# Social Distancing Simulator: 2D Planner in a Multi-Agent and Dynamic Environments

1st Weiqi Wang
*Mechanical and Aerospace Engineering*
*University of California, Los Angeles*
wwang79@ucla.edu

2nd Nicole Nadim
*Electrical and Computer Engineering*
*University of California, Los Angeles*
nicolenadim@ucla.edu

3rd Shengkai Peng
*Mechanical and Aerospace Engineering*
*University of California, Los Angeles*
shp066@g.ucla.edu

4th Michael Kim
*Electrical and Computer Engineering)*
*University of California, Los Angeles*
mhk150230@g.ucla.edu

***Abstract*—Path planning in an environment cluttered with moving obstacles is a challenging and extensively researched topic. Mature algorithms in such setting can be deployed to various applications in our daily life: household assistance robots, self-driving vehicles, warehouse robots etc. In this paper, we propose an approach that breaks down the complex setup into two separate but interconnected problems: dynamic obstacle trajectory prediction given social context and path planning of a single robot through an environment with dynamic obstacles. We examined two methods, one is a traditional RRT* and MPC path planning supplemented with an RNN, and the other is Deep Q-learning. Within a simulated environment, we showed that using social information for path prediction greatly aids the performance of robot's path planner.**

***Index Terms*—Robotics, path planning, Neural Network, Reinforcement Learning**

## I. Introduction

Throughout this course we discussed multi-agent systems, more specifically, pursuer-evader problems with both single and multiple pursuer/evader configurations. The formulation of this system is described with a Markov Decision Process (MDP) problem, as well as a method prescribed by function approximation. Despite these explorations into multi-agent systems, there has yet to be a description of a problem set where these agents have to navigate through a dynamic environment with multiple obstacles.

In our everyday life, we navigate through complex environments without a second thought. We drive in downtown LA or walk through the heavily trafficked UCLA campus effortlessly. When we see a pedestrian at a red light, we know they intend to cross the street, and that a person walking towards a drinking fountain is likely going to pause for a drink. We have this knowledge from dozens of years of learning in real world. We understand the dynamics and intentions of other people, thus we are able to predict the world's state in the next few seconds and plan our steps accordingly. Replicating this behavior in robotics has been a topic of research for decades, and is our main motivation for this project. As our main contribution in this work, we encode social elements in the prediction of dynamic obstacles, allowing our agents to proactively avoid collision.

Our project goal is defined As:

*A single agent navigating through an environment cluttered with predictable dynamic obstacles.*

This can be further broken down into two problems:

*1. Predicting other agents' trajectories and goals given social context (the position of goals and other agents)*
and
*2. Path planning of a single agent through an environment cluttered with predictable dynamic obstacles.*

We introduce two pipelines so that we may tackle the two problems from different approaches. Pipeline 1: a Recurrent Neural Network (RNN) is implemented to learn the obstacles' behaviors while they navigate the environment, and produce a prediction to use in the planning module. For path planning, we use $RRT^*$ as a long time horizon planner and MPC for local path planning.

Pipeline 2: a Q-Learning method is implemented in a decentralized setting. The agent is given rewards and an upper bound of steps to explore and learn its environment. Due to the large state space of this setting, it becomes unsuitable for a traditional Q-Learning approach. Therefore, Deep Q-Learning (DQN) is implemented to account for the complexity of the problem. The agent is expected to learn and understand the relationships between different states within this dynamic environment and with this learned behavior, plan a suitable path from its starting point to its prescribed goal point.

We show that both pipelines allow the robot agent to navigate through a simplified environment. Additionally, we show that using a RNN for path prediction can significantly improve the performance of the path planning module.

For the purpose of testing our framework, we created a virtual environment in which obstacles exhibit predictable patterns. This is accomplished by collecting human input for similar objectives.

## II. Related Works

### A. Prediction of Human Full-Body Movements with Motion Optimization and Recurrent Neural Networks

The works of Marc *et al.* [3] serve as one of our main inspiration. In this literature, the authors introduced a framework that uses RNN and motion planning algorithms to predict human motion. The low-level dynamics are modeled as a Gaussian process, allowing them to use RNN for kinematic predictions. The network learns from previous human motion capture data. To ensure their predicted path is natural and doesn't violate any constraints, an optimization algorithm is used to further refine the prediction. Such framework is able to generate a full set of human motion based on a short history and goal pose. Their environment only consists of static obstacles and has a fixed goal set. As we focus on incorporating social elements in the path prediction process, our environment is more dynamic, while our kinematic structures are much simpler.

### B. MADER: Trajectory Planner in Multi-Agent and Dynamic Environments

The paper by Lei *et al.* [1] describes MADER, a 3D-decentralized, asynchronous trajectory planner designed for collision-free navigation of unmanned aerial vehicles (UAVs) in an environment with static and dynamic obstacles. This was accomplished by creating a planning mechanism for mobile robots using double deep Q-network (DDQN) reinforcement learning to navigate through an unknown environment. The Q-learning algorithm is beneficial as it improves the robot's ability to navigate through dynamic environments using local path planning. This strategy allows for generalization of the dynamic path planning problem from doing reinforcement learning DDQN on LIDAR sensor information. Additionally, a training method and reward function was developed to address the problem of non-converging algorithms due to the reward sparsity of the state space. Utilizing their work, we used a similar DDQN for the agent's trajectory.

### C. PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs

The paper by Wang *et al.* [10] expands on the problem of image generation of consecutive frame. To build upon Lon Short-Term Memory (LSTM), spatiotemporal memory needs to be included in order to accommodate for the significance of spatial deformations and temporal dynamics. The idea is that in order to predict the future, it is helpful to memorize as many historical details as possible. To accomplish this, the authors created an architecture called Predictive RNN (PredRNN), allows memory states of different LSTMs to interact across the different layers. This is achieved by the creation of the Spatiotempral LSTM (ST-LSTM) unit. It is able to model spatial and temporal representation of a unified memory cell and share the memory vertically across layers and horizontally over the states. It is a general and modular framework, which can be used in many applications. Inspired by this work, we used a similar PredRNN structure to forecast social behavior and predict how agents will navigate the environment.

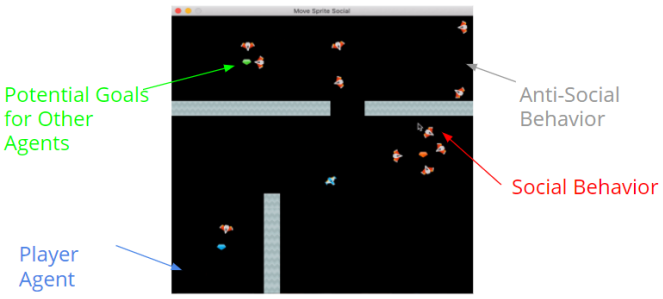


Fig. 1. Social game GUI

## III. Methods

### A. Environment Setup

We created a virtual environment using Python and PyArcade. The environment can produce a graphical user interface (GUI) and allows users to operate their own agents. As depicted in Figure 1, each previous player's path is represented by the orange planes and should be treated as obstacles to avoid by the player agent, which is represented as a blue plane.

Each time a human player enters the game, a randomized goal is generated. The player then navigate to their goal without colliding with any static or dynamic obstacles to the best of their ability. The player can produce two kinds of input: a rotational velocity and a linear velocity. Trajectories of previous actors are recorded and played back for each new incoming player. As the number of players increase, the playground become more clustered and harder to navigate through.

To simulate social elements in this environment, we included goals that require social interpretation, in addition to the resources collecting goal. The potential goals are: {go to blue gem, go to green gem, go to red gem, isolate, and socialize}

Aside from the dynamic obstacles, three rectangular obstacles divide the environment into three rooms with two narrow passageways. This further helps create traffic in the environment as players often need to cross through the narrow passage to reach their goal.

### B. Environment Formulations

In order to define the environment mathematically, the following formulations were made:

*1) State Space:* The state space is described as:

$$S_{agent} = \{(p_x, p_y, p_\theta), \tag{1}$$

$$\forall p_x \in [0, 600], p_y \in [0, 600], p_\theta \in [-180, 180)\} \tag{2}$$

where $p_x$ and $p_y$ are the x-y coordinates of the agent in the 2D continuous space, $p_\theta$ is the agent's forward facing angle.

*2) Observation Space:* The observation space is described as:

$$O = \{(p_{a_x}, p_{a_y}), \tag{3}$$

$$\forall p_{a_x} \in [0, 600], p_{a_y} \in [0, 600]\} \tag{4}$$

where $p_a$ is a collection of the positions of the actors (other agents acting as dynamic obstacles) with respect to the agent.

*3) Action Space:* The action space is described as:

$$A_{Agent} = \{v_p \in [-2, 2], v_r \in [-2, 2]\} \tag{5}$$

where $v_p$ is the linear velocity of the agent, $v_r$ is the rotational velocity of the agent.

*4) System Dynamics:* The system dynamics of the agent moving through the environment is modelled by the following formulation

$$s_{t+1} = \begin{bmatrix} p_{x_{t+1}} \\ p_{y_{t+1}} \\ p_{\theta_{t+1}} \end{bmatrix} = \begin{bmatrix} p_{x_t} + v_p cos(p_{\theta_t})dt \\ p_{y_t} + v_p sin(p_{\theta_t})dt \\ p_{\theta_t} + v_r dt \end{bmatrix} = f(s_t, a_t) \tag{6}$$

where $[s_{t+1}, s_t] \in S$, $(v_p, v_r) = a \in A$, $dt = 1/60$s.

Additionally, the system dynamics are deterministic. Thus, for each state-action pair ($s_t \in S, a_t \in A$), the subsequent state $s_{t+1} \in S$ that satisfies the above dynamic equation will have a probability of 1, otherwise 0. To put it in mathematical form:

$$p(s, a, s') = \begin{cases} 1 & if \quad s' = f(s, a) \\ 0 & o.w. \end{cases} \tag{7}$$

$$\forall (s, a)\ s.t.\ s \in S, a \in A \tag{8}$$

### C. RNN

As our goal is to encode social interactions, which differs greatly from the work of Marc and others [3], our input to the network is richer in altruistic information while less rich in kinematic information. For each time step, we encode the state and action of the agent in a $6x1$ vector, followed by observation of obstacles. We initially defined observation as all of the obstacle's states. It was discovered that such method hinders learning and the network fails to capture the behavior of obstacles. We experimented with relational observation and were able to achieve much better predictions. To summarize, the input to the network, $X$ is the history of agent trajectory and observations over a short time period $H$:

$$x_{agent}(t) = \begin{bmatrix} p_x(t) \\ p_y(t) \\ sin(p_\theta(t)) \\ cos(p_\theta(t)) \\ v_p(t) \\ v_r(t) \end{bmatrix}, x_{obs_i}(t) = \begin{bmatrix} p_{obs_x}(t) - p_x(t) \\ p_{obs_y}(t) - p_y(t) \end{bmatrix} \tag{9}$$

$$x(t) = \begin{bmatrix} x_{agent}(t) \\ x_{obs_1}(t) \\ \vdots \\ x_{obs_n}(t) \end{bmatrix}, X = \begin{bmatrix} x(t-H), & x(t-H+1), \ldots, x(t) \end{bmatrix} \tag{10}$$

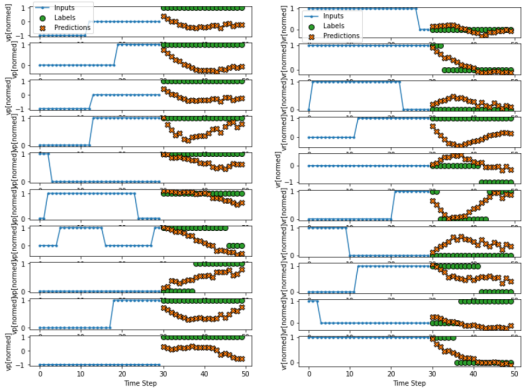


Fig. 2. RNN action prediction over a obstacle reaching for the green gem. On the left is the prediction for $v_p$ on 10 different time step. On the right is the prediction for $v_r$ for 10 different time steps

It is worth noting that we encoded the front facing angle $p_\theta(t)$ as two values to avoid sudden transition from 0 and $2\pi$ when the agent makes a full rotation.

$$Y = \begin{bmatrix} v_p(t+1), & v_p(t+2), \ldots, & v_p(t+T) \\ v_r(t+1), & v_r(t+2), \ldots, & v_r(t+T) \end{bmatrix} \tag{11}$$

The output $Y$ of our RNN model represents future actions the agent would perform over a short time horizon $T$. The labels used for training are the recorded action histories of users. The initial implementation of the RNN model was to train a single neural network using data from one scenario/room and perform on a new and unknown environment. After a few trials, we discovered that this approach does not generalize well given that each room contains players with unique combinations of tasks. Whichever room the network is trained with, the model will try to learn the pattern of the specific combinations of tasks the players in that room are doing. Models trained with this method perform well when applied to agents in the training room, but fail to succeed in other rooms when the neural network tries to force the learned information upon obstacles doing tasks in a different order. To address this issue, we changed our RNN pipeline to be multiple networks, with each one dedicated to a specific goal that the obstacles try to achieve. Although this contradicts our design proposal of estimating obstacles goals and assumes that the agent is fully aware of each player's goal, this approach is still sufficient to provide evidence that RNNs can successfully improve the performance of path planning by predicting trajectories of dynamic obstacles.

We evaluated the performance of the RNNs and extracted a sample result which is shown in Figure 2. After given the information of the obstacles' goal, which is "Green Gem" in this case, and the past information of the obstacles, our neural network outputs their predicted future actions labeled as orange crosses in the figure. From Figure 2, we can see that the network successfully predicts the obstacles' future action inputs for most of the time. The structures and hyperparameters of the networks can be further tuned to reach a

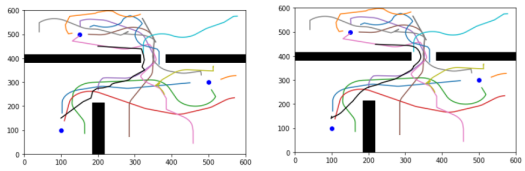

Fig. 3. Final trajectory results for start Point = (100, 150), and goal Point = (150, 500). Each colored line indicates an obstacle's trajectory, the black line indicates the robot's trajectory. On the left is the global path produced by $RRT^*$ after 2000 iteration. On the right is the MPC generated path following the waypoints produced by $RRT^*$

higher accuracy but given the time constraint of the class, we will feed the outputs from these networks into our traditional path planning. In the next section, we will show with results that the RNNs are capable of improving the performance and robustness of the agent's path planning.

### D. Path Planning

We use RRT* for global path planning and for local path planning. RRT is a popular and reliable sampling based path planning method that can quickly produce a valid trajectory. The RRT* derivation can produce an optimal path given enough time. Sampling based methods generally do not take dynamics constraints into consideration and can only produce waypoints. We use model predictive control to plan locally within a short time horizon. We used an off-the-shelf implementation of RRT*. The global planner does not take dynamics obstacles into consideration. Instead, for MPC we formulated a cost function tailored to the goal of this project. The cost of the optimization problem consists of: cost for reaching the intermediate waypoint, cost of collision with static obstacles, and cost of collision with dynamic obstacles. The goal of this model is to find a series of input and future states that minimize these costs.

$$\min_{U,X} \int_{t=0}^{T} \{J_x(x(t), x_{ref}(t)) + J_{cs}(x(t)) + J_{cd}(x(t), x_a\hat{o}bs(t))\} \tag{12}$$

$$\text{subject to: } \dot{x} = f(x, u) \tag{13}$$

$$u(t) \in \mathbb{U} \tag{14}$$

$$x(0) = x(t_0) \tag{15}$$

The costs for dynamic and static collisions are:

$$J_{cd}(x(t), \hat{x_a}(t)) = Q_c/(1 + e^{\kappa*(||x(t)-\hat{x_a}(t)||-2r_{robot})}) \tag{16}$$

$$J_{cs}(x(t))) = Q_c/(1 + e^{\kappa*(d-2r_{robot})}) \tag{17}$$

where $Q_c$ and $\kappa$ are tunable parameters that adjust the weight of collision detection, and $x_a\hat{o}bs$ is the predicted trajectory generated using the predicted action from RNN. To evaluate the performance of such combination, alternation to this setup is introduced as the baseline model, where we assume the input velocity is constant throughout the entire period.

In practice, we use a time horizon $T = 5$ , $Q_c = 5$ and $\kappa = 1$ for MPC.

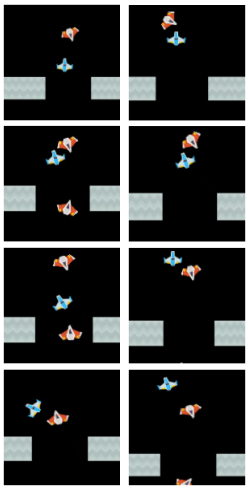

Fig. 4. Different robot behavior with different obstacle prediction model. On the left hand side is the behavior using only observed input, starting from the top we can see the robot (blue) comes close to another obstacle and violently recoiled backwards, which almost caused another collision. On the right hand side we that the robot was able to successfully maneuver around the orange obstacle. For the full videos, please see supplemental materials

As seen in Figure 3, MPC is able to increase the smoothness of the trajectory provided by the sampling based method. The framework has no issues navigating through the environment when obstacles are static. However, it's behavior when dynamic obstacles are present are more apparent in real time. As seen in Figure 4, the robot act more natural and robust when using RNN as obstacle prediction.

### E. Deep Q-Learning

In order to implement DQN in the system, each of the previous environmental formulation spaces were modified. Under DQN, the system is decentralized and the agent is no longer aware of the movements made by other actors in the environment. This will inherently lead to a different definition of the state and observation space.

To begin, the action space was discretized into 4 possible states: up, down, left, and right. A continuous and infinitely large set of actions, as previously prescribed, is unlikely to converge efficiently. The state space and observation space were adjusted to match the updated action space and account for a time metric in the environment. By using these new formulations, the agent attains a higher likelihood to converge on a potential viable trajectory with DQN.

*1) Modified State Space:*

$$S_{agent} = \{(p_x, p_y, p_m), \tag{18}$$

$$\forall p_x \in [0, 600], p_y \in [0, 600], p_m \in [0, 50000)\} \tag{19}$$

where $p_x$ and $p_y$ are the x-y coordinates of the agent in the 2D space of all real numbers, $p_m$ is the agent's time spent in motion. $p_m$ can also be recognized as the total number of movements made by the agent in the system.

The modified observation space is equivalent to the modified state space described. As mentioned previously, the agent will

not have knowledge regarding the positions of other actors and will only be aware of its position within the environment. Noticeably, the state space consists of $18 \times 10^9$ possible states where each x-y coordinate could have a different reward state depending on the current status in time. Consequently, traditional Q-Learning would be extremely inefficient and the usage of neural networks via DQN would be the more appropriate approach.

*2) Modified Action Space:*

$$A_{Agent} = \begin{cases} Up & \delta_y = +1 \\ Down & \delta_y = -1 \\ Right & \delta_x = +1 \\ Left & \delta_x = -1 \end{cases} \tag{20}$$

*3) Modified System Dynamics:*

$$s_{t+1} = \begin{bmatrix} p_{x_{t+1}} \\ p_{y_{t+1}} \\ p_{mt+1} \end{bmatrix} = \begin{bmatrix} p_{x_t} + \delta_x \\ p_{y_t} + \delta_y \\ p_{mt} + 1 \end{bmatrix} = f(s_t, a_t) \tag{21}$$

*4) Rewards System:* To create a suitable environment for the agent to explore and learn, a viable rewards system had to be formulated. In the initial trials, the rewards were set to reflect the setup described by J. Zhang [11]. While this reward system allows the agent to explore every state and is more likely to converge, the approach is time-consuming. Therefore, a similar rewards system was adopted with the inclusion of a penalty associated with the distance between the agent and their goal. Therefore, the agent would be aware of what general area it should be looking towards and plan a path accordingly. The reward system was defined as the following:

$$R_{Agent} = \begin{cases} -0.25 & p(x,y) = o(x,y) \\ M & p(x,y) = g(x,y) \\ -0.04 - 0.1d(p,g) & \begin{cases} p(x,y) \neq g(x,y) \\ p(x,y) \neq o(x,y) \end{cases} \\ -0.4 & p(x,y) \in P(x,y) \end{cases} \tag{22}$$

where $p(x,y)$ is the current x-y coordinates of the agent, $o(x,y)$ is the current x-y coordinates of an obstacle (includes other actors, bounds of the room, and restricted areas), $g(x,y)$ is the x-y coordinates of the goal, $P(x,y)$ is the list of previously visited states, $d(p,g)$ is the distance between the goal and the agent, and $M$ is the maximum reward. The maximum reward was often defined as the maximum number of allowable time steps for observational purposes. The negative reward of $P(x,y)$ was implemented to prevent the agent from exploring previously visited portions of the map. In initial trials, the agent noticeably converged towards small areas within the room. These areas were characterized by light traffic which would be preferable to the negative reward obtained when encountering obstacles. Consequently, these models would always diverge, demonstrating the importance of including a negative reward moving into previously visited spaces.

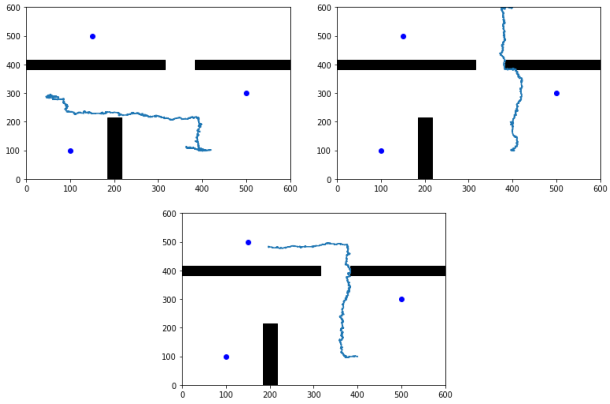

Fig. 5. Deep Q-Learning Process: Start Point = (400, 100), Goal Point = (150, 500), 15000 time steps, 100 episodes

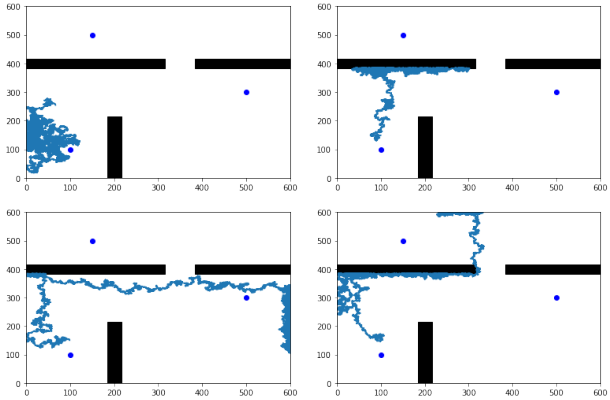

Fig. 6. Deep Q-Learning Process: Start Point = (100, 150), Goal Point = (150, 500), 50000 time steps, 100 episodes

*5) Training:* The number of neurons in the first and second layers of the neural network correlate to the dimensions of the room (600x600). The number of neurons in the third layer correlate to the maximum number of time steps allowed in the environment. The network was compiled with an Adam optimizer and a mean-squared error loss function. These parameters were modelled from other DQN systems described in literature. Due to project constraints, the network was run with small batch sizes ranging from 10 to 20 samples, but with total time steps ranging from 10,000 to 50,000. Consequently, the temporary memory space for these samples were set to a high value to simulate the learning process. As a frame of reference [11], the complete learning process is described by a batch size of about 40,000 samples, 1,000's of epochs, and 10,000's of episodes. Therefore, a fully completed neural network was not trained in the span of our project, but strong results from the achievable number of episodes demonstrate the underlying success behind a fully implemented DQN network. More details on the network structure can be found in Table A.1.

*6) Observations:* In total, approximately 2000 episodes (5 total sets of different parameters) were performed via DQN.

These 5 total sets included changes to the maximal number of time steps, the goal points, the starting points, and the batch sizes. The sets were used as a method of fine tuning and the differences between their outputs were insignificant. Also due to the relatively small sample size of these non-converged sets of trials, loss and other statistics can be considered uninteresting. From a completely observational point of view however, the algorithm could be seen to demonstrate strong growth in the early stages of DQN. As can be seen in Figure 5 and 6, the agent is able to follow a trajectory that is within close proximity of the desired goal state defined by (150,500). It should also be observed that the maximum number of allowable time steps were increased with more difficult trajectories. Although these plots are a small sample, the agent is seen to be gradually learning the best path to take as it is given the time to learn and explore the environment. Additionally, a key point is that these plots do not demonstrate a start, middle, and end representation of the learning process. The timeline of learning produces a variety of these plots from start to finish, but as the agent continues to learn, the less valuable trajectories are slowly filtered out. Noticeably in each set, the agent was able to reach its goal in approximately 3% of its total number of episodes. This low success rate can be taken lightly however due to the nature of the earlier stages of training. This learning trend was seen in each of the 5 sets of parameters and provides a baseline to the DQN approach on the agent's ability to navigate without collision. Despite not producing a fully converged model, the results of these episodes demonstrate the potential of implementing reinforcement learning to path plan in a dynamic environment.

## IV. Conclusion

In this paper we examined approaches to path planning in a cluttered environment with dynamic obstacles. We combined different traditional methods and showed improvement over baseline path planning approaches. Although there are still details needed to be tweaked and algorithms that can be further optimized, our overall architecture and its performance show promising possible extensions and applications of this technology.

### A. Future Direction

This technology could be used in several different applications such as self driving cars, autonomous cleaning robots, and NPC in video games. It can be applied to situations where spacial navigation is cluttered with static and predictable dynamic obstacles. The main contribution of this work is to provide a framework that can incorporate social interaction of outside agents into a path planning problem. Our next steps would be to provide a more explainable learning structure to encode social interactions. As of now we have a different model for each goal, in the future we wish to distinguish an agent's goal using their trajectories, which would greatly help generalize our framework. To achieve this, a larger training database is required and the addition of different neural network models (e.g. CNN) may help to further decode the temporal-spatial relations of the data.

The DQN algorithm was seen to be another potential solution to the dynamic path planning problem. Unfortunately, due to time constraints, a fully converged neural network was not feasible. Based upon the attained results, the potential of DQN within this dynamic problem is highly encouraging. This model of the DQN should be able to learn and explore the environment at a reasonable rate. This would include increasing the batch size, the number of episodes, and the maximum number of allowable time steps. These improved parameters would surely help the agent learn more about its environment in a more efficient manner. Other iterations of DQN could also be explored such as the usage of sparse rewards. By randomly placing relevant rewards, the agent is given more information on how to proceed in the dynamic environment and has the potential to outperform the vanilla DQN. Therefore, viable future directions of this model could be to improve the current iteration of DQN and also explore the use of additional features.

## References


[1] J. Tordesillas and J. P. How, "MADER: Trajectory Planner in Multi-Agent and Dynamic Environments." [Online]. Available: https://arxiv.org/pdf/2010.11061.pdf.

[2] J. Martinez, M. J. Black, and J. Romero, "On Human Motion Prediction Using Recurrent Neural Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[3] P. Kratzer, M. Toussaint, and J. Mainprice, "Prediction of Human Full-Body Movements with Motion Optimization and Recurrent Neural Networks," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020.

[4] Y. Tang, L. Ma, W. Liu, and W.-S. Zheng, "Long-Term Human Motion Prediction by Modeling Motion Context and Enhancing Motion Dynamics," Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, 2018.

[5] X. Li, S. Liu, K. Kim, X. Wang, M.-H. Yang, and J. Kautz, "Putting Humans in a Scene: Learning Affordance in 3D Indoor Environments," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[6] A. Jati and P. Georgiou, "An Unsupervised Neural Prediction Framework for Learning Speaker Embeddings Using Recurrent Neural Networks," Interspeech 2018, 2018.

[7] T. Catfolis, "Generating adaptive models of dynamic systems with recurrent neural networks," Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94).

[8] Y. Zhou, "Automatic generation of fuzzy neural networks via reinforcement learning with applications in path planning of mobile robots."

[9] P. Rodriguez, J. Wiles, and J. L. Elman, "A Recurrent Neural Network that Learns to Count," Connection Science, vol. 11, no. 1, pp. 5–40, 1999.

[10] Y. Wang, Y. Fang, P. Lou, J. Yan, and N. Liu, "Deep Reinforcement Learning based Path Planning for Mobile Robot in Unknown Environment," Journal of Physics: Conference Series, vol. 1576, p. 012009, 2020.

[11] J. Zhang, "Path Planning for a Mobile Robot in Unknown Dynamic Environments Using Integrated Environment Representation and Reinforcement Learning," 2019 Australian amp; New Zealand Control Conference (ANZCC), 2019.

[12] "Deep Reinforcement Learning for Maze Solving¶," qmaze. [Online]. Available: https://www.samyzaf.com/ML/rl/qmaze.html. [Accessed: 19-Dec-2020].

[13] Brendan Martin, Satwik Kansal, "Reinforcement Q-Learning from Scratch in Python with OpenAI Gym," [Online]. Available: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/. [Accessed: 19-Dec-2020].

## APPENDIX

### *B. Network Structures*

| Deep Q-Learning Neural Network Structure | |
|---|---|
| Feature | Value |
| First Layer | 3 Inputs; 600 neurons |
| Second Layer | 600 neurons |
| Third Layer | 800 neurons |
| Output Layer | 4 Outputs |
| Activation Functions | RELU |
| Discount Factor $\gamma$ | 0.9 |
| Exploration Decay | 0.999 |
| Learning Rate | 0.001 |

TABLE A.1
FEATURES OF RNNS

| Recurrent Neural Network Structure | |
|---|---|
| Feature | Value |
| GRU | 30x28 Inputs; 18048 neurons |
| GRU | 24960 neurons |
| Dense Layer | 2600 neurons |
| Output Layer | 20 Outputs |
| Activation Functions | RELU |

TABLE A.2
FEATURES OF RECURRENT NEURAL NETWORKS