10]. At each discrete time step, t, the agent receives the state information $s_t \in S$ from the environment. Then, the agent selects an action $a_t \in A$, depending on its policy, $\pi(s_t)$. After this action, the agent receives the next state information s_{t+1}, and the reward, r_t indicating how valuable the action taken by the agent in s_t.

In real-world applications, the possible state-action pairs are numerous, making the visitation of every state-action pair practically impossible [22], [23]. Therefore, an agent should perform a good generalization on similar states while distinguishing the differences. Utilizing the neural networks as function approximators remedies one of the main drawbacks of RL. By employing parameterized value function and policy, joint usage of Deep Learning and RL has shown remarkable success in solving high dimensional deterministic continuous control tasks [5], [24]. The Deep Deterministic Policy Gradients(DDPG) is the first deep RL algorithm that tackles deterministic continuous control tasks [24]. It adopts two networks called the Actor and the Critic that are responsible for approximating the policy and value functions, respectively. The Critic-Network guides the Actor-Network throughout the learning by estimating the value function given the state-action pair [24]. While this approach proved to be effective, it is known for overestimating the value function, which deteriorates the performance of the algorithm in particular tasks [5]. Twin Delayed Deep Deterministic Policy Gradients(TD3) algorithm addresses the overestimation error by taking the minimum of two Critic-Networks [5]. These networks are independently initialized and updated through the same set of transitions. Reducing the overestimation error proved to be pivotal for the agent performance [5].

Several methods are developed through the combination of point-to-point (P2P) navigation and sampling-based path planning (probabilistic roadmaps, rapidly exploring random trees) [3], [4], [25], [26]. One such work, AutoRL, is an end-to-end learning method that tackles long-range navigation tasks by training a P2P navigation model and using it for probabilistic roadmap construction. In P2P navigation tasks, the trained agent takes LiDAR data and location information to predict the wheel velocities of the differential drive indoor robot vehicle. Then, based on the success rates in navigation between points, it constructs a roadmap for the environment. The algorithm iteratively searches for the best reward-shaping strategy and the optimal network architecture for the agent, which play fundamental roles in performance. This optimization proved to be effective. However, it is extremely time consuming even in small search spaces [3]. Similar to several works in this line, AutoRL requires

constructing road-maps for each new environment [3], [4], [25]. This is also highly time-consuming, making it laborious for testing in a real-world case. Another issue with these speed-controlling policies is that they require different rule-based robot controller modules for simulation and hardware deployment due to the gap between simulation and reality.

III. NAVTD3

A. POMDP Setup and Reward Shaping

The state information that we propose for the task is given as follows:

$$\boldsymbol{s} = \left(\boldsymbol{s}_d^k, \boldsymbol{s}_m^k, \boldsymbol{s}_v, \boldsymbol{s}_a, \boldsymbol{s}_g, cos(\theta), sin(\theta)\right) \in S. \quad (1)$$

This includes outputs from various sensors. $\boldsymbol{s}_d^k$ and $\boldsymbol{s}_m^k$ are 3-D tensors that represent sensor outputs of the horizontally concatenated 3 depth cameras in the last k frames and information received from the minimap in the last k frames. Stacking the last k frames to produce inputs provides a significant decrease in the partial observability of autonomous driving tasks. This enables the agent to extract some of the crucial state information such as speed, orientation, and direction for the vehicle. $\boldsymbol{s}_v$ is a 1-D vector that represents the angular and linear velocity of the vehicle according to the goal. $\boldsymbol{s}_a$ is a 1-D vector that represents the angular and linear acceleration of the vehicle according to the goal. $\boldsymbol{s}_g$ is the relative position of the vehicle with respect to the goal. $\theta \in (-\pi, \pi)$ represents the angle in a clockwise direction between the agent's movement vector and the vector connecting the agent point to the target point. We summarize our introduced state information in detail in Table I.

The action that our algorithm applies to the vehicle is a 2-D continuous action vector, $a = (\omega, \phi)$, where ω is the throttle, and ϕ is the steering angle. The goal of the agent is to accomplish various P2P navigation tasks without colliding with the objects in the simulation environment.

We design a reward function for the agent that includes several components as follows:

$$r_t = r_t^s + r_t^c + r_t^{ms} + r_t^{sb} + r_t^d, \quad (2)$$

where r_t is the overall reward that the agent receives at each time step t. r_t^s is the reward component that indicates the agent successfully accomplished the task. r_t^c is the collision reward that is given to the agent when it collides with an object in the environment. To prevent the agent try to accomplish one task for a long period of time, we introduce r_t^{ms}, maximum time step reward. If the agent spends more than 500 time steps in one episode, we terminate the episode and penalize the agent by r_t^{ms}. If the agent outputs negative throttle, which translates into the break while having 0 velocity, we penalize the agent by r_t^{sb}, stationary brake reward, and terminate the episode to encourage the agent to explore the environment and improve the sample efficiency. Sparse reward settings may deteriorate the training process of the agent. Since the aforementioned reward components are sparse, we add one more reward component to prevent the sparsity of the reward signal. At each time step, t, we reward the agent with $(d_t - d_{t-1})$, where d_t and d_{t-1} represents the Euclidian distance between the position of the agent and the position of the goal at control step t and $t-1$. We define r_t^d, the distance reward, as $0.25(d_t - d_{t-1})$. We set r_t^s, r_t^c, r_t^{ms}, r_t^{sb} as 120, -60, -80, -80, respectively. While determining these values, we prioritize preventing the agent from exploiting the reward function by traveling across the virtual environment. Then, we fine-tune them after observing behaviors of the agents in several training processes.

State	Dimension	Feature Space
$\boldsymbol{s}_d^k$	(12x72x128)	$\boldsymbol{s}_d^k \in (0, 255)$
$\boldsymbol{s}_m^k$	(4x80x80)	$\boldsymbol{s}_m^k \in (0, 255)$
$\boldsymbol{s}_v$	(2)	$\boldsymbol{s}_{v_1} \in [-1.43, 1.43]^{\text{rad/s}}$, $\boldsymbol{s}_{v_2} \in [0, 10]^{\text{m/s}}$
$\boldsymbol{s}_a$	(2)	$\boldsymbol{s}_{a_1} \in [-0.5, 0.5]^{\text{rad/s}^2}$, $\boldsymbol{s}_{a_2} \in [-4, 2.5]^{\text{m/s}^2}$
$\boldsymbol{s}_g$	(1)	$\boldsymbol{s}_g \in [10, 50]^{\text{m}}$
$cos(\theta)$	(1)	$\theta \in (-\pi, \pi)$, $cos(\theta) \in (-1, 1)$
$sin(\theta)$	(1)	$\theta \in (-\pi, \pi)$, $sin(\theta) \in (-1, 1)$

TABLE I: Introduced State Information and Their Dimensions

Actor-Network Architecture			
Layer Name	Input Size	Output Size	Parameters
DepthMap-CNN Block	(Bx12x72x128)	(Bx2304)	Table IV
MiniMap-CNN Block	(Bx4x80x80)	(Bx288)	Table V
Fully-Connected 1	(Bx2599)	(Bx256)	FC-256, Leaky ReLU
Fully-Connected 2	(Bx256)	(Bx256)	FC-256, Leaky ReLU
Fully-Connected 3	(Bx256)	(Bx2)	FC-2, Tanh

TABLE II: Introduced Actor-Network Architecture

B. Training Procedure

Our algorithm starts training with an empty experience replay buffer. Conventional TD3 and DDPG algorithms fill the replay buffer by sampling actions from a Multivariate Gaussian distribution that covers the action space of the problem. In general, this procedure is a reasonable way to fill the buffer. However, on autonomous driving tasks, consecutive positive acceleration actions should be applied to the vehicle to make it non-stationary. Otherwise, the steering actions that are applied to the vehicle will not change the state of the vehicle if the vehicle is stationary. We observe that using conventional exploration strategies makes the vehicle stationary and deteriorates the learning process.

Critic-Network Architecture			
Layer Name	Input Size	Output Size	Parameters
DepthMap-CNN Block	(Bx12x72x128)	(Bx2304)	Table IV
MiniMap-CNN Block	(Bx4x80x80)	(Bx288)	Table V
Fully-Connected 1	(Bx2601)	(Bx256)	FC-256, Leaky ReLU
Fully-Connected 2	(Bx258)	(Bx256)	FC-256, Leaky ReLU
Fully-Connected 3	(Bx258)	(Bx1)	FC-1, No Activation

TABLE III: Proposed Critic-Network Architecture

DepthMap-CNN Block Network Architecture			
Layer Name	Input Size	Output Size	Parameters
conv1	(Bx12x72x128)	(Bx32x34x62)	5x5, stride=2, Leaky ReLU
conv2	(Bx32x34x62)	(Bx32x15x29)	5x5, stride=2, Leaky ReLU
conv3	(Bx32x15x29)	(Bx64x6x13)	5x5, stride=2, Leaky ReLU
conv4	(Bx64x6x13)	(Bx64x4x11)	3x3, stride=1, Leaky ReLU
conv5	(Bx64x4x11)	(Bx128x2x9)	3x3, stride=1, Leaky ReLU
flatten	(Bx128x2x9)	(Bx2304)	-

TABLE IV: Introduced DepthMap-CNN Architecture

MiniMap-CNN Block Network Architecture			
Layer Name	Input Size	Output Size	Parameters
conv1	(Bx4x80x80)	(Bx8x39x39)	4x4, stride=2, Leaky ReLU
conv2	(Bx8x39x39)	(Bx16x18x18)	4x4, stride=2, Leaky ReLU
conv3	(Bx16x18x18)	(Bx16x8x8)	3x3, stride=2, Leaky ReLU
conv4	(Bx16x8x8)	(Bx32x3x3)	3x3, stride=2, Leaky ReLU
flatten	(Bx32x3x3)	(Bx288)	-

TABLE V: Introduced MiniMap-CNN Architecture

Therefore, we introduce a heuristic exploration strategy to fill the replay buffer before updating the parameters of the networks, as follows:

$$\begin{aligned} a_a(t, t_e) &= 0.4 + 0.5cos(\pi \times 13 \times t/t_e), \\ a_s(t, t_e) &= cos(\pi \times 17 \times t/t_e), \end{aligned} \tag{3}$$

where a_a, and a_s represent acceleration and steering actions, respectively, t and t_e represent the current exploration time step and the total number of the exploration time steps, respectively. By introducing the aforementioned exploration strategy, we aim to force the agent to cover the action space of the given task and make it mostly non-stationary as much as possible.

In the first step of the training procedure, the Critic-Networks and the Critic-Target-Networks sample as many transitions as the batch size from the replay buffer. Then, the Critic-Target-Networks yield a target value for the Critic-Networks as follows:

$$y = \mathbb{E}_{(r,s')\sim\mathcal{D}}[r + \gamma \min_{i=1,2} C'_i(s', \psi(A'(s'; \phi')); \theta'_i)], \tag{4}$$

where y is the target value for the Critic-Networks training, D is the experiences that are collected by the agent, γ is the discount factor, C'_i is the ith Critic-Target-Network, s' is the next state, ψ is the Action Normalizer, A' is the Actor-Target-Network, ψ' is the Actor-Target-Network parameters and ϕ'_i is the ith Critic-Target-Network parameters. The conventional TD3 algorithm uses the Target Policy Smoothing Regularization. This component enables actions near each other to produce values close to each other in the same state. However, due to the nature of the autonomous driving task, even a minuscule amount of change in the action should have a significant effect on the expected return of the agent, especially in tasks such as passing through a narrow passage. Therefore, in our approach, we reduce the impact of that component by a factor of 10.

In the second step of the training procedure, we train the parameters of the Critic-Networks by using the value produced by the Critic-Target-Networks and Actor-Target-Network. We use the same transitions that were sampled at the first step of the training. We define the loss function for both Critic-Networks as follows:

$$L(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y - C(s, a; \theta_i)]^2, \tag{5}$$

where $L(\theta_i)$ is the loss function for the ith Critic-Network. We update the parameters of each of the Critic-Network by minimizing the loss function given in Eq. (5).

In the third step of the training procedure, we train the parameters of the Actor-Network. We use the same transitions that are sampled at the first step of the training. In our algorithm, the parameters of the Actor-Network are updated by using the parameters of the first Critic-Network. Therefore, the training process of Actor-Network relies mainly on the performance of the first Critic-Network. If the first Critic-Network cannot approximate the expected returns of the given state action pairs well enough, then the Actor-Network outputs values mainly from the extreme points of the action-space of the task. This indirectly affects the distribution of the samples in the replay buffer. The experience buffer may consist of transitions that the agent has collected by choosing the actions from the extreme points of the action space. Although this is a common case at the beginning of the training process, applying extreme points of action space to the vehicle results in sudden acceleration, emergency braking, and rotation around a center point continuously, which yields poor or no exploration of the state-action space. We highlight that we also restrict the action space that the Actor-Network outputs by adding the Tanh activation function as conventional deep RL algorithms such as DDPG and TD3 do. If the above problem occurs, the magnitude of the gradients can vanish due to the derivative of the Tanh activation function approaching zero. Then, it becomes hard to properly train the Actor-Network even with a perfectly trained Critic-Network. Thus, we propose a novel regularization technique for both deep RL and autonomous driving studies, Tanh Regularization, to remedy the aforementioned effect. We define the Tanh Regularization component as Λ as follows:

$$\Lambda = \begin{cases} tanh^{-1}(A(s;\phi)) & tanh^{-1}(A(s;\phi)) \geq \kappa \\ 0 & \text{otherwise}, \end{cases} \tag{6}$$

where $tanh^{-1}$ is the inverse Tanh function, ϕ is the parameters of the Actor-Network, and κ is the cut-off value that activates the Tanh Regularization. the Tanh Regularization scales down the solution space of the problem and increases the magnitudes of the gradients if the agent chooses an action that lies at the extreme points of the action space. We calculate the objective function by adding the Λ to the objective function of the Actor-Network:

$$J(\theta_1, \phi) = \mathbb{E}_{s\sim\mathcal{D}}[C_i(s, a; \theta_1)|_{a=A(s;\phi)}\psi(A(s;\phi))] + \Lambda, \tag{7}$$

where $J(\theta_1, \phi)$ is the objective function, and ψ is the Action Normalizer. We update the parameters of each of the Critic-Network by maximizing the loss function in Eq. (7). If the Tanh Regularization successfully constrains the input values of the Tanh activation function, the Actor-Network cannot yield actions that fully cover the action space of the given task. To remedy this issue, the Action Normalizer casts the output of the Actor to the action space of the problem concerning the cut off value, κ, as follows:

$$\psi(A(s;\phi)) = A(s;\phi)/tanh(\kappa), \tag{8}$$

We also use the Action Normalizer when testing navTD3 since the Actor-Network would not yield values that fully cover the action space of the task due to Tanh Regularization.

In the fourth step of the training procedure, we train the parameters of the Critic-Target-Networks and Actor-Target-Network. The training process of these networks does not require any transitions. We use these networks to avoid the overestimation problem. We softly update the parameters of the Target-Networks using the primary Critic-Networks and Actor-Network as follows:

$$\phi' = \tau\phi + (1-\tau)\phi', \quad \theta_i' = \tau\theta_i + (1-\tau)\theta_i', \qquad (9)$$

where τ is the rate of the soft update.

We summarize our training procedure in Algorithm 1. We provide implementation of the crucial parts of our algorithm and performance of navTD3 in test seeds at https://github.com/doganjr/navTD3.

Algorithm 1 navTD3

Initialize Actor-Network $A(\phi)$
Initialize Critic-Networks $C_1(\theta_1)$, $C_2(\theta_2)$
Initialize Target-Networks $A'(\phi') \leftarrow A(\phi)$,
$C_1'(\theta_1') \leftarrow C_1(\theta_1)$, $C_2'(\theta_2') \leftarrow C_2(\theta_2)$
Initialize batch size b, replay buffer $\mathcal{D}$
Initialize M for delayed policy updates
for $t = 1$ **to** T **do**
  Observe s_t Choose action $a_t \sim \psi(A(a|s_t)) + \epsilon$
  Observe reward r_t and next state s_t'
  Store transition (s_t, a_t, r_t, s_t') in $\mathcal{D}$
  Sample batch of transitions B_i from $\mathcal{D}$
  Compute target value for Critic Training Eq. (1)
  Update the weights of the Critic-Networks by minimizing Eq. (2)
  if k mod M **then**
    Update the weights of the Actor-Network by maximizing Eq. (4)
    Softly update the weights of the Actor-Network
  end if
  Update the weights of the Target networks Eq. (5)
end for

C. Implementation Details

To make a fair comparison, for all algorithms, each algorithm is implemented with identical hyper-parameters. Both policy and value networks employ the same convolutional neural network (CNN) models as feature extractors, followed by multilayer perceptron (MLP) with two hidden layers (256, 256). Table II, III, IV, V show proposed Actor-Network and Critic-Network architectures in detail. The input for DepthMap-CNN Blocks is the last four 360° depth map scans (72x384), and the input for the MiniMap-CNN Blocks is the last four cropped floor sketches (80x80) generated in run-time in line with agent position and orientation. Figure 2 shows examples of depth map scans and cropped floor sketches. All deep neural network (DNN) models are optimized using Adam optimizer [27]. The learning rates are 3×10^{-4} for both the Actor and the Critic-Networks of navTD3 and TD3 algorithms. For DDPG, learning rates for the Actor-Network and the Critic-Network is 1×10^{-4} and 3×10^{-4}, respectively. We adopt the mini-batch size of 64 for DDPG and 128 for TD3 and navTD3. The experience replay size is set to 10^6 for all algorithms, and exploration timesteps are set to $25,000$ for all algorithms. Evaluations are performed using the mean cumulative rewards of the last 100 episodes during the training.

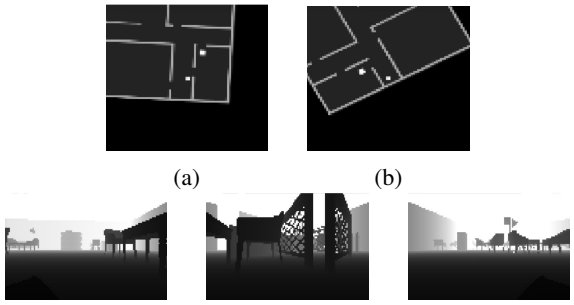


Fig. 2: Random instances of inputs to MiniMap-CNN blocks (a), (b) and 360° depth map scan as input to DepthMap-CNN blocks (c) left, (d) center, (e) right.

IV. Experiments

A. Learning Environment

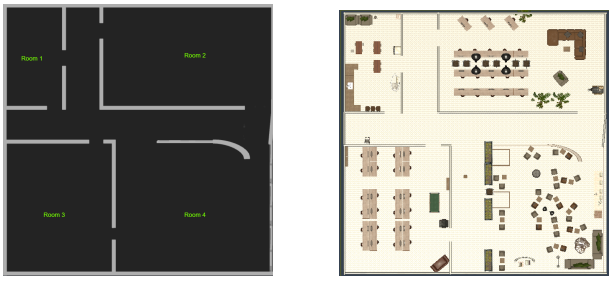
In general, training for an autonomous driving task in a real-world scenario is risky, costly, complicated, time-consuming, and usually not reproducible [28], [29], [11]. The best choices would involve professional solutions like CarSIM [30], CarMaker (by IPG Automotive), and PaTAVTT [31]. However, the utilization of these solutions is not typical among autonomous driving researchers since they generally do not provide user-friendly Python interfaces. To achieve the near-realistic, high-fidelity simulation environment for autonomous navigation tasks while ensuring the speed, memory, and determinism constraints, we adopt AirSim by Microsoft Research [32]. It is an open-source, cross-platform simulator built on Unreal Engine 4 for AI research on drones and vehicles. Besides being modern, robust, fast, and highly scriptable, AirSim allows interfacing with detailed sensor designs to achieve practically realistic simulations and noise models of external hardware [28], [32].

We evaluate our algorithm in a self-made office-like working space environment to perform complete control over both the world and vehicle model on the simulation. Figure 3 shows the floor plan of the designed map with and without static objects. Our office-like environment consists of four rooms connected through the main hallway. This hallway acts as a gateway for the entrances of rooms. The map and the interior are constructed so that a 1:10 scaled mini-SUV model can handle a complete P2P navigation objective while facing complex navigation challenges such as driving through narrow passages, switching rooms by cruising through corridors, favoring to plan the shortest-path routes, etc. Since the whole map plan is unknown to the agent, room plans and static objects are arranged so that

Sensor Models	
Stereolabs Zed 2	
Depth Camera Resolution	320x240
Lens Focal Length	2.12 mm
Field of View	120°(H) x 70°(V)
Lens Aperture	f/1.8
TV Distortion	5.07%
Accelerometer	
Range	+/- 8G
Resolution	0.244 mg
Noise Density	3.2 mg
Gyroscope	
Range	+/- 1000 dps
Resolution	0.03 dps
Noise Density	0.16 dps
Sensitivity Error	+/- 0.4%

TABLE VI: The details of sensor models in the self-made simulation environment

it contains several different navigation tasks which do not require any mapping or planning module. Figure 1 shows random instances from our training environment. Moreover, the diversity of objects in the interior (distinct types of furniture, tables, chairs, desks, couches, and office equipment) allows CNN feature extractors to learn better representations for divergent autonomous driving circumstances.



(a) (b)

Fig. 3: Floor plan of designed map in Unreal Engine 4: (a) without static objects, (b) with static objects

Our car model is a 1:10 scaled all-wheel drive SUV that adopts all dynamics of the SUV model proposed by AirSim [32]. It employs three depth cameras, one inertial measurement unit (IMU), and a GPS sensor. Using raw sensor data produced by simulations like AirSim grants the advantage of improved contextual information by reducing the state space complexity since they utilize unrealistic, near-optimal sensor models by default. To realize the sensor noises, we exploit both the IMU noise model and depth camera noise model from Stereolabs ZED 2 Camera. The details of sensor models in the simulation are given in Table VI and Figure 2 shows examples of horizontally concatenated three depth cameras' observation that provides 360° field of view for the agent.

B. Experimental Configuration

We use AirSim v1.6.0-windows and Unreal Engine 4.24.2 to create the office-like simulation environment. The model is implemented in Python using PyTorch and trained on a computer with NVIDIA GeForce RTX 3060 12GB GPU and Intel Core i9-10900K 3.7GHz 16-core CPU. We choose the operation frequency of our model as 4 Hz to redress the balance of training time and situational awareness of the vehicle.

V. Results

A. Evaluation

We evaluate our algorithm by using 40 different P2P navigation scenarios. These tasks are symmetric to each other. For each task, the start and goal positions are swapped to obtain their corresponding task. In both cases, the agent starts with the same orientation. This methodology makes a fair comparison of the algorithms by providing a difficulty balance over the tasks. For example, if one task just requires straight driving to accomplish, then its symmetric task requires a complete U-turn. Also, there are a small number of trivial tasks to enable the agent quickly explore the presence of a significantly big reward upon completing the task. The distance between start and goal positions of each task that our algorithm learns through during the training is between 20 and 40 meters.

We use another 40 P2P navigation scenarios to measure the generalization ability of our algorithm. None of these tasks belong to training scenarios. These scenarios do not include trivial scenarios such as driving straight to reach the target. We test the generalization ability of our algorithm by the P2P navigation tasks that relatively require a long-term strategy. Distances between start and goal points for algorithm evaluation range between 20 and 80 meters.

B. Values Before Tanh

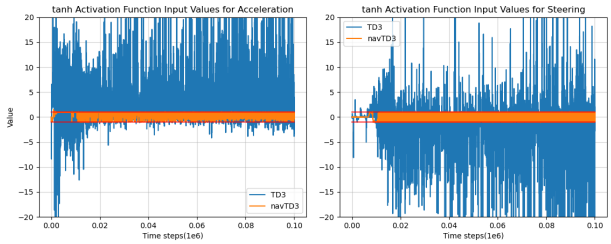


Fig. 4: The Actor-Network outputs before the Tanh activation function. Orange and Blue curves represent values yielded by TD3 and navTD3, respectively. The Red line represents the cut-off value used for Tanh Regularization.

We remark that both TD3 and DDPG use the Tanh activation function in their original implementations at the last layer of the Actor-Network. The purpose of this procedure is to clamp the output of the Actor and ensure the Actor-Network yields a valid policy for the action space of the given task. We claim that the very high input values of the Tanh activation function of the Actor deteriorate the learning process by lowering the gradient magnitudes of the Actor-Network when the Actor-Network yields actions from the endpoints of the action space of the given problem. To verify our claim, we keep track of the inputs of the Tanh

activation function 100,000 time steps after the exploration time steps for navTD3 and TD3. In Figure 3 blue and orange lines represent Tanh activation function input values of the last layer of the Actor-Network for TD3 and navTD3, respectively, and red lines represent the cutoff value, κ, which we set to 1 to restrict inputs of the Tanh activation function between 1 and -1, that Tanh Regularization uses. Figure 3 shows that the input values that the Tanh activation function receives are significantly high for the TD3 algorithm. On the other hand, navTD3 successfully bounds the inputs of the Tanh activation function of the last layer of the Actor-Network, and prevents the agent from unnecessarily selecting the actions from the extreme points of the action space. To inspect the effect of the Tanh Regularization, we compare the cumulative rewards collected by navTD3, TD3, and DDPG at the beginning of the training. Figure 4 shows the performance of navTD3 drastically increases in terms of cumulative reward and significantly outperforms conventional TD3 for the first 100,000 time steps.

C. Performance for Training Tasks

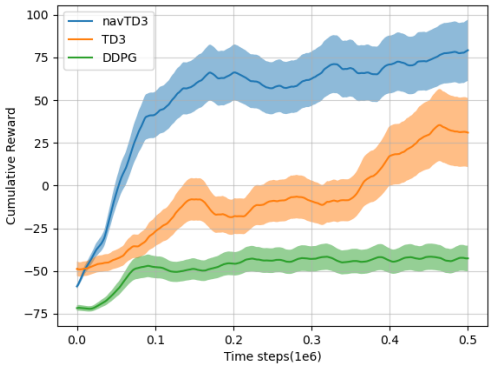


Fig. 5: Performance comparisons of algorithms. Blue, Orange, and Green curves represent the average cumulative rewards collected by the agents over ten trials for navTD3, TD3, and DDPG, respectively. The shaded regions represent half standard deviation.

We train ten agents for each algorithm, navTD3, TD3, and DDPG for a fair comparison. Figure 4 shows averages of cumulative rewards that each algorithm yields and shaded regions represent the standard deviation. At the beginning of the training, TD3 agents adopt a policy that encourages moving straight. Therefore, TD3 agents start training by accomplishing relatively easier P2P tasks. The performance of navTD3 agents sharply increases in the first stages of the training with respect to agents that we couple with TD3 and DDPG. We associate this effect mainly with our proposed Tanh regularization technique and exploration strategy. Our algorithm, navTD3, outperforms TD3 and DDPG in terms of cumulative rewards collected by the agents during the training.

We calculate the success percentage of the agents by dividing the number of accomplished P2P tasks by the number of available P2P tasks. Table VII shows the success percentages of the agents on tasks that we expect them to complete during training. The overall performance of navTD3 agents outperforms the best-performing TD3 and DDPG agents. As a qualitative analysis of the best performing navTD3 agent, we observe that the agents trained with navTD3 move with slaloms when they reach a high velocity. We associate this behavior with trying to keep the velocity at the same level by making slaloms rather than using the brake. Also, navTD3 agents mainly struggle with scenarios that start with a goal position that is not in sight of the agent. These types of scenarios are designed to be solved by many consecutive optimal actions. Therefore, it is hard to find even a suboptimal solution for these scenarios if a critical mistake has been made by the agent.

Success Rates of the Agents			
Method Name	**$20-40$m Training Tasks**	**$20-40$m Test Tasks**	**$40-80$m Test Tasks**
navTD3	0.803 ± 0.031	0.592 ± 0.028	0.451 ± 0.023
TD3	0.534 ± 0.043	0.423 ± 0.035	0.319 ± 0.019
DDPG	0.294 ± 0.013	0.129 ± 0.011	0.038 ± 0.008

TABLE VII: Success rate comparison for each algorithm

D. Performance for Test Tasks

We divide the test tasks into two groups according to the distance between the starting and goal positions. Short-distance test tasks share the same distance difference with the training tasks, and distances for the long-distance test tasks are between 40 meters and 80 meters. Table 5 represents success rates on 20-40m training tasks and 20-40m test tasks. Also, it shows that the success rate of the navTD3 on training tasks decreases less when trained agents are evaluated on test tasks with the same distances when compared to the other algorithms. Therefore, navTD3 provides more generalization ability in comparison to the other algorithms. Long-distance test tasks require more long-term planning rather than shorter tasks. Moreover, most of these tasks also require navigation between different rooms of the simulation environment. Table VII represents success rates on 40-80m test tasks of the agents, and it states that navTD3 performs better than the other algorithms on long-distance test tasks. Even though the performance drop is higher for the long-distance test tasks, our algorithm provides a worthwhile performance without utilizing any planning algorithm. We also observe that in overall test tasks, all algorithms tend to find suboptimal routes for the accomplished test tasks when compared to the accomplished training tasks.

VI. Conclusion & Future Work

In this paper, we present an end-to-end deep RL algorithm that works on P2P navigation tasks. Our algorithm, navTD3, works on continuous action and state spaces, and

it provides a policy for acceleration and steering control for autonomous vehicles. The navTD3 mainly utilizes one of the state-of-the-art deep RL algorithms, TD3, and remedies its main drawbacks on autonomous driving tasks by utilizing a different exploration technique and a novel method which we propose, the Tanh Regularization.

We design a realistic simulation environment to assess the performance of navTD3 and compare it with previous approaches such as TD3 and DDPG. Besides comparing algorithms on the tasks that we provide during training, we test algorithms on a large number of test tasks that differ from the tasks during the training. We inspect test tasks into two groups, short-distance, and long-distance tasks, to compare the generalization ability of the algorithms with different aspects. The results show that navTD3 significantly outperforms TD3 and DDPG in terms of final performance and generalization ability.

In our future work, we aim to bring more generalization ability to our proposed algorithm and evaluate it in a real-world environment rather than a simulation environment. Also, navTD3 can be applied to other robot tasks, such as drone and plane navigation, with proper action and state space formulation.

VII. Acknowledgment

Dogan C. Cicek and Furkan B. Mutlu were supported by Türk Telekom and Vodafone within the framework of 5G and Beyond Joint Graduate Support Programme coordinated by Information and Communication Technologies Authority.

References

[1] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[4] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. G. Francis, J. Davidson, and L. Tapia, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5113–5120, 2018.

[5] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[6] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029, 2019.

[7] S. Han, H. Wang, S. Su, Y. Shi, and F. Miao, "Stable and efficient shapley value-based reward reallocation for multi-agent reinforcement learning of autonomous vehicles," in *ICRA*, 2022.

[8] B. Li, N. Ammar, P. Tiwari, and H. Peng, "Decentralized ride-sharing of shared autonomous vehicles using graph neural network-based reinforcement learning," *2022 International Conference on Robotics and Automation (ICRA)*, pp. 912–918, 2022.

[9] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[11] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5671–5677, 2020.

[12] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning robust control policies for end-to-end autonomous driving from data-driven simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, 2020.

[13] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. V. Gool, "End-to-end urban driving by imitating a reinforcement learning coach," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15202–15212, 2021.

[14] T. Glasmachers, "Limits of end-to-end learning," in *ACML*, 2017.

[15] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *NIPS*, 1988.

[16] F. Codevilla, M. Müller, A. Dosovitskiy, A. M. López, and V. Koltun, "End-to-end driving via conditional imitation learning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, 2018.

[17] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," *CoRR*, vol. abs/1805.01954, 2018.

[18] K. Chitta, A. Prakash, and A. Geiger, "Neat: Neural attention fields for end-to-end autonomous driving," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15773–15783, 2021.

[19] Y. Oh, K. Lee, J. Shin, E. Yang, and S. J. Hwang, "Learning to sample with local and global contexts in experience replay buffer," *arXiv preprint arXiv:2007.07358*, 2020.

[20] J. Zhang and E. Ohn-Bar, "Learning by watching," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12706–12716, 2021.

[21] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," *CoRR*, vol. abs/1912.12294, 2019.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[23] 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, pp. 484–489, jan 2016.

[24] 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.

[25] A. Francis, A. Faust, H.-T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T.-W. E. Lee, "Long-range indoor navigation with prm-rl," *IEEE Transactions on Robotics*, vol. 36, pp. 1115–1134, 2020.

[26] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, pp. 4298–4305, 2019.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[28] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, 2019.

[29] H. Hu, K. Zhang, A. H. Tan, M. Ruan, C. Agia, and G. Nejat, "A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6569–6576, 2021.

[30] R. Johansson, D. Williams, and P. Nugues, "Carsim: A system to convert written accident reports into animated 3d scenes," in *Proceedings of the 2nd Joint SAIS/SSLS Workshop Artificial Intelligence and Learning Systems, AILS-04*, (Lund, Sweden), pp. 76–86, April 15-16 2004.

[31] Z. Xu, M. Wang, F. Zhang, S. Jin, J. Zhang, and X. Zhao, "Patavtt: A hardware-in-the-loop scaled platform for testing autonomous vehicle trajectory tracking," *Journal of Advanced Transportation*, vol. 2017, pp. 1–11, 11 2017.

[32] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.