2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

NaviFormer: A Data-Driven Robot Navigation Approach via Sequence Modeling and Path Planning with Safety Verification

Xuyang Zhang, Ziyang Feng, Quecheng Qiu, Yu'an Chen, Bei Hua, and Jianmin Ji

Abstract— **Reinforcement learning has shown great potential in improving the performance of robot navigation. In response to the increasing deployments of mobile robots within various scenarios, a data-driven paradigm of navigation approach with safety verification is preferred where one can train RL algorithms with large amounts of prior data, keep learning continuously, and ensure safe navigation in applications. Conventional end-to-end reinforcement learning navigation paradigms have encountered multiple challenges in meeting these demands. In this work, we introduce a novel robot navigation approach termed NaviFormer. This approach handles navigation tasks based on sequence modeling to obtain the data-driven ability. It also integrates rule-based verification for safety insurance. We conduct a series of experiments to validate the data-driven ability of our approach and to compare it with existing navigation methods. We also perform quantitative tests on a real-world robot platform, TurtleBot. The experimental results show our method's outstanding data-driven ability and highlight its superior arrival rate and generalization compared to other state-of-the-art methods like the PPO-based navigation method.**

I. Introduction

Reinforcement learning (RL) has captured significant attention within the robotics field for its potential to address challenges in robot navigation [1]. One prevalent method in RL is end-to-end online reinforcement learning [2]–[4], where a single network that maps the robot's observations to corresponding motion commands is trained through online trial-and-error, often in interaction with a simulation environment. Notably, it has demonstrated remarkable performance by employing advanced online reinforcement learning algorithms such as asynchronous advantage actor-critic (A3C) [5] and proximal policy optimization (PPO) [6].

Nevertheless, this end-to-end online reinforcement learning framework has inherent limitations. Firstly, the direct output of low-level motion commands from the network leads to less interpretable robot behavior and makes enforcing safety constraints challenging [7]. This framework can be considered "fragile" and might compromise safety in complex navigation environments. Moreover, the exceptional performance of this framework relies on sophisticated online reinforcement learning algorithms, resulting in difficulties in continuous performance enhancement. This framework leads to high maintenance costs and limited scalability for real-world deployment. Thus, the enhancement of a navigation framework capable of safety verification and easy performance improvement becomes imperative for broader robot navigation applications.

On the other hand, navigation submodules have made a lot of progress. Local control algorithms such as the dynamic window approach (DWA) [8] and trajectory elastic band (TEB) [9] offer efficient control solutions. Numerous robot navigation datasets [10]–[12] furnish diverse real-world expert navigation data, proven to be more efficient for training compared to online trial-and-error [13]. Making full use of these navigation submodules is beneficial to the performance of the navigation framework.

To address these challenges, we introduce a novel offline reinforcement learning robot navigation approach, NaviFormer[1]. We gather an offline dataset and leverage the decision transformer (DT) [14], an advanced offline reinforcement learning approach, to model navigation tasks as sequences and train a transformer-based network to predict the robot's future path for a predefined duration. The predicted future paths allow for straightforward safety verification and precise motion control.

Our approach makes the network focus on planning by entrusting perception and control tasks to other navigation submodules, which not only enhances safety but also reduces the difficulty of network training. The utilization of sequence modeling equips the network to handle long-term perception [14]. The offline reinforcement learning foundation allows it to effectively learn policies from only offline datasets. Importantly, our approach exhibits a data-driven capacity for performance improvement, i.e. the quality enhancement in the offline dataset translates to improved navigation performance. It also displays a level of resistance to dataset noise, which is helpful for real-world deployment. Our primary contributions are summarized as follows:

- We present NaviFormer, a robot navigation approach based on offline reinforcement learning, encompassing sequence modeling and safety verification.
- We showcase NaviFormer's ability to improve performance through data-driven means. This improvement is achieved with reduced sensitivity to dataset noise compared to imitation learning.
- We evaluate NaviFormer's performance across a series of experiments in both simulated and real-world environments. The results underline its superiority concerning performance and generalization.

{zxy2533, fzy617, qiuqc, an11099}@mail.ustc.edu.cn, {bhua, jianmin}@ustc.edu.cn. School of Computer Science and Technology, University of Science and Technology of China. Corresponding author: Jianmin Ji.

The work is partially supported by Guangdong Province R&D Program 2020B0909050001, Anhui Province Development and Reform Commission 2021 New Energy and Intelligent Connected Vehicle Innovation Project.

[1]https://github.com/DRL-Navigation/NaviFormer

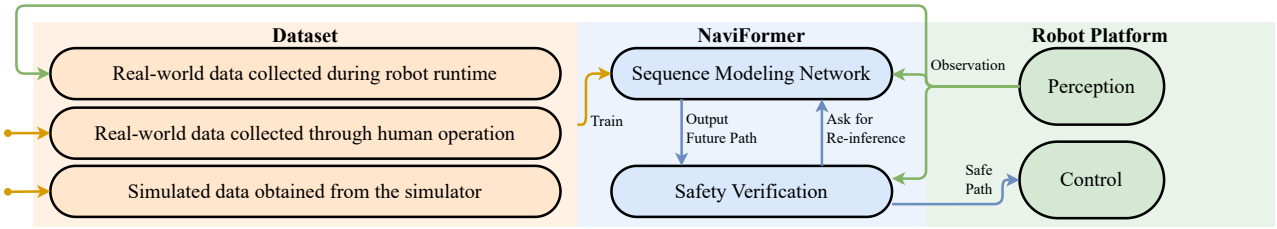


Fig. 1. Pipeline of the NaviFormer approach. Offline datasets can be collected through diverse methods, enabling network training without manual annotations. After the training process, the network can generate the predicted future path based on real-time observations. The safety verification algorithm checks the path and can iteratively call the network until a safe path is determined. The robot's local control algorithm takes charge of motion control upon acquiring a verified safe path, thereby accomplishing successful navigation.

II. Related Work

A. Robot Navigation

The task of robot navigation, involving safe and efficient movement to the goal point, has been extensively studied. Classical hierarchical planning and online reinforcement learning [7] are two kinds of approaches that are commonly applied for robot navigation.

On the one hand, global planning algorithms like A* search [15] and probabilistic road maps (PRM) [16] can be coupled with local control algorithms such as dynamic window approach [8] and trajectory elastic band [9] to achieve precise navigation, though this approach still faces challenges in complex environments [7].

On the other hand, online reinforcement learning approaches hold substantial potential. Tai et al. [17] introduced a sensor-level reinforcement learning network based on deep deterministic policy gradients [18]. Building on this, Zeng et al. [19] successfully applied asynchronous advantage actor-critic [5] to robot navigation. Yao et al. [20] further enhanced the robot's ability to evade dynamic obstacles using proximal policy optimization [6]. However, challenges persist in the domain of online reinforcement learning, including issues of interpretability and optimal data utilization [7].

B. Offline Reinforcement Learning

The main difference between offline and online reinforcement learning is that offline one cannot interact with the environment for trial-and-error during the learning process, but can only learn from pre-collected datasets. This framework possesses several advantages, including a more generalized policy and a higher data utilization rate, while it is harder to be conducted. Moreover, offline reinforcement learning is considered a data-driven paradigm [21].

In the context of the Markov decision process (MDP), which comprises components $(\mathbf{S}, \mathbf{A}, P, R)$, $s_t \in \mathbf{S}$ denotes states, $a_t \in \mathbf{A}$ signifies actions, $P(s_{t+1} \mid s_t, a_t)$ represents state transition probabilities, and $r_t = R(s_t, a_t)$ denotes rewards. The experience sequence is denoted as $\tau = (s_0, a_0, r_0, \ldots, s_T, a_T, r_T)$. The objective of most offline reinforcement learning methods is to find an optimal policy $\pi^*(a_t|s_t)$ from an offline dataset $\mathbf{D}$ to maximize the cumulative rewards $\mathbb{E}_{\tau\sim P,\pi^*}[\sum r_i]$. Among the various offline reinforcement learning methods, decision transformer [14] emerges as a promising and recent approach that has yielded impressive results. They introduce a novel sequence $\tau_{DT} = (\widehat{R}_0, s_0, a_0, \ldots, \widehat{R}_T, s_T, a_T)$, where the return-to-go (RTG) $\widehat{R}_t = \sum_{i=t}^{T} r_i$ signifies the sum of all rewards attainable starting from s_t. The objective of the decision transformer is to model the sequence's conditional distribution $P(\tau_{DT} \mid \widehat{R}_0, s_0, a_0, \ldots, \widehat{R}_t, s_t)$ within the offline dataset $\mathbf{D}$ to predict a_t and complete the Markov decision process.

Inspired by the decision transformer, we extend the application of sequence modeling to robot navigation in this paper, enhancing the experience sequence and integrating navigation submodules to improve overall performance.

III. Approach

In this section, we present an in-depth overview of the NaviFormer robot navigation approach. Our approach's pipeline is depicted in Fig. 1.

A. Dataset

Given the minimal impact of noise data on NaviFormer, as expounded in Sec. IV-B, there is no need to filter the collected raw data. All we need to do with the raw data are two simple steps: computing future paths and annotating uniform rewards.

Firstly, the computation of future paths is based on the sequence of states within the raw data. The positional information of sequence $(s_t, s_{t+1}, \ldots, s_{t+l})$ inherently delineates the robot's path from time t to $t+l$. We denote this as $p_{t:t+l} = (c_t, c_{t+1}, \ldots, c_{t+l})$, where c_t symbolizes the robot's global coordinates and orientation. Then, we need a discretization step for network training. For each c_t in $p_{t:t+l}$, we calculate relative polar coordinates and the orientation change concerning c_{t-1}. And even discretizations are applied to these polar coordinates and orientation changes. Importantly, due to the limited displacement of the robot within one timestep, the values obtained through this discretization method are notably fewer than those that would result from directly discretizing c_t. The outcome of this discretization step is denoted as the discretized future path, represented as $p^d_{t:t+l}$.

The next step is to annotate uniform rewards for the raw data. Eq. (1) elucidates our reward function, where $\alpha_{\text{collision}}$, $\beta_{\text{collision}}$, α_{reach}, α_{closer}, and α_{further} are constants. $MinDistance2Obs(s_t)$ returns the minimum distance between the robot and all obstacles in timestep t, while

MinDistance2Goal(τ) returns the minimum distance between the robot and the goal point within the sequence τ. This reward function has four components. $r_{\text{collision}}$ and r_{reach}, respectively, serve to penalize collisions and reward goal reach. Furthermore, r_{closer} assumes the role of a dense reward, instilling motivation for the robot to get closer to the goal point. Lastly, r_{further} serves as a sparse reward, discouraging deviations from the goal while alleviating the local minima issues [22].

$$
\begin{aligned}
&r_t = r_{\text{collision},t} + r_{\text{reach},t} + r_{\text{closer},t} + r_{\text{further},t} \\
&r_{\text{collision},t} = \begin{cases} -\alpha_{\text{collision}}, & \text{If a collision occurs,} \\ -\beta_{\text{collision}} \times (0.5 - b_t)^2, & \text{If } b_t < 0.5, \\ 0, & \text{Otherwise.} \end{cases} \\
&r_{\text{reach},t} = \begin{cases} \alpha_{\text{reach}}, & \text{If reaches the goal,} \\ 0, & \text{Otherwise.} \end{cases} \\
&r_{\text{closer},t} = \begin{cases} \alpha_{\text{closer}} \times (d_{t-1} - d_t)^2, & \text{If } d_t < d_{t-1}, \\ 0, & \text{Otherwise.} \end{cases} \\
&r_{\text{further},t} = \begin{cases} -\alpha_{\text{further}}, & \text{If } d_t >= d_{t-1} \text{ and } d_{t-1} < d_{t-2}, \\ 0, & \text{Otherwise.} \end{cases} \\
&b_t = \mathit{MinDistance2Obs}(s_t) \\
&d_t = \mathit{MinDistance2Goal}(\tau[:t])
\end{aligned}
\tag{1}
$$

Upon the computation of paths and the annotation of rewards for the raw data, we acquire the offline dataset formulated as $\tau_{\mathbf{D}} = (s_0, a_0, r_0, p^d_{0:0+l}, \ldots, s_T, a_T, r_T, p^d_{T:T+l})$, where T is the length of the experience sequence and may be unequal in different experience sequences.

B. NaviFormer

The NaviFormer approach is comprised of two primary components: a sequence modeling path planning network and a rule-based safety verification algorithm. This novel combination makes use of the strengths of both offline reinforcement learning and classical algorithms, ensuring not only optimal navigation paths but also safety assurance.

1) Sequence Modeling Network: Enhancing the decision transformer network, we have extended its capabilities to handle navigation planning tasks and generate future paths as its outputs. The detailed architecture of our sequence modeling network is illustrated in Fig. 2. Our modifications begin by transforming the experience sequence τ_{DT} of the decision transformer. In this context, we substitute the action a_t with the future path $p^d_{t:t+l}$, which is treated as a sequence of l consecutive tokens to emphasize the temporal evolution inherent in the path. In addition, we partition state s_t into multiple tokens, as it often encapsulates several distinct observations of significance in navigation tasks. The method for computing return-to-go $\widehat{R}_t$ is also adapted: $\widehat{R^{'}}_t = \sum_{i=t}^{t+l} r_i$, which omits rewards beyond the future path, creating a more accurate depiction of the future path's potential rewards. This updated experience sequence for NaviFormer is denoted as $\tau_{NF} = (\widehat{R^{'}}0,) + s_0 + p^d_{0:l} + \cdots + (\widehat{R^{'}}_T,) + s_T + p^d_{T:T+l}$, where we use the plus symbol $(+)$ to denote the concatenation of two sequences. Subsequently, each token in τ_{NF} is fed into its respective encoder for encoding into the same dimension. Positional information is added by directly using the index of tokens as positional encodings. The token sequence is then modeled autoregressively through several layers of the generative pre-training (GPT) structure [23]. The part of GPT's output token sequence representing the path is extracted and passed through a linear decoder layer with a softmax function to produce a discrete probability distribution of the path.

In the training phase, we construct τ_{NF} by extracting data from the offline dataset. Additionally, we introduce Gaussian noise to each $\widehat{R^{'}}_t$ to counter overfitting. The network is trained using cross-entropy loss, with the loss calculation focusing solely on path prediction, excluding other tokens in τ_{NF}. Given that a path is encoded into l tokens, its cross-entropy loss also has l components. Recognizing the greater influence of points closer to the path's beginning, we apply a weighted approach: $\mathit{Loss}(p^d_{t:t+l}) = \sum_{i=0}^{l} \eta^i \times \mathit{Loss}(p^d_{t:t+l}[i])$, where $0 < \eta < 1$ is a constant, and $\mathit{Loss}(x)$ signifies the cross-entropy of token x.

In the inference phase, we maintain a history experience sequence $\tau_{NF}[0:t-1]$. At timestep t, we provide a rough estimate of $\widehat{R^{'}}_t$ based on the real-time observation s_t. Due to the Gaussian noise added to return-to-go during training, we do not need to know the precise value of $\widehat{R^{'}}_t$. We can intentionally overestimate $\widehat{R^{'}}_t$ to potentially yield better paths. Next, $\widehat{R^{'}}_t$ and s_t are concatenated with $\tau_{NF}[0:t-1]$ as the input of the network, resulting in the predicted future path $p^d_{t:t+l}$. Then, $p^d_{t:t+l}$ is concatenated with the input, updating to $\tau_{NF}[0:t]$ for timestep $t+1$.

2) Safety Verification: Before delving into the presentation of the safety verification algorithm, let us first elucidate the network inference process. When the network receives the input sequence $\tau_{\text{input}} = \tau_{NF}[0:t-1] + (\widehat{R^{'}}_t,) + s_t$, it proceeds to yield an output sequence of equal length but shifted by one position. The final element in the output sequence represents the discrete distribution of the first point in the future path, i.e., $\tau_{\text{output}}[-1] = P(p^d_{t:t+l}[0] \mid \tau_{\text{input}})$. Typically, we select the maximum value from this discrete distribution as the predicted outcome, i.e., $p^d_{t:t+l}[0] = \arg\max(\tau_{\text{output}}[-1])$. Subsequently, we update the input sequence $\tau_{\text{input}} = \tau_{\text{input}} + (p^d_{t:t+l}[0],)$, which is then fed back into the network. In this turn, the last element of the output sequence captures the discrete distribution of the second point in the future path, leading to the prediction of the second point. This iterative process continues until the complete future path is generated.

With the incorporation of the safety verification algorithm, the network's inference process is altered as follows: after obtaining the discrete distribution $P(p^d_{t:t+l}[i] \mid \tau_{\text{input}})$, the network no longer directly selects the maximum value from the distribution as the predicted outcome. Instead, it collaborates with the top$k$ vector from the safety verification algorithm, where top$k[i]$ represents the rank of the distribution that should serve as the prediction for $p^d_{t:t+l}[i]$. This strategy empowers the safety verification algorithm to gather a range of distinct paths from the network, giving priority to paths deemed more favorable by the network. Consequently, the network can be iteratively called until a path that meets the safety constraints is found.

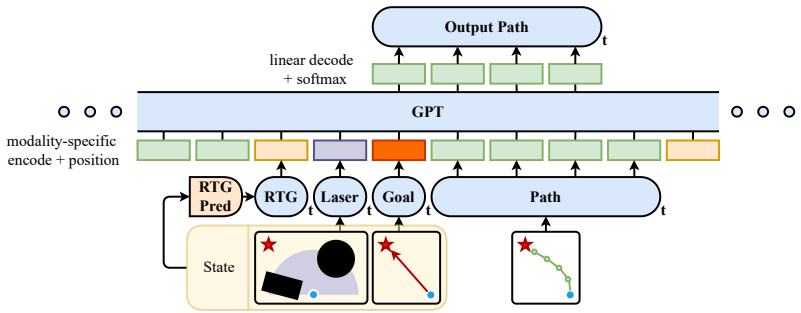
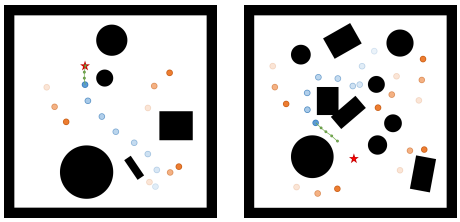


Fig. 2. Architecture of the sequence modeling network. The return-to-go, the state, and the future path are separated and encoded into tokens through modality-specific encoding with the addition of positional information. The tokens are then input into the GPT architecture, which uses the causal self-attention to predict the future path autoregressively.



(a) Sim(5, 3) (b) Sim(10,5)

Fig. 3. Simulation environments. The black area represents obstacles, the orange dots represent the pedestrians and their recent history paths, the blue dots represent the running robot and its history path, the red star represents the goal point of the robot, and the green dots and lines represent the predicted future path from NaviFormer.

C. Details of Our Implementation

Due to space limitations, we specify details of all parameter configurations for the dataset and network, as well as the pseudo-code of the return-to-go prediction algorithm and safety verification algorithm, in our appendix on GitHub.

IV. Experiments

In this section, we conduct a series of quantitative comparisons against prevailing navigation methods in various environments, including some simulation environments, a common indoor navigation benchmark, and five real-world scenarios. Our insights are to provide the following inquiries:

- Does NaviFormer exhibit superior data-driven ability compared to imitation learning?
- Does the sequence modeling network of NaviFormer return efficacious predictions of future paths?
- Does the performance and generalization exhibited by NaviFormer surpass that of other state-of-the-art navigation methods?

A. Setup

We use Img-Env[2], an open-source 2D navigation simulator, to build simulation environments by randomly placing obstacles and pedestrians on a $10m \times 10m$ open space, as depicted in Fig. 3. Obstacle positions are randomized and may overlap, while pedestrians follow the reciprocal velocity obstacles (RVO) [24] crowd simulation algorithm, navigating between randomly selected goal points. The robot and pedestrians share a maximum velocity of $1\,m/s$, reflecting typical indoor walking speeds. We denote a simulation environment with N_1 obstacles and N_2 pedestrians as $\mathrm{Sim}(N_1, N_2)$.

For offline datasets, we use different navigation policies to create diverse datasets. We initiate training of two reinforcement learning navigation policies, namely PPO and CHEAT, within $\mathrm{Sim}(5,3)$. CHEAT can access internal information from the simulator such as the precise positions and velocities of all pedestrians, which is not available for normal navigation policies. This privileged insight contributes to CHEAT's enhanced performance than PPO, albeit being considered a "cheating" model. Then, we implement DWA as a representative classical navigation policy. These three policies are employed within $\mathrm{Sim}(5,3)$ for data collection. Furthermore, the real-world expert navigation data from SCAND dataset [10] is incorporated to diversify our dataset. This results in multiple datasets for network training. We denote the dataset that contains data from policy A as $\mathbf{D}(\mathrm{A})$, while $\mathbf{D}(\mathrm{B},\mathrm{C})$ denotes a dataset uniformly mixed with data from policies B and C.

The following three categories of metrics are utilized to conduct a comprehensive evaluation of the navigation methods, including metrics of path quality, motion quality, and overall quality. Path quality is for evaluating the effectiveness of the predicted path explicitly, we have **Reduced Distance to the Goal per Unit Length (DpL):** this measures how much closer the predicted path gets to the goal per unit length of the path; **Minimum Distance to Obstacles (DtO):** the minimum distance between waypoints in the path and all obstacles; **Curvature (Curv):** the curvature of the path. Regarding motion quality, we compute **Average Velocity ($\boldsymbol{v}$ & $\boldsymbol{w}$)** and **Average Jerk ($\boldsymbol{v}_{\mathbf{jerk}}$ & $\boldsymbol{w}_{\mathbf{jerk}}$)** to reflect the speed and smoothness of motion respectively. And we also have some metrics to show the overall quality.**Arrival Rate (Arrv):** percentage where the robot reaches the goal without any collision; **Collision Rate ($\mathbf{Coll}_{\mathbf{obs}}$ & $\mathbf{Coll}_{\mathbf{ped}}$):** percentage where the robot collides with obstacles or pedestrians; **Stuck Rate (Stuck):** percentage where the robot runs out of timesteps before reaching the goal.

B. Data-driven Ability

To verify the data-driven ability of the NaviFormer approach, we choose four different quality datasets. According to their quality, they are sorted from low to high as $\mathbf{D}(\mathrm{DWA})$, $\mathbf{D}(\mathrm{DWA},\mathrm{PPO})$, $\mathbf{D}(\mathrm{PPO},\mathrm{CHEAT})$, $\mathbf{D}(\mathrm{CHEAT},\mathrm{SCAND})$. We individually train sequence modeling networks on these datasets, maintaining uniform parameters during training. Subsequently, four navigation approaches, each integrated with the same safety verification algorithm, are established. Then, we evaluate them within 10,000 navigation tests in a more intricate environment, $\mathrm{Sim}(10,5)$. The corresponding arrival rates are shown in Fig. 4.

[2]https://github.com/DRL-Navigation/img_env

The results indicate that incorporating higher-quality experiences from PPO substantially enhances navigation performance compared to that from only DWA experiences, with arrival rates steadily increasing as we introduce superior experiences from the cheating model and real-world experts. This establishes NaviFormer's data-driven ability.

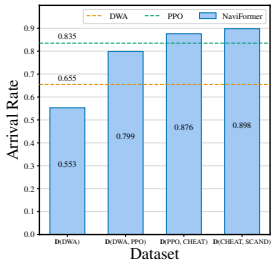


Fig. 4. Arrival rates of NaviFormer from different datasets in $\mathrm{Sim}(10,5)$.

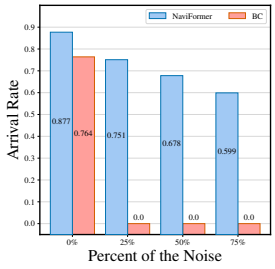


Fig. 5. Variation of arrival rate in $\mathrm{Sim}(10,5)$ with the number of noise.

Next, we conduct an extra experiment to compare NaviFormer with imitation learning, which has also been proven to have the data-driven ability [25]. A navigation policy network based on behavior cloning (BC), a representative imitation learning method, is trained and compared against NaviFormer on $\mathbf{D}(\mathrm{CHEAT})$. We demonstrate the noise resistance of these two methods by replacing some data in the dataset with harmful noise where the robot stays still.

The impact of the amount of noise in the dataset on the arrival rates of these two methods is shown in Fig. 5. While BC achieves a close arrival rate to NaviFormer on the noise-free dataset, it completely fails when noise is added, rendering it incapable of learning an efficacious navigation policy. In contrast, even after the addition of 75% noise, NaviFormer's arrival rate remains competitive, which shows the strong noise resistance of our method. Our experiments highlight NaviFormer's strengths in data-driven navigation and noise tolerance when compared to imitation learning techniques.

C. Efficacious Predictions of Future Paths

To ensure the network predicts future paths effectively, we conduct a single-step navigation test, which refers to not completing the entire navigation. Instead, after random initialization of the testing environment, the predicted future path in the first timestep is directly evaluated. This particular test design allows us to eliminate potential influences from later processes of navigation, such as local control, and directly assess the performance of our network.

We use the network trained on $\mathbf{D}(\mathrm{CHEAT},\mathrm{SCAND})$ as the representative. As baselines for comparison, we incorporate two classical path-planning algorithms. The first is the A* algorithm, which finds the optimal path from the start to the goal points. The first four points of the optimal path are considered the predicted future path of this method. The second baseline is the DWA algorithm. This method selects the optimal action, and we define the predicted future path of this method as the path obtained by moving with the optimal action for 4 timesteps.

We randomly generate 10,000 $\mathrm{Sim}(10,5)$ and conduct single-step tests within these environments. The outcome is summarized in Tab. I. The network-predicted future paths exhibit notably higher DpL values compared to classical algorithms, signifying their enhanced ability to approach the goal point efficiently. Despite their proximity to obstacles, the DtO value remains at a safe margin of 0.5 meters, ensuring adequate space. Furthermore, the curvature of network-predicted paths falls within a range comparable to DWA and A*, suggesting that the smoothness of the predicted path is on par with classical algorithms.

TABLE I
PERFORMANCE OF FUTURE PATHS FROM DIFFERENT METHODS

		Network (Ours)	A*	DWA
Path Quality	DpL	**0.957**	0.862	0.533
	DtO	0.516	0.667	**0.700**
	Curv	1.311	2.579	**0.644**

Thus, we confidently conclude that our sequence modeling network has adeptly acquired path-planning capabilities through offline datasets, thereby enabling the prediction of future paths of exceptional quality.

D. Performance and Generalization

In this section, we conduct a more comprehensive performance evaluation of the NaviFormer approach and extensively compare it with a series of baseline navigation methods. We implemented state-of-the-art classical and reinforcement learning navigation methods, denoted as DWA and PPO respectively. We also implement two representative offline reinforcement learning methods: conservative q-learning [27] and decision transformer [14], denoted as CQL and DT respectively. Additionally, we include behavior cloning, denoted as BC, as a baseline for imitation learning. In our notation, the NaviFormer approach is referred to as NF. While PPO is trained through online trial-and-error within $\mathrm{Sim}(5,3)$, CQL, DT, BC, and NF are all trained on $\mathbf{D}(\mathrm{CHEAT},\mathrm{SCAND})$.

We subject these navigation methods to 10,000 navigation tests within $\mathrm{Sim}(10,5)$. To further assess the generalization of these navigation methods, we extend our evaluations to include the BARN benchmark [26], a rigorous robot navigation benchmark. The comparative performance outcomes are summarized in Tab. II. Whether evaluated in $\mathrm{Sim}(10,5)$ or the BARN benchmark, NaviFormer consistently demonstrates the highest arrival rates. Its collision rates with pedestrians and obstacles are lower than those of PPO and are comparable to the classical navigation method, DWA. Notably, NaviFormer sustains its performance across both environments, unlike the performance drop observed in PPO, DT, or BC within the BARN benchmark. Although NaviFormer's smoothness may slightly lag behind DWA, it exhibits substantial improvement compared to PPO.

TABLE II

PERFORMANCE OF OUR APPROACH AND BASELINE

		NF (Ours)		PPO		DWA		DT		CQL		BC	
Overall Quality	Arrv	**0.898**	(**0.757**)	0.835	(0.685)	0.655	(0.690)	0.573	(0.132)	0.004	(0.000)	0.731	(0.235)
	Coll_{obs}	0.004	(0.240)	0.044	(0.315)	**0.000**	(**0.010**)	0.233	(0.868)	0.712	(1.000)	0.040	(0.233)
	Coll_{ped}	0.081	(0.000)	0.121	(0.000)	**0.073**	(0.000)	0.192	(0.000)	0.181	(0.000)	0.144	(0.000)
	Stuck	0.017	(0.003)	**0.000**	(0.000)	0.272	(0.300)	0.002	(0.000)	0.103	(0.000)	0.085	(0.532)
Motion Quality	v	0.526	(0.564)	0.632	(0.763)	0.182	(0.694)	**0.664**	(**0.986**)	0.277	(0.626)	0.397	(0.254)
	w	0.676	(0.591)	0.702	(0.656)	**0.140**	(0.154)	0.562	(**0.132**)	0.264	(0.256)	0.363	(0.329)
	v_{jerk}	4.951	(3.808)	6.698	(6.555)	**1.139**	(**0.261**)	2.238	(0.578)	2.386	(0.413)	2.661	(3.393)
	w_{jerk}	6.925	(6.926)	14.850	(20.006)	**2.132**	(1.414)	4.912	(1.248)	6.000	(**0.243**)	6.390	(2.923)

The values shown outside of the brackets are obtained from $\text{Sim}(10,5)$, while the values inside are obtained from the BARN benchmark [26].

In conclusion, NaviFormer boasts remarkable arrival rates, improved smoothness compared to PPO, and consistent safety levels close to DWA, all while demonstrating exceptional generalization.

E. Real-world Experiments

To validate the effectiveness and generalization of NaviFormer in the real world, we conduct real-world experiments using a Kobuki base TurtleBot 2. We design five representative scenarios to evaluate the performance of the navigation methods, as depicted in Fig. 6. In each scenario, we perform navigation tests using our NaviFormer and three baseline methods: DWA, PPO, and DT. The starting point and the target goal are set at a fixed distance of 6.4 meters in each scenario, and the maximum velocity is constrained to $0.6 m/s$ for safety considerations.

(a) Straight (b) Narrow (c) Obstacle

(d) S-shaped (e) U-shaped

Fig. 6. Five testing scenarios in the real world. The Straight scenario demonstrates the basic movement smoothness of the navigation methods. Narrow assesses the method's capability to navigate through confined spaces. Obstacle tests the fundamental obstacle avoidance ability. S-shaped evaluates the steering capacity while maintaining movement smoothness. And U-shaped shows the method's response to the local minimum issues.

The quantitative results for each scenario are presented in Tab. III. NaviFormer excels in maintaining a faster and more stable linear velocity in Straight and Narrow scenarios, while also exhibiting low angular velocity jerk. Additionally, it demonstrates more stable angular velocity in Obstacle and S-shaped scenarios. On the other hand, DWA performs well in most cases, except for the U-shaped scenario where it becomes stuck due to a local optimum. In contrast, NaviFormer effectively explores free space to overcome the local optimum.

TABLE III

PERFORMANCE IN THE REAL WORLD

		NF (Ours)	PPO	DWA	DT
Straight	v	**0.600**	**0.600**	0.580	0.580
	w	0.110	0.654	0.055	**0.032**
	v_{jerk}	**0**	**0**	0.040	0.101
	w_{jerk}	0.266	3.891	0.201	**0.115**
Narrow	v	**0.597**	Collide	0.576	0.580
	w	0.144		0.153	**0.094**
	v_{jerk}	**0.019**		0.117	0.037
	w_{jerk}	0.360		0.234	**0.144**
Obstacle	v	0.561	**0.600**	0.545	0.469
	w	0.416	0.756	**0.288**	0.344
	v_{jerk}	0.293	**0**	0.263	0.232
	w_{jerk}	0.898	3.654	1.299	**0.614**
S-shaped	v	**0.598**	0.518	0.584	Collide
	w	**0.360**	0.740	0.408	
	v_{jerk}	**0.016**	1.259	0.036	
	w_{jerk}	0.929	3.298	**0.589**	
U-shaped	v	0.387	**0.600**	Stuck	Stuck
	w	**0.680**	0.747		
	v_{jerk}	0.247	**0**		
	w_{jerk}	**3.237**	4.41		

"Collide" and "Stuck" indicate that the robot does not reach the goal in that scenario.

These outcomes demonstrate the generalization capability of NaviFormer across diverse real-world environments, and that NaviFormer achieves high-quality motion performance and ensures reliable arrival rates.

V. CONCLUSION

In this paper, we propose a data-driven robot navigation approach, NaviFormer, that uses sequence modeling to predict future paths and safety verification algorithms for rule-based safety constraints. Our simulated and real-world experiments validate that NaviFormer possesses data-driven ability and strong noise resistance and highlight NaviFormer's exceptional navigation performance and generalization.

NaviFormer has the potential to autonomously enhance its performance during runtime without any additional manual process, though it is still challenging to achieve this data closed-loop as its utilization of real-time navigation data is not efficient enough.

REFERENCES

[1] B. Singh, R. Kumar, and V. P. Singh, "Reinforcement learning in robotic applications: a comprehensive survey," *Artificial Intelligence Review*, pp. 1–46, 2022.

[2] M. Luong and C. Pham, "Incremental learning for autonomous navigation of mobile robots based on deep reinforcement learning," *Journal of Intelligent & Robotic Systems*, vol. 101, no. 1, p. 1, 2021.

[3] G. Chen, L. Pan, P. Xu, Z. Wang, P. Wu, J. Ji, X. Chen, *et al.*, "Robot navigation with map-based deep reinforcement learning," in *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*. IEEE, 2020, pp. 1–6.

[4] H. Shi, L. Shi, M. Xu, and K.-S. Hwang, "End-to-end navigation strategy with deep reinforcement learning for mobile robots," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2393–2402, 2019.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[7] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: a survey," *Autonomous Robots*, vol. 46, no. 5, pp. 569–597, 2022.

[8] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[9] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Efficient trajectory optimization using a sparse model," in *2013 European Conference on Mobile Robots*. IEEE, 2013, pp. 138–143.

[10] H. Karnan, A. Nair, X. Xiao, G. Warnell, S. Pirk, A. Toshev, J. Hart, J. Biswas, and P. Stone, "Socially compliant navigation dataset (scand): A large-scale dataset of demonstrations for social navigation," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 807–11 814, 2022.

[11] N. Carlevaris-Bianco, A. K. Ushani, and R. M. Eustice, "University of michigan north campus long-term vision and lidar dataset," *The International Journal of Robotics Research*, vol. 35, no. 9, pp. 1023–1035, 2016.

[12] A. Rudenko, T. P. Kucner, C. S. Swaminathan, R. T. Chadalavada, K. O. Arras, and A. J. Lilienthal, "Thör: Human-robot navigation data collection and accurate motion trajectories dataset," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 676–682, 2020.

[13] R. F. Prudencio, M. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy," *Review, and Open Problems*, pp. 1–21, 2022.

[14] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[16] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[17] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.

[18] 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.

[19] J. Zeng, R. Ju, L. Qin, Y. Hu, Q. Yin, and C. Hu, "Navigation in unknown dynamic environments based on deep reinforcement learning," *Sensors*, vol. 19, no. 18, p. 3837, 2019.

[20] S. Yao, G. Chen, Q. Qiu, J. Ma, X. Chen, and J. Ji, "Crowd-aware robot navigation for pedestrians with multiple collision avoidance strategies via map-based deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8144–8150.

[21] R. F. Prudencio, M. R. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[22] J. Velagic, B. Lacevic, and N. Osmic, "Efficient path planning algorithm for mobile robot navigation with a local minima problem solving," in *2006 IEEE International Conference on Industrial Technology*. IEEE, 2006, pp. 2325–2330.

[23] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[24] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE international conference on robotics and automation*. Ieee, 2008, pp. 1928–1935.

[25] S. Belkhale, Y. Cui, and D. Sadigh, "Data quality in imitation learning," *arXiv preprint arXiv:2306.02437*, 2023.

[26] Z. Xu, B. Liu, X. Xiao, A. Nair, and P. Stone, "Benchmarking reinforcement learning techniques for autonomous navigation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9224–9230.

[27] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.