2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)
Würzburg, Germany, September 2-4, 2019

Sample Efficient Reinforcement Learning for Navigation in Complex Environments

Barzin Moridian[1], Brian R. Page[2], and Nina Mahmoudian[3]

***Abstract*— Navigation of mobile robots in unstructured, time-varying environments is challenging. It becomes even more complicated in disaster scenarios where logistical difficulties, as well as technical issues such as reactive and time-varying obstacles, exist. These scenarios are too complex for classical obstacle avoidance methods to navigate through successfully. This paper presents a sample efficient reinforcement learning algorithm for navigation in complex environments. The approach augments training data with randomly generated target location data to accelerate learning. A Q-learning approach is implemented, which is capable of quick training with limited episodes. The procedure is tested in four scenarios in Gazebo and one scenario in a real-world experiment. In the two simulation scenarios with no obstacles, the method can learn to navigate towards the target in fewer than 200 episodes. For environments with moving obstacles, training takes slightly longer, but the process is still able to learn an effective policy quickly.**

***Index Terms*— Navigation in complex and/or extreme environments; Deep Learning; Q-Learning; Data Augmentation; Disaster Robotics**

I. INTRODUCTION

The operation of mobile robotic platforms in disaster scenarios is difficult in terms of both logistics as well as technical challenges. This is because every disaster is unique and often generates complex and/or extreme operating environments. Navigation of autonomous mobile robots in these unstructured environments is commonly addressed by Simultaneous Localization and Mapping (SLAM). SLAM methods build a map [1], [2], [3], [4] of the environment using onboard sensors such as lidar, stereo vision, or temporal matching of a monocular camera [5]. The map is then used to find feasible trajectories from robot's location to the desired location, using global path planners such as Dijkstra or A* [6]. The robot then uses the overall planned trajectory and the current sensor readings to make immediate action decisions and avoid collisions. The whole stack of algorithms involved in SLAM needs to reliably recognize obstacles and their geometries in order to generate global trajectories to avoid collisions. These trajectories become more complex to calculate as the area of operation grows larger and more cluttered [7], [8].

*This material is based upon work supported by Office of Naval Research N00014-16-1-2422 and National Science Foundation 1453886, 1921060.

[1]At the time of conducting this study Barzin Moridian was a Senior Research Associate of Mechanical Engineering, Purdue University, West Lafayette, IN, USA. barzinm@gmail.com

[2]Brian R. Page is a Graduate Research Assistant of Mechanical Engineering, Purdue University, West Lafayette, IN, USA page82@purdue.edu,

[3]Nina Mahmoudian is an Associate Professor of Mechanical Engineering, Purdue University, West Lafayette, IN, USA ninam@purdue.edu.

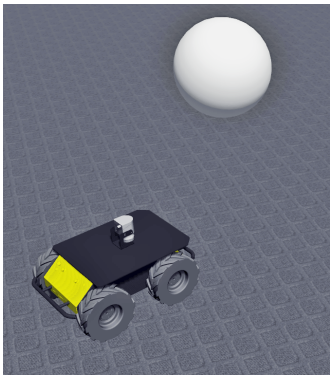


Fig. 1: A mobile robot equipped with lidar learning to navigate in an environment with moving obstacles.

The information provided by such perception does not provide sufficient information for decision-making in challenging environments. These environments contain obstacles of different shapes, sizes, and textures. The environment changes are unforeseeable and not suitable for modeling possible motion patterns and inclusion into a static map. Ultimately, the environment might be interactive and react based on the movement of the robot. As an example, navigation in crowded areas classical algorithms might determine that reaching the desired location is not feasible due to detecting moving obstructions in all directions. To solve these types of unique challenges, hand-tuned methods could be used, however the algorithm quickly becomes technically impractical to maintain. Therefore, to have mobile robots autonomously navigate in unstructured, disaster environments, other plausible approaches should be studied.

An alternative approach for robotic perception and decision-making is data-driven methods. In this category, Reinforcement Learning (RL) algorithms can build decision-making models that improve their performance as they accumulate more interaction experiences [9], [10]. The capabilities of RL algorithms have been extended using Deep Network structures to perceive high-dimensional and feature-rich sensor inputs, such as cameras and lidars, resulting in Deep RL [11], [12], [13]. Deep RL has already been applied to a number of robotic sensorimotor skill learning [14], [15], [16], in addition to planning tasks [17], [18], [19], [20]. Such learning algorithms benefit from exposure to diverse environments and tasks, in contrast to classical methods. These diversities improve the generalization capability of the model and boost overall performance [21], [22].

This work applies reinforcement learning to the problem of mobile robot navigation in complex environments. The

978-1-7281-0778-3/19/$31.00 ©2019 IEEE

primary contribution of this work is the development of a sample efficient reinforcement learning navigation algorithm through the use of a data augmentation approach. This sample efficient algorithm and model are analyzed through application to two simplified scenarios 1) waypoint navigation without the presence of obstacles and 2) avoiding collision with moving obstacles (Fig. 1). Preliminary lab experiments are also presented as the first attempt at real-world validation.

The paper presents a study of related works addressing decision-making for autonomous agents in Sec. II. Formulation of the proposed method and a brief background is presented in Sec. III. Description of experimental settings, implementation details, results, and in-depth analysis of performance are provided in Sec. IV. Finally, Sec. V lists aspects of the presented method that can be further studied in future works.

II. Related Work

Machine learning approaches are already outperforming hand-engineered methods in multiple aspects of autonomous navigation. For example, supervised learning methods have significantly improved the perception of the environment. These methods can reliably detect distant outdoor objects and traversable areas [23]. Supervised learning and imitation learning have also been used for motion planning of autonomous mobile robots. Without the need to design explicit feature extraction tools and decision-making rules, these methods can learn to navigate by imitating a human expert's decisions in challenging environments, such as forest trails [24], cluttered natural environments [25], 3D racing games [26], sidewalks [27], or indoor environments [28]. Alternatively, these methods can learn from other control systems, such as model predictive control [29] or imitating the A* navigation algorithm applied to a graph representation of street-view images [30]. While these methods rely on expert demonstrations for gathering training samples, the approach in this paper does not require the learning agent to be provided with correct actions. It can continue to improve the decision-making policy from its own experiences. As a result, the training period of this approach is not limited to the presence and skills of a human expert to present correct actions.

Reinforcement learning methods build models that learn by observing the environment and performing actions that maximize the expected sum of future rewards [9], [10]. Reinforcement learning methods have had great success in simulation environments such as Atari games, outperforming humans in many cases [11], [31], [32], [12]. Applying these methods to the low-level control tasks of real robotic platforms such as learning sensorimotor skills is challenged due to their need for a large number of experiences. There are a number of efforts focusing on reducing required training sample complexity, such as pre-training an agent in a simulation environment [28], [33], [34], guiding the policy to explore a narrow space of states [35], or gathering initial experiences from expert demonstrations [25]. To the best of authors' knowledge, there has not been a work focusing on providing a sample efficient learning approach for collision-free navigation of mobile robots in dynamic environments.

In addition to immediate motion planning, RL methods are also used for far-horizon path planning resulting in explicit planning without requiring a model of the environment [17]. Since the planning policies are relatively abstract, they enjoy better generalization and can be transferred between environments, for example, from simulation to real-world settings [28]. The planning policies can also learn to reason based on common sense patterns, leading to better generalization [18]. Similar to SLAM methods, these works assume a static environment. On the other hand, navigation in a dynamic environment requires a planning method that focuses on understanding the immediate motions occurring in the environment, which is considered in this work.

III. Background and Method

The problem is formulated as a Markov Decision Process (MDP). In each state $s \in S$, the agent performs an action $a \in A$ following the policy $\pi(s)$. The agent then receives a reward r and reaches a new state, which might be a terminal state that marks the end of the episode. The goal is to find a policy π that provides actions to maximize the expected sum of discounted future rewards. A model-free reinforcement learning technique, known as Q learning[36], builds an estimation of the expected sum of rewards $Q(s, a)$ for each action in different states. It is then possible to use a greedy policy at deployment time that performs the action with highest action value $a_\pi = \text{argmax}_a\, Q(s, a)$.

Training of a model that approximates the action-value function $Q(s, a)$ is achieved by storing the agent's experiences with the environment $e_t = (s_t, a_t, r_t, s_{t+1})$ in a replay buffer. In Q-learning, the value of each state action pair is calculated from

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a), \tag{1}$$

where $Q(s, \cdot)$ is equal to zero if s is a terminal state. The value of γ controls the preference of future over immediate rewards. A deep network parameterized by θ can learn to estimate the action values $Q(s, a; \theta)$ by finding parameters of the network that minimizes J_Q in

$$J_Q(\theta) = (r_t + \gamma \max_a(Q(s_{t+1}, a; \theta)) - Q(s_t, a_t; \theta))^2 \tag{2}$$

For deep networks, parameters θ are found by stochastic gradient descent or its variants [11]. To further overcome the challenges of training deep networks for Q learning, the parameters of the target network $Q(s_{t+1}, a')$ are frozen for c intervals [31]. After each c training steps, the θ parameters are copied over to $\bar{\theta}$ parameters of the target network $Q(s_{t+1}, a'; \bar{\theta})$. Additionally, the action a' is decided using the parameters θ. Overall, the more applicable form of Eqn. 2 is:

$$\begin{aligned} a' &= \text{argmax}_a\, Q(s_{t+1}, a; \theta_t) \\ J_{DD}(\theta) &= \big(r_t + \gamma Q(s_{t+1}, a'; \bar{\theta}_t) - Q(s_t, a_t; \theta)\big)^2 \end{aligned} \tag{3}$$

Although the model in Eqn. 3 uses environment state s for estimation, in some environments the full state is not available at each time step. In such environments, the agent can take advantage of remembering information from previous observations in addition to the current one and use temporal relations between observations to infer underlying states. For example, speed and moving direction of an object can be inferred from two consecutive images. A well-studied deep network mechanism for carrying this information through time is the Long Short-Term Memory (LSTM) recurrent module [37]. This gated module can learn to add, overwrite, and discard information in its memory. The forward propagation of this module for input x_t can be represented as

$$[h_t, c_t] = f_{\text{LSTM}}(x_t, h_{t-1}, c_{t-1}), \tag{4}$$

where $c_t, h_t \in \mathbb{R}^n$ are respectively the internal state of LSTM memory cell and its output.

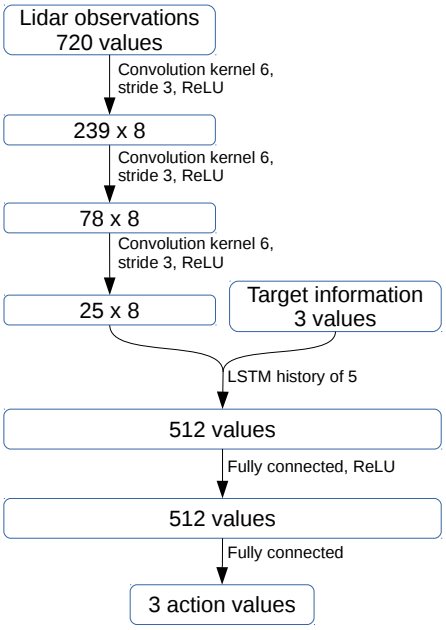


Fig. 2: Model used for navigation in dynamic environments.

To train a model for navigation of mobile robots, we configured the deep network's inputs and outputs to match the nature of the problem. The information used for decision-making comes from a lidar sensor, and the robots pose relative to the target. The robot's pose relative to the target is the distance to target d_g and the angle between the robot's heading and the direction toward the target ϕ. The deep network inputs are formed as a tuple $(d_g, sin(\phi), cos(\phi))$, in which the use of sin and cos provide better consistency in all cases compared to directly using ϕ, without masking vital information. The action space of the agent is configured to consist of three discrete actions of moving forward or turning left/right.

The structure of the model (Fig. 2) used for this task is comprised of three convolution layers, stacked one after another, that encode lidar readings. Each convolution layer has 8 channels, a kernel size of 6, strides of 3, and ReLU (Rectified Linear Unit) activation. The encoded lidar readings and target values then go through a LSTM unit with a dimension of 512. The LSTM output is fed to a fully connected layer with a dimension of 512 and ReLU activation and then another fully connected layer with three outputs and no activation function. The outputs of the last layer are estimations of the expected sum of future rewards for each action under current observations (model inputs).

At each time step, the location and orientation of the robot and the action performed are stored in the experience replay buffer[38]. At training time, sample experiences holding sequences of the previous k lidar readings and robot poses, action performed, and the resulting lidar reading and robot pose are drawn from the buffer. For each sample in a training iteration, a randomly generated target location is assigned to augment the collected data. Based on robot pose and assigned target, model inputs $(d_g, sin(\phi), cos(\phi))$ of this augmented training iteration are formed. This method makes it possible to train the model using many more interaction samples compared to the ones directly experienced during interaction with the environment. The data augmentation method enables our method to train faster than using only raw data. The sequence of k observations helps the model to infer temporal relations and gain information such as the velocity of objects and the momentum of the robotic vehicle.

To have a wide variety of states and actions explored, an ϵ-greedy policy chooses the agent's actions while collecting interaction samples for training. The ϵ-greedy policy chooses the action with the highest estimated action value with the probability of $1-\epsilon$. Otherwise, random actions are taken for five steps. Action values are estimated by providing a First-In, First-Out (FIFO) buffer with a maximum length of k as model input, updated after each step of training episode with the most current reading and discarding the oldest one. Using longer sequences lets the agent learn temporal relations that expand over longer horizons. Also, for a sufficiently large sequence, it is possible to carry (h_t, c_t) from Eqn. 4 instead of using the FIFO buffer.

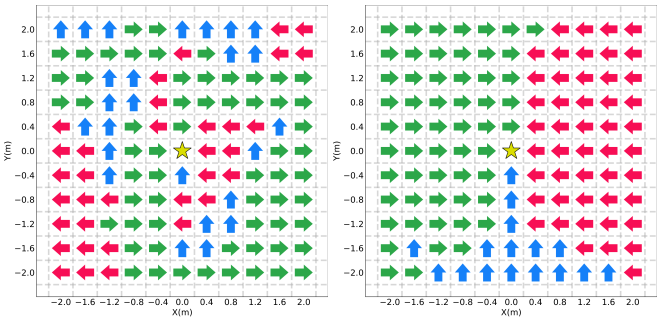
To reward the robot based on reaching the goal or colliding with objects, a robot space with radius l is defined around the robot. The agent receives a reward of +1 for a distance of less than l between the robot and target. Additionally, a reward of -1 is received if the lidar observes an obstacle closer than l. Both cases result in the termination of the episode. In all other cases, the agent is rewarded as $r = c_1 \min(0, d_l - 2l) - c_2 \min(d_g, c_3)$ where d_l is the closest lidar reading and $c_{1,2,3}$ are user-defined constants.

IV. Experiments

The experiments are conducted in the Gazebo robot simulation environment (Fig. 4). At the beginning of each episode, the robot, obstacles, and target are randomly positioned in a 10×10 m^2 area. Walls surround the outside of the environment with few small openings. A skid-steer vehicle is used as the robot platform equipped with a lidar which makes 720 readings equally spaced in $[0, 2\pi]$ up to 20m in distance.

Lidar values are divided by 20. There is a 0.2 second delay between consecutive lidar and location observations. The vehicle has a maximum forward velocity of 0.5 m/s and a maximum turn rate of 0.5 radian/s. A simplified obstacle field is created with spherical objects moving in randomly initialized directions with a velocity randomly sampled from a normal distribution with a mean value of 1 m/s, a standard deviation of 0.3 m/s, and truncated by 0.5m/s and 1.5m/s. This obstacle field is meant as a simplified approximation of the real world with future work needed to experiment with more complex obstacles such as reactive and time-varying obstacles.

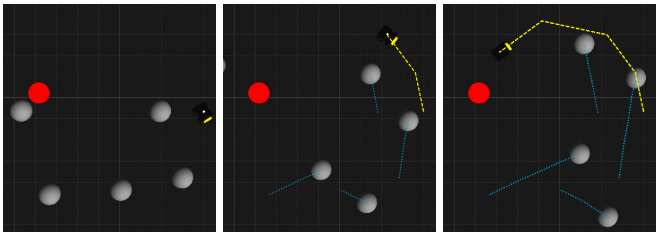
Parameters of the network are updated using Adam optimizer [39] with a learning rate of $1e$-4 and default values for other optimizer-specific parameters. The network parameters are copied over the settings of the target network every $c = 10000$ training steps. Both training and execution of the policy use a sequence of $k = 5$ observations. At the beginning of training, the random action probability is set to $\epsilon = 0.7$ and is exponentially decayed with a rate of 0.98 after each episode. Other parameters have been set as $\gamma = 0.97$, $c_1 = 0.01$, $c_2 = 0.002$, $c_3 = 10$,



(a) After 10,000 training steps. (b) After 100,000 training steps.

Fig. 3: Policy illustration early and late during training. The goal is illustrated by a star and the robot is facing toward positive Y in each cell.

We first investigate how the policy changes during training with no obstacles. Fig. 3 presents the agent's decisions in the state space, early and late during the training process. Each cell in these figures represents a state where the robot is placed in that cell facing upward (positive y), and the target is located at the center of the grid. The three types of arrows in these figures represent the three actions of moving forward or rotating left/right. For example, if the robot is placed at $x = 0, y = -0.4$, the trained policy prescribes moving forward (blue arrow) and following that policy, the robot gets closer to the origin. It should be noted that the left or right actions mean that the robot will rotate in place, which is not the same as translating to left or right cells. For example, after turning left at $x = 2.0, y = -1.6$, the next state is closer to $x = 1.6, y = -2.0$. The early policy (Fig. 3a) shows different regions in the state with seemingly random actions. Through more training, the agent divides the state space into three regions for the three available action (Fig. 3b). One can observe regions in the state space in which the agent chooses inconsistent action (e.g. $-1.6, -1.6$). Such inconsistencies can be the result of factors such as the initial random values for network parameters (i.e. weights and biases), insufficient number of experience samples and those regions, or simply the need for more training steps and lower learning rate.



(a) (b) (c)

Fig. 4: Example motion of different objects in the environment. The mobile robot is moving towards the target (red circle) while avoiding collision with moving obstacles. The approximate trajectory of obstacles and the vehicle are illustrated using cyan and yellow dashed lines respectively.

For simulation experiments, each agent is trained for 600 episodes. An episode lasts up to 200 steps or until a termination condition is met. At the end of each episode, except for the first five, the agent is trained for 1000 iterations with a batch size of 256 samples. At test time, each trained model is tested with $\epsilon = 0$ for 200 episodes that can last up to 500 steps. Test results indicate the fraction of episodes finished successfully by reaching the goal, the fraction of failed episodes as a result of a collision, and the fraction of episodes not finished during the 500 steps. Each number is the average of the training of five agents with different random parameter initialization seeds.

TABLE I: Test results with five moving obstacles.

	Success	Collision	Unfinished
Random Agent	0.047	0.953	0.0
Feedforward	0.464±0.104	0.462±0.094	0.074±0.019
Feedforward (best)	0.575	0.34	0.085
Recurrent	0.872±0.034	0.128±0.034	0.0
Recurrent (best)	0.895	0.105	0.0

The agent is first tested in an environment with five moving obstacles (Fig. 4). To study the influence of memory in mobile navigation tasks, a similar agent without LSTM was also trained and tested in this environment. The results of both the trained recurrent agent with LSTM and the feedforward agent are presented in Table I and during training in Fig. 5. Results show that the agent with memory was much more successful in reaching its goal. During experiments, the feedforward agent made more conservative decisions and kept more distance from obstacles. This behaviour can be due to lacking the capacity to understand the motion direction and speed of obstacles.

We then performed similar experiments for both agents in an environment without obstacles (Table II). Fig. 6 shows how well the recurrent model converges during training,

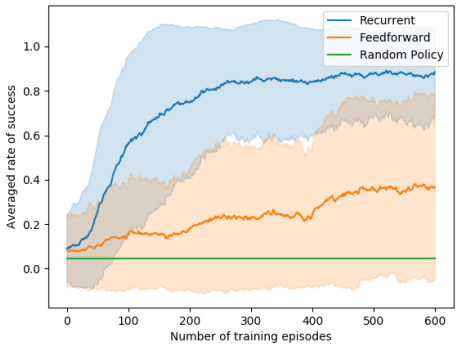


Fig. 5: Training with five moving obstacles.

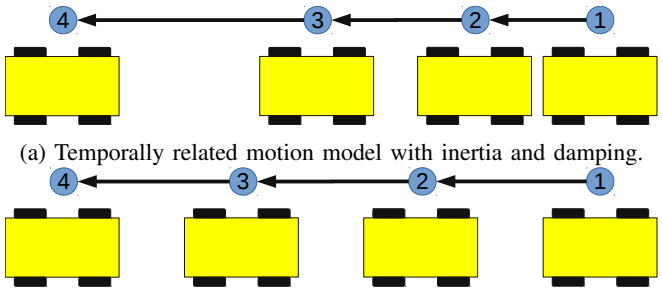
TABLE II: Test results with only walls as obstacles.

	Success	Collision	Unfinished
Random Agent	0.13	0.869	0.001
Feedforward	0.628±0.222	0.057±0.02	0.315±0.213
Feedforward (best)	0.9	0.025	0.075
Recurrent	1.0	0.0	0.0
Recurrent (best)	1.0	0.0	0.0

while the feedforward model struggles. The results show that the recurrent agent performs flawlessly in the environment, while the feedforward agent only reaches the target 63% of the time. Instead, the number of episodes that were not terminated by the feedforward agent was significantly increased. In general, the dominating behaviour in unfinished episodes is the agent oscillating between turning right and turning left, staying in place for a long time. The oscillating behaviour comes from policy inconsistencies that make the agent loop between a set of states without making progress. We interpret that the reason the recurrent model is robust against such inconsistencies is that its input is a sequence of observations. As a result, the first time the robot is in a specific pose gives different inputs to the model than the second time it experiences the same pose.

Next, we studied scenarios similar to the two presented above but with a fundamental change to vehicle motion dynamics. In the first two scenarios, the vehicle was modeled with inertia and damping and moved by applying force and torque to the body, similar to Fig. 7a. The next two scenarios study navigation learning for a mobile robot without inertia and damping. In these cases, each robot action translates the body in constant increments. Fig. 7b illustrates the difference. With Fig. 7b dynamics, similar to forward movement, turning left or right also rotates the body in constant increments. In this setting, the motion of the robot can be accurately anticipated from current observation and without the need to remember a history of poses. Specifically, forward action moves the vehicle 0.2 meters forward and turn actions rotate the body 0.2 radians. These values were set arbitrarily and are assumed as characteristics of the world. Understandably, the success rate of the problem depends on vehicle dynamics, as arbitrarily small displacement values would make it difficult for the agent to dodge or maneuver around obstacles and very large values would introduce undesirable overshoot for each action.

Fig. 6: Training with only walls as obstacles.



(a) Temporally related motion model with inertia and damping.

(b) Constant displacement vehicle motion model.

Fig. 7: Illustration of the different vehicle motion models used for training and testing.

The performance of both trained recurrent and feedforward agents in the constant displacement environment without obstacles is presented in Table III and during learning in Fig. 8. The feedforward model performs significantly better in this case compared to when the vehicle has an underlying temporally related dynamics. We used this setting to take a closer look at the unfinished episodes by the feedforward agent. In addition to the tests of feedforward agent presented in Table III, we tested adding a 10% probability of performing a random action during each time step of test episodes. The addition of random actions helped the robot to break free from the oscillation deadlocks, dropping the unfinished episode to 0.006±0.012 with success and collision rates rising to 0.925±0.096 and 0.069±0.085, respectively. This test confirmed that the unfinished episodes are the results of policy inconsistencies in some state neighborhoods and not usually due to the robot wandering around in the environment without reaching the goal.

The last simulation environment setting tested is the constant displacement of vehicle in addition to presence of 5 moving obstacles (Table IV and Fig. 9). Both models were less successful in this scenario compared to Table I. This is partially due to the design of this test scenario in which agents have lower speed compared to the maximum speed in case of vehicle dynamics with inertia and damping. As a result, dodging obstacles and maneuvering in between them is more challenging.

To study the generalization capability of the agents, we trained them with one vehicle dynamic and tested with another while avoiding moving obstacles. An agent was trained

TABLE III: Test results of constant displacement vehicle dynamics and only walls as obstacles.

	Success	Collision	Unfinished
Random Agent	0.103	0.889	0.008
Feedforward	0.813±0.231	0.003±0.004	0.184±0.227
Feedforward (best)	0.99	0.0	0.01
Recurrent	.995±0.007	0.0	0.005±.007
Recurrent (best)	1.0	0.0	0.0

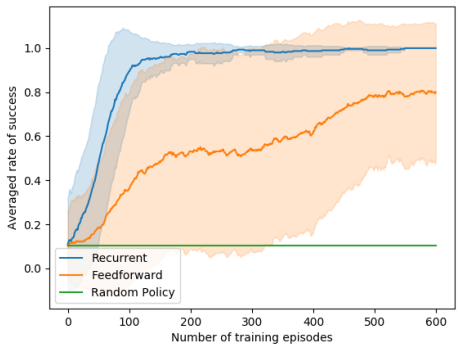


Fig. 8: Training with constant displacement vehicle and only walls as obstacles.

with constant displacement vehicle dynamics and tested on a vehicle with inertia and damping that is the same as the environment tested for Table I. This agent had 0.765±0.064 success, 0.229±0.064 collision, and 0.006±0.007 unfinished episode ratios (average of five seeds). This performance was significantly less compared to when both testing and training were subjected to vehicle inertia and damping. Agents trained with inertia and damping and tested with constant displacement had 0.743±0.019 success, 0.251±0.021 collision, and 0.006±0.005 unfinished episode ratios. Surprisingly, this performance is similar to the results of Table II, suggesting better generalization capability when trained in a more challenging setting that requires understanding of underlying temporal relations.

As an early stage, real-world experimental validation of the RL algorithm, we have implemented a simplified version on a mobile robot. The robot used for this experiment is a toy vehicle, to which an Odroid XU4 was added. The onboard computer communicates with a motion capture camera system in the lab to gain information on position and orientation of the vehicle. The deep network architecture was modified and the convolution layers were removed since the car did not have a lidar sensor. The agent's task was to learn to control the motion of the vehicle for reaching target locations. The onboard computer also communicates with a desktop computer that is responsible for running the deep learning libraries for both training and generating model outputs. Fig. 10 presents the robot's motion from different initial positions to a target location. We have since added a lidar sensor and full experimental validation of the algorithm is underway.

V. Discussion and Future Work

This paper presented our method of applying deep reinforcement learning for mobile robot motion control and

TABLE IV: Test results of constant displacement vehicle dynamics and five moving obstacles.

	Success	Collision	Unfinished
Random Agent	0.064	0.936	0.0
Feedforward	0.292±0.204	0.39±0.102	0.318±0.121
Feedforward (best)	0.54	0.28	0.18
Recurrent	0.735±0.048	0.257±0.048	0.008±0.013
Recurrent (best)	0.78	0.22	0.0

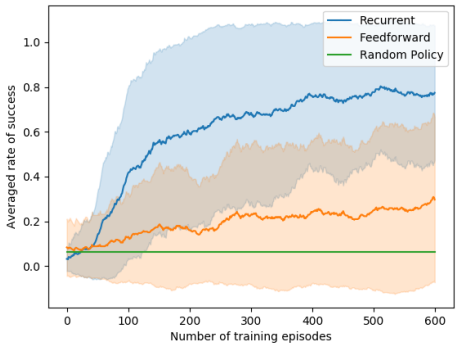


Fig. 9: Training with constant displacement vehicle dynamics and five moving obstacles.

Fig. 10: Trajectories of the robot from various initial locations, moving toward the goal (star).

collision avoidance with moving obstacles. This is applicable to mobile robots in the general case, but could be particularly useful in disaster scenarios where environments are too complex and/or extreme for classical navigation methods. We propose a sample efficient navigation learning approach to facilitate real-world implementations through data augmentation.

In the presence of moving obstacles, our deep network models were able to plan trajectories based on the learned expected motion of obstacles in the environment to avoid collision with dynamic obstacles. The model was also capable of learning and taking advantage of the motion model of our robotic vehicle for maneuvering in the environment. In environments without obstacles, we were able to train deep networks in less than 200 episodes, about an hour of simulation and 20,000 training iterations, to navigate to nearby points with a near perfect success rate. We also identified and analyzed different attributes of this method in section IV.

In the future, we would like to expand the result to study the effect of using only sparse rewards at the termination state and the use of Prioritized Experience Replay. Training mod-

els with continuous action space and fine tuning using policy gradient methods is another direction of exploration. We will also deploy the suggested method on a physical platform that we have built, equipped with a camera, 3D scanning lidar, and IMU/GPS to further explore the challenges and solutions of navigation learning in real-world complex environments.

REFERENCES

[1] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical slam: Real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

[2] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[3] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[4] S. Thrun *et al.*, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, vol. 1, pp. 1–35, 2002.

[5] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[6] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[7] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[8] S. Huang and G. Dissanayake, "A critique of current developments in simultaneous localization and mapping," *International Journal of Advanced Robotic Systems*, vol. 13, no. 5, p. 1729881416669482, 2016.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.

[10] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] 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*, 2016, pp. 1928–1937.

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[14] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[15] 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.

[16] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[17] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.

[18] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2616–2625.

[19] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3357–3364.

[20] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[21] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[22] R. Rahmatizadeh, P. Abolghasemi, L. Blni, and S. Levine, "Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 3758–3765.

[23] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *Journal of Field Robotics*, vol. 26, no. 2, pp. 120–144, 2009.

[24] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[25] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1765–1772.

[26] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning." in *AISTATS*, 2011, p. 6.

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

[28] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3357–3364.

[29] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, "Plato: Policy learning using adaptive trajectory optimization," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3342–3349.

[30] S. Brahmbhatt and J. Hays, "Deepnav: Learning to navigate large cities," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[31] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning." in *AAAI*, 2016, pp. 2094–2100.

[32] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[33] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5055–5065.

[34] 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)*, Sep. 2017, pp. 31–36.

[35] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1–9.

[36] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

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

[39] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.