Context Aware Mamba-based Reinforcement Learning for Social Robot Navigation

Syed Muhammad Mustafa
Computer Engineering
Habib University
Karachi, Pakistan
sm06554@st.habib.edu.pk

Zain Ahmed Usmani
Computer Science
Habib University
Karachi, Pakistan
zu06777@st.habib.edu.pk

Omema Rizvi*
Computer Science
Habib University
Karachi, Pakistan
or06360@st.habib.edu.pk
*Corresponding author

Abdul Basit Memon
Electrical & Computer Engineering
Habib University
Karachi, Pakistan
basit.memon@sse.habib.edu.pk

Muhammad Mobeen Movania
Computer Science
Habib University
Karachi, Pakistan
mobeen.movania@sse.habib.edu.pk

2024 12th International Conference on Control, Mechatronics and Automation (ICCMA) | 979-8-3315-1751-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICCMA63715.2024.10843924

***Abstract*—Social robot navigation (SRN) is a relevant problem that involves navigating a pedestrian-rich environment in a socially acceptable manner. It is an essential part of making social robots effective in pedestrian-rich settings. The use cases of such robots could vary from companion robots to warehouse robots to autonomous wheelchairs. In recent years, deep reinforcement learning has been increasingly used in research on social robot navigation. Our work introduces CAMRL (Context-Aware Mamba-based Reinforcement Learning). Mamba is a new deep learning-based State Space Model (SSM) that has achieved results comparable to transformers in sequencing tasks. CAMRL uses Mamba to determine the robot's next action, which maximizes the value of the next state predicted by the neural network, enabling the robot to navigate effectively based on the rewards assigned. We evaluate CAMRL alongside existing solutions (CADRL, LSTM-RL, SARL) using a rigorous testing dataset which involves a variety of densities and environment behaviors based on ORCA and SFM, thus, demonstrating that CAMRL achieves higher success rates, minimizes collisions, and maintains safer distances from pedestrians. This work introduces a new SRN planner, showcasing the potential for deep-state space models for robot navigation.**

***Index Terms*—Social Robot Navigation (SRN), Deep Reinforcement Learning (DRL), Socially acceptable robot navigation, Path planning, Human-robot interaction (HRI), Mobile robots, Autonomous wheelchairs, Warehouse robots, Companion robots, Deep State Space Models (DSSM), Mamba.**

I. Introduction

Mobile robots are increasingly being deployed in pedestrian-dense environments such as shopping centers, restaurants, convention halls, and warehouses. In these crowded settings, effective navigation requires that the robot generates collision-free paths while avoiding numerous dynamic agents. an agent is any entity in the environment that has the agency to make decisions that change the state of the system. Additionally, it is desirable that the robot behavior in such environments adheres to human social norms, ensuring seamless integration. The challenge is compounded by limited communication between the robot and surrounding agents, preventing direct query of intent (an agent's intention refers to its desired goal position and preferred velocities). To address this, it is often necessary for the robot to model the behavior of each agent in the crowd, as well as its interactions with others, enabling the path planner to devise collision-free paths based solely on observable agent behavior.

The problem is complex as the communication between the robot and the other agents in the crowd, such as humans, is limited. This means that the robot's path planning module must plan a collision-free path using only the observable information available, i.e. intentions (goal position and preferred velocities) of agents would be hidden from the robot.

Reaction-based collision avoidance algorithms, such as Optimal Reciprocal Collision Avoidance (ORCA) [1] and the Social Force Model (SFM) [2], are widely used for this problem. These algorithms employ a one-step look-ahead strategy to avoid both static and dynamic obstacles at each time step. While computationally efficient, this approach often results in myopic and oscillatory behaviors [3]. In contrast, recent advancements in deep reinforcement learning (DRL) have demonstrated its capacity to solve complex tasks, sometimes surpassing human performance. DRL holds significant promise for crowd navigation, due to its ability to model complex decision-making processes [3]–[7]. In learning-based methods, the crowd navigation problem is typically formulated as a Markov decision process, where a DRL agent uses a neural network to select the next action, either by directly predicting the optimal action or by choosing one that maximizes the expected value of state at the next time step.

The neural network employed by the deep reinforcement learning (DRL) agent for social navigation typically consists of two main components: the crowd state encoder and the value/policy network (see Figure 1). The crowd state encoder is responsible for the spatial processing of crowd data, map-

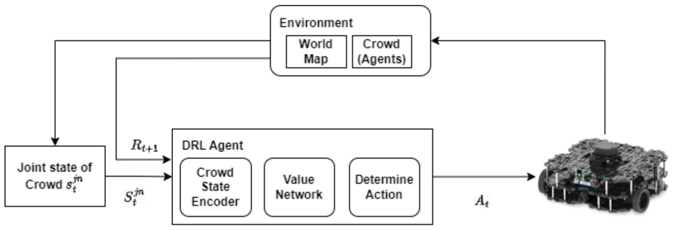


Fig. 1: Framework for collision avoidance using deep reinforcement learning

ping all observable agent information to a latent space. This approach is designed to accommodate a variable number of agents [5] and to capture the interactions between agents and the robot [6]. Most research in social robot navigation (SRN) using DRL has concentrated on improving crowd state encoder strategies, and comparatively little attention has been directed toward the value network. In current implementations, the value network is modeled as a simple multi-layer perceptron.

In recent years, deep learning-based state space models (SSM) [8], [9] have shown to be powerful tools for sequence modelling tasks such as audio generation, natural language processing, and DNA sequencing. "Mamba" is a new deep learning-based State Space Model (SSM) that has achieved results comparable to transformers in sequence modelling tasks [9].

This paper presents a novel approach for collision avoidance using deep reinforcement learning, CAMRL (Context Aware Mamba-based Reinforcement Learning). Our main contribution here is introducing temporal processing of crowd states (joint state of the robot and all the other agents) by utilizing Mamba's ability to model sequences. We also identified gaps in the simulation on which previous models were evaluated on, and hence used a more rigorous testing approach [10] to evaluate the effectiveness of our model.

II. The collision avoidance paradigm

The state of the agents in the context of non-communicating multi-agent collision can be defined as the concatenation of its observable state and its hidden state. For each agent, the position $\mathbf{p} = [\mathbf{p_x}, \mathbf{p_y}]$, velocity $\mathbf{v} = [\mathbf{v_x}, \mathbf{v_y}]$, and the radius of the bounding circle of the agent, r, can be observed. The intended goal position $\mathbf{p_g} = [\mathbf{p_{gx}}, \mathbf{p_{gy}}]$, and the preferred velocity $\mathbf{v_{pref}}$ are hidden. Thus, the problem can be formulated as a partially observable sequential decision-making problem. In this paper, the hidden state of an agent would be sometimes called the intention of the robot. Furthermore, the joint state of the system denotes the concatenation of the complete state of the robot and the observable state of the other agents, i.e. for a system with n+1 agents the joint state is given by $s_t^{jn} = [s_t, \tilde{s}_t^{o,1}, \tilde{s}_t^{o,2}, ..., \tilde{s}_t^{o,n}]$, where the observable state of i_{th} other agent is given by $\tilde{s}_t^{o,i}$.

The goal of a collision avoidance algorithm is to reduce the expected time to reach the goal while avoiding any collisions. The n-agent collision avoidance problem is mathematically formulated following the approach presented in [3].

$$argmin_{\pi(s_t^{jn})}\mathbb{E}[t_g|s_0^{jn}, \pi, \tilde{\pi}_1, ...\tilde{\pi}_n] \tag{1a}$$

$$s.t. \; \forall\; t, i : \|p_t - \tilde{p}_{i,t}\|_2 \geq r + \tilde{r}_i \tag{1b}$$

$$p_{end} = p_g \tag{1c}$$

$$\begin{aligned} p_t &= p_{t-1} + \Delta t\; \pi(s_{0:t}^{jn}) \\ \tilde{p}_{i,t} &= \tilde{p}_{i,t-1} + \Delta t\; \tilde{\pi}_i(\tilde{s}_{0:t}^{jn}), \end{aligned} \tag{1d}$$

where t_g is the time to reach the goal, s_t^{jn} is the joint state of the system at time t, $\tilde{s}_t^{jn}$ is the joint state at time t of the system concerning another agent, π and $\tilde{\pi}_i$ are the robot and i^{th} agents policy (a function that maps the states to actions), p_g is the goal position, p_t, $\tilde{p_{t,i}}$ is the position of the robot and the i^{th} agent at time t, r and r_i is the radius of the robot and the i^{th} agent, Δt is the time step and p_{end} is the position of the robot at the end of the current episode. Equation 1a is the expectation constraint, i.e. the robot's policy should minimize the expected time to reach the goal. Equation 1b is the collision constraint that the robot should not collide with the i^{th} agent. Equation 1c is the goal constraint, i.e. the robot is at the goal position at the end of the episode

III. Related Work

This section reviews some basic and relevant ideas important to deep reinforcement learning-based social robot navigation solutions and the background for Mamba and state space models.

A. DRL models for collision avoidance

A number of DRL-based collision avoidance algorithms have been proposed to date. Some of these algorithms along with their brief descriptions are included here for reference.

1) **CADRL**: It uses a simple multi-layer perception (MLP) to approximate the value function [3]. In this model the crowd encoding layer does not exist and the value network is operating on joint state.
2) **GA3C CADRL (LSTM-RL)**: It uses a long short-term memory (LSTM) to compress all the observable information to a fixed-length vector before passing it into an MLP block [5].
3) **SARL**: It uses a social attention network on top of an MLP, assigning a weight to each pedestrian in the crowd [6].
4) **LM-SARL**: It uses a local map as an input to the social attention network. This is done to model human-to-human interactions as well [6].
5) **Relational Graph learning**: It uses graph convolution neural networks to learn interaction between all the agents in the environment [7].

As noted in [10], algorithms trained and tested on simple circle crossing environments (see Figure 3), which has been the case in [3]–[7], often fail to generalize to more complex, real-world scenarios. This limitation arises because such environments do not accurately simulate the dynamics of real-world crowds [11]. To address this, [10] has proposed improved training schemes using more diverse datasets, which show promise and will also be utilized in this work. Nevertheless, further investigation is needed to assess how effectively these simulated datasets replicate real-world crowd behaviors.

B. State Space Models

State space models (SSM) have been widely used to describe dynamical systems in econometrics and finance [12], control systems [13], robotics [11], etc., using first-order differential equations. The standard state-space model in continuous time is described by Equation 2 and in discrete-time by the recurrence relations in Equation 3, where x is the state, u is the input signal/sequence and y is the output signal/sequence.

$$\begin{aligned} x'(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2}$$

where $A, B, C,$ and D are matrices of appropriate sizes.

The continuous form of the state space model can be discretised using a fixed step size Δ and any discretization technique such as zero order hold sampling.

$$\begin{aligned} x_n &= \bar{A}x_{n-1} + \bar{B}u_n \\ y_n &= \bar{C}x_n + \bar{D}u_n \end{aligned} \tag{3}$$

The discrete form of SSM can also be represented using a convolutional view [14] :

$$y = \bar{K} * u \tag{4}$$

$$\bar{K} \in \mathbb{R} := K_L(\bar{A}, \bar{B}, \bar{C}) := (C\bar{A}^iB)_{i\in[L]} \tag{5}$$

where $[L]$ is an indexing set defined over the input sequence and $\bar{K}$ is called the SSM convolutional filter. The convolutional representation of SSM is relevant because now output sequence y given the input sequence u can be computed efficiently using Fast Fourier Transform [8].

C. State Space Models in Learning

Basic state-space models (SSMs) often perform poorly when used as a model in deep learning applications, where matrices A, B, C, and D are learned. However, recent work [9], [14]–[16] has achieved significant success with SSMs by imposing special structures on the state matrix A. One such example is the structured state-space model (S4) [15], where the matrix A is designed as a normal plus low-rank matrix, initialized using the HiPPO matrix. The HiPPO matrix provides an optimal solution to an online function approximation problem [17], and models utilizing HiPPO have shown remarkable performance on tasks involving long-range dependencies, outperforming transformers on the PathBench test suite. Additionally, due to the convolutional representation of SSMs, they can be parallelized, offering faster training times compared to recurrent neural networks [15]. And with their recurrent representation of SSMs they can infer faster than a transformer (which scales quadratically with the input sequence).

1) Selective State Space Model (Mamba): An addition to the S4 model, called Mamba, was presented by Gu et al. in [9]. Unlike S4, which uses time-invariant parameters, Mamba proposes selective SSMs through input-dependent parameterization, enabling the model to selectively propagate or forget information based on the current token. This allows the model to use only the relevant information from the input sequence. The algorithm for S6 (Mamba) is given in **Algorithm 1**.

Algorithm 1 Mamba (S6) [9]

```
Input: $x : (B, L, D)$
Output: $y : (B, L, D)$
A:$(D, N) \leftarrow Parameter$ {Represents structured $N \times N$ matrix}
B:$(B, L, N) \leftarrow s_B(X)$
C:$(B, L, N) \leftarrow s_C(x)$
$\Delta : (B, L, D) \leftarrow \tau_\Delta(Parameter + s_\Delta(x))$
$\overline{A}, \overline{B} : (B, L, D, N) \leftarrow discretize(\Delta, A, B)$
$y \leftarrow SSM(\overline{A}, \overline{B}, C)(x)$ {Time varying: recurrence(scan) only}
return $y$
```

2) State Space Models in the Context of Reinforcement Learning: A reinforcement learning algorithm that is capable of long-term planning is one of the central goals of reinforcement learning (RL) [18]. The ability of DSSM models to handle long-range dependencies and handle large contexts opens new possibilities in RL. Due to the cutting-edge nature of DSSM, little work has been done to investigate its efficacy in reinforcement learning. However, the work that has been done indicates that DSSM is a promising avenue for RL. For example, [19] used S5 (a variant of S4) for online meta-reinforcement learning where S5 outperformed LSTMs [20], while [18] showed an offline implementation of S4 and showed that S4 outperformed transformers in various RL tasks. In this paper, we are adopting this approach for reinforcement learning and testing it for social robot navigation.

IV. Methodology

In this section, we first discuss the DRL algorithm used, the reward function, and how it connects to social robot navigation, SSMs in the context of DRL, and then explain our model CAMRL (Context-Aware Mamba-based Reinforcement Learning) in detail.

A. Preliminaries

1) Deep V-learning: Deep V-learning, a deep reinforcement learning framework based on the Deep Q-Network algorithm but designed for continuous action spaces, was first introduced in [3] for the development of CADRL. It has two learning steps:

- **Imitation Learning:** The value network is trained on the training dataset.
- **Reinforcement Learning:** Iteratively improves the value network through experience replay and target network updates.

2) Reward Function: The reward function is modelled as follows:

$$R_1(s,a_t) = \begin{cases} -0.25, & \text{if } \mathbf{d_t} \leq 0 \\ \dfrac{(\mathbf{d_t} - \mathbf{r_c})\Delta t}{2}, & \text{else if } \mathbf{d_t} < \mathbf{r_c} \\ 1, & \text{else if } \mathbf{p_t} = \mathbf{p_g} \\ -0.5, & \text{else if } \mathbf{t} = \mathbf{25} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $\mathbf{d_t}$ is the separation distance between robot and the humans, $\mathbf{p_t}$ is robot's current position and $\mathbf{p_g}$ is the goal position, and $\mathbf{r_c}$ is the radius of discomfort.
This reward function encourages the robot to avoid collisions, not violate the radius of discomfort of a human, and to finish the navigation task in a set time.

3) State space models in the context of reinforcement learning: The discrete representation can be thought of as sampling from a continuous signal. The authors reported that deep-state space models are good at handling long-range dependencies. Given that the optimal value function is a causal system, i.e. dependent on previous states:

$$V^8(\mathbf{s_0^{jn}}) = \sum_{k=0}^{T}[\gamma^t R(\mathbf{s_t^{jn}}, \pi^*(\mathbf{s_t^{jn}}))], \tag{7}$$

where $V(s_t^{jn})$ is the value of the joint state at time t, R_t is the reward at time t, $\gamma \in [0,1)$ is a discount factor, and $\pi^*(\mathbf{s_t^{jn}})$ is the optimal policy [3].

Hence, it makes sense to use a neural network that can model a sequence of states instead of a simple multi-layered perceptron.

B. Context Aware Mamba-based Reinforcement Learning

Leveraging the ability of SSMs to handle long-range dependencies effectively, our proposed model, Context-aware Mamba-based Reinforcement Learning (CAMRL), is provided with a sequence of states at the present and past times. This is different from most previous works that provide the current state to the value network. The CAMRL model, as illustrated in Figure 2, can be divided into two primary components: the crowd state encoder and the value network. The crowd state encoder uses a GRU encoder to process a temporal slice of joint states, that will be referred to as the temporal crowd state vector, i.e. the joint state of the crowd from time $t-T$ to t, i.e. $s_{t-T:t}^{jn}$. This encoder functions as a mechanism that compresses all the information of the crowd at a particular time instant into a fixed-length vector L_t within a latent space, referred to as the crowd space which has all configurations of a crowd.

Following the encoding process, the output of the crowd state encoder is fed into the value network. This value network is modeled using a Mamba block, which consists of four stacked Mamba layers. These vectors are then processed through the Mamba block to produce the temporal value vector $v_{k=t-T}^{t}$, i.e. a vector containing the values at time $t-T$ to t. The final value at time t is used to determine the action a_t for the environment. The action that maximizes value at time t is selected.

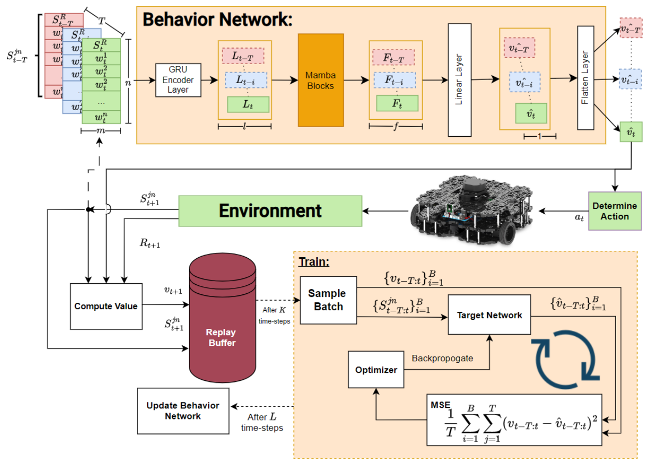


Fig. 2: Architecture for CAMRL

During training, a replay buffer stores the temporal crowd state and value vectors, which are used to train the target network through backpropagation and optimization based on the mean squared error (MSE) loss on a batch of size B of the temporal value vector ($\{v_{t-T:t}\}_{i=1}^{B}$) and the predicted batch of temporal value vectors ($\{\hat{v}_{t-T:t}\}_{i=1}^{B}$). After every K episode the behavior network is updated by the target network.

This systematic approach allows CAMRL to effectively learn and predict actions based on long-range dependencies, ensuring robust performance in dynamic and complex environments.

V. Experimental Setup

The developed CAMRL model was tested for social-robot navigation in simulation. The training and testing setup is outlined in this section. Metrics used for comparing our model with existing models are described in subsection V-C.

A. Environment setup

The typical setup for training and testing employed in literature [3]–[6], [10] is shown in Figure 3(a). The basic idea is as follows: humans (other agents), indicated by numbered circles, start at a random position on the perimeter of a circle or square and their goal position is set roughly on the opposite side of the shape. The other agents try to navigate to their end goals using ORCA [1]. The robot, represented by the yellow circle, follows the algorithm-determined policy to navigate to its goal, the red star, avoiding any collisions. The robot is only aware of the variables in the observable state detailed in section II. The robot is not visible to the humans during testing or training, ensuring the trained policy does not have the robot forcing the other agents to change their trajectories to allow the robot to achieve a high reward [21].

However, the simplistic circle crossing setting does not generalize well to real-world scenarios [22], [10]. Thus [10] proposed the Diverse-4 dataset, introducing two additional environments, dense and large, that take into account the crowd density and the distance the robot has to cover. It also recommends SFM as the model for human agents for more rigorous testing of the policy. The test environments are divided into simple (or baseline), dense, and large. The simple environment is small with a crowd density of 0.1 humans per m^2, and is typically used in the literature. The dense environment has the same area but doubles the crowd density, simulating navigation in challenging conditions. The large environment covers a larger area while maintaining the baseline density; this allows for evaluating the model's long-horizon task capabilities.

B. Training and Testing Setup

Our model was trained in the baseline circle crossing environment used by previous models like CADRL [3], LSTM-RL [5], and SARL [6] with pedestrians modelled by ORCA only.

Based on the recommendations in [10], our model was tested in six testing environments, namely, baseline circle and square crossing, dense circle and square crossing, large circle and square crossing (as shown in Figure 3) with 50 test cases in each environment. These environments were chosen to span diverse crossing types, crowd densities, and task lengths.

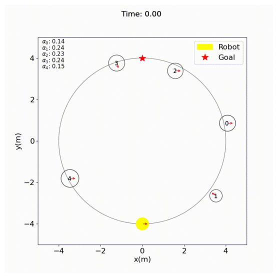
Additionally, we used both ORCA and SFM for crowd modelling in tests, which allowed for a more realistic simulation of humans [22].

For training, the models learn their policy over a number of episodes. In each episode, the robot tries to reach the goal from its start position while trying to avoid other agents. An episode is terminated when the robot either reaches the goal, collides with an agent or does not make it to the goal within the given time (times out). In our case, the time out is 25 seconds.

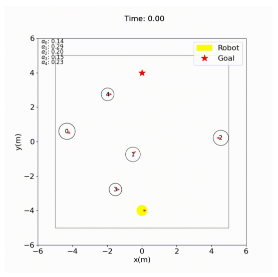
We trained and tested our model using a 32 GB Titan RTX with an Intel Core i7-9700K CPU at 3.60GHz. Mamba utilizes CUDA functionalities and hence requires a GPU; however, the other models can be trained using a CPU only.

C. Metrics

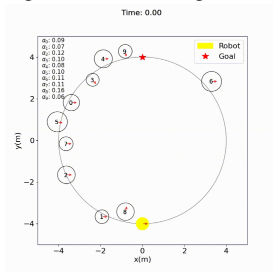
In order to evaluate the performance of our model we used standard metrics commonly used for comparing DRL models in the literature. We compared our model with three state-of-the-art models found in the literature: CADRL [3], LSTM-RL [5], and SARL [6]. The metrics are defined in Table I.



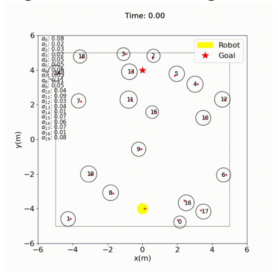
(a) Baseline Circle Crossing (4m radius, 5 agents)



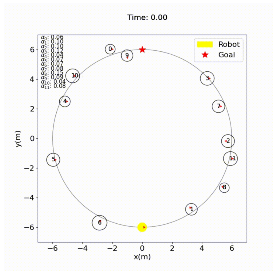
(b) Baseline Square Crossing (10m wide, 10 agents)



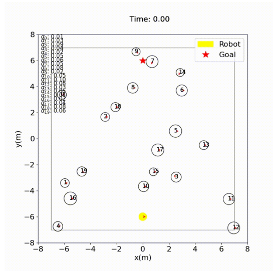
(c) Dense Circle Crossing (4m radius, 10 agents)



(d) Dense Square Crossing (10m wide, 20 agents)



(e) Large Circle Crossing (6m radius, 12 agents)



(f) Large Square Crossing (14m wide, 20 agents)

Fig. 3: Diverse-4 Testing Scenarios from [10].

TABLE I: Metrics identified for Evaluation

Metrics	**Definition**
Success Rate	The amount of times the robot reaches the goal without collisions and within the given time frame in the given number of episodes.
Collision Rate	The amount of times the robot collides with an agent or an obstacle in the given number of episodes.
Timeout Rate	The amount of times the robot does not reach the goal within the given time and avoids collisions in the given number of episodes.
Time taken (s)	The average time the robot takes to reach the goal.
Discomfort Frequency	The number of times the robot got too close to other agents in the given number of episodes.
Discomfort Distance (m)	The average distance the robot maintained with other agents.

VI. Results

A. Comparison of CAMRL with other models

The results of testing are summarized in Table II.

Compared to CADRL, LSTM-RL and SARL, our model, CAMRL, performs the best. It gives a 5% higher success rate than SARL, which is the best out of the three. CAMRL also has the least number of collisions and manages to bring down the collision rate by a margin of 6% compared to SARL. CAMRL has a timeout rate of 1%, where it does not reach the goal in the given time but manages to avoid collisions. This weakness is covered up by the higher success rate and lower collision rate compared to the other three models, for scenarios where avoiding collisions might be a more preferred outcome than the robot not reaching its end destination in time. These results can also be seen in Figure 4. Additionally, CAMRL maintains a greater distance from other agents, on average, compared to other models.

TABLE II: Comparison of our model with state-of-the-art models on all metrics

Policy	Success	Collision	Timeout	Time taken (s)	Disc. Freq	Disc. Dist. (m)
ORCA	0.51	0.49	**0.00**	12.26 ± 2	0.24	0.09 ± 0.1
CADRL	0.73	0.16	0.11	16.04 ± 4	0.10	0.13 ± 0.06
LSTM-RL	0.56	0.38	0.06	15.75 ± 5	0.21	0.09 ± 0.06
SARL	0.83	0.17	**0.00**	$\mathbf{11.20 \pm 1}$	0.21	0.13 ± 0.05
CAMRL	**0.88**	**0.11**	0.01	13.91 ± 2	**0.07**	$\mathbf{0.16 \pm 0.04}$

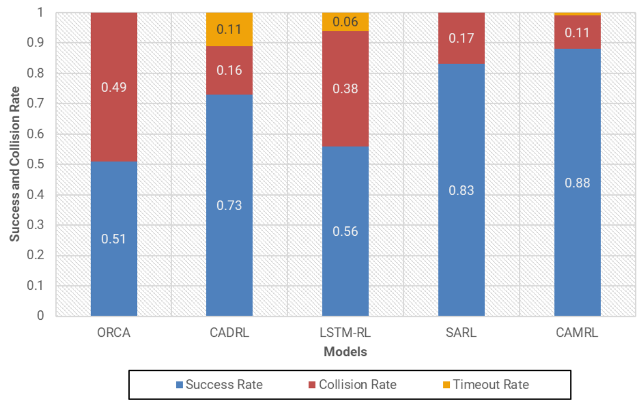


Fig. 4: Comparison of our model with state-of-the-art models on the basis of success, timeout and collision rate

Overall, it performed better in maintaining comfort distance by maintaining discomfort frequency at 7%, the least compared to its counterparts.

VII. Conclusion

This work proposes CAMRL, a new model for path planning in social robot navigation (SRN). CAMRL performed either comparably or better than current state-of-the-art solutions across a rigorous set of test cases. This work also demonstrates that state space modelling offers promising solutions for path planning in SRN. Following this, future work could involve comparing CAMRL with graph learning solutions [7] [21] and implementing CAMRL on a real-world robot.

References

[1] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Robotics Research: The 14th International Symposium ISRR*. Springer, 2011, pp. 3–19.
[2] D. Helbing and P. Molnar, “Social force model for pedestrian dynamics,” *Physical review E*, vol. 51, no. 5, p. 4282, 1995.
[3] Y. F. Chen, M. Liu, M. Everett, and J. P. How, “Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning,” in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 285–292.
[4] Y. F. Chen, M. Everett, M. Liu, and J. P. How, “Socially aware motion planning with deep reinforcement learning,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1343–1350.
[5] M. Everett, Y. F. Chen, and J. P. How, “Motion planning among dynamic, decision-making agents with deep reinforcement learning,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.
[6] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, “Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning,” in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 6015–6022.
[7] C. Chen, S. Hu, P. Nikdel, G. Mori, and M. Savva, “Relational graph learning for crowd navigation,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 007–10 013.
[8] A. Gu, K. Goel, and C. Ré, “Efficiently modeling long sequences with structured state spaces,” *CoRR*, vol. abs/2111.00396, 2021. [Online]. Available: https://arxiv.org/abs/2111.00396
[9] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” *arXiv preprint arXiv:2312.00752*, 2023.
[10] A. Sigal, H.-C. Lin, and A. Moon, “Improving reinforcement learning training regimes for social robot navigation,” *arXiv preprint arXiv:2308.14947*, 2023.
[11] S. Thrun, “Probabilistic robotics,” *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.
[12] J. D. Hamilton, “State-space models,” *Handbook of econometrics*, vol. 4, pp. 3039–3080, 1994.
[13] K. J. Åström and R. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.
[14] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, “Combining recurrent, convolutional, and continuous-time models with linear state space layers,” *Advances in neural information processing systems*, vol. 34, pp. 572–585, 2021.
[15] A. Gu, K. Goel, and C. Re, “Efficiently modeling long sequences with structured state spaces,” in *International Conference on Learning Representations*, 2021.
[16] A. Gu, I. Johnson, A. Timalsina, A. Rudra, and C. Ré, “How to train your hippo: State space models with generalized orthogonal basis projections,” *arXiv preprint arXiv:2206.12037*, 2022.
[17] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, “Hippo: Recurrent memory with optimal polynomial projections,” *Advances in neural information processing systems*, vol. 33, pp. 1474–1487, 2020.
[18] S. B. David, I. Zimerman, E. Nachmani, and L. Wolf, “Decision s4: Efficient sequence-based rl via state spaces layers,” in *The Eleventh International Conference on Learning Representations*, 2022.
[19] C. Lu, Y. Schroecker, A. Gu, E. Parisotto, J. Foerster, S. Singh, and F. Behbahani, “Structured state space models for in-context reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
[20] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, “Rl2: Fast reinforcement learning via slow reinforcement learning,” *arXiv preprint arXiv:1611.02779*, 2016.
[21] S. Liu, P. Chang, Z. Huang, N. Chakraborty, K. Hong, W. Liang, D. L. McPherson, J. Geng, and K. Driggs-Campbell, “Intention aware robot crowd navigation with attention-based interaction graph,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 12 015–12 021.
[22] C. Mavrogiannis, F. Baldini, A. Wang, D. Zhao, P. Trautman, A. Steinfeld, and J. Oh, “Core challenges of social robot navigation: A survey,” *ACM Transactions on Human-Robot Interaction*, vol. 12, no. 3, pp. 1–39, 2023.