Adaptive Robotic Path Planning via Obstacle Trajectory-Guided Reinforcement Learning

Ali Nafih Pullani[1], Frank Ortmeier[2]

Abstract— **Path planning for high degrees-of-freedom robots in a collaborative workspace requires real-time adaptability to continuous changes in the environment. Intelligent path adaptations that account for the motion characteristics and attributes of objects in the robot's workspace are essential for a closer human-robot collaboration. This paper presents a reinforcement learning (RL)-based path planning approach that identifies and intelligently adapts the robot's path to obstacles based on their motion patterns of varying dimensionality - ranging from one-dimensional linear to three-dimensional helical obstacle trajectories. The experiments indicate that the RL algorithm learned to proactively redirect the robot trajectories to regions of reduced collision risk based on the obstacle motion, resulting in higher success rates than conventional planners, such as artificial potential fields. The planner was tested against repeated high-speed (linear speeds up to 2.5 m/s and angular velocities up to 8π rad/s) path obstructions by dynamic obstacles with noisy, perturbed trajectories. The results highlight the adaptive potential of RL-based path planning for next-generation cobot applications in human-robot collaboration.**

I. Introduction

The recent shift towards Industry 5.0, which emphasizes a closer collaboration between humans and robots [1], has introduced additional complexity to the path planning process by transforming the robot's workspace into an uncontrolled dynamic environment. The presence of humans and other dynamic elements leads to persistent interruptions of the preplanned robotic trajectories. Consequently, the development of path planners that can evolve and adapt the robot's trajectory to rapid real-time changes in the environment has become necessary. Such path planners must accurately identify the obstacle motion patterns, account for its trajectories, and safely maneuver around them dynamically, demonstrating *intelligent* adaptive behavior.

Path planning methods used by manipulators can be broadly classified into two categories: classical and learning-based approaches. The classical methods include combinatorial, sampling-based, potential-based, and bio-inspired planning, whereas learning-based methods encapsulate machine learning (ML) paradigms such as deep learning (DL) and RL [2]. Recent scientific breakthroughs in ML have established learning-based path planning- particularly RL- as an *intelligent* alternative to conventional methods, especially for high Degrees of Freedom (DOF) robotic systems operating in dynamic environments due to their adaptability and uncertainty handling capability [2]–[4]. However, current RL implementations with articulated robots have several limitations (refer to Section III). Many existing studies on RL-based path planning account for a single (or similar) motion profile, requiring path corrections that follow a predetermined pattern. Moreover, the scenarios are often simplistic, with the obstacles obstructing the target path momentarily at lower speeds.

[1]Ali Nafih Pullani and [2]Frank Ortmeier are with Chair of Software and Systems Engineering, Otto-von-Guericke University, Magdeburg, Germany, (e-mails: `ali.pullani@ovgu.de`, `frank.ortmeier@ovgu.de`)

In this study, an RL-based path planner that adapts robot trajectories according to the motion type and attributes of the obstacle is proposed. The proposed approach is capable of identifying and avoiding obstacles with different motion profiles of varying dimensionality: one-dimensional (linear motion with different velocity profiles), two-dimensional (circular and elliptical trajectories with varying angular velocities), and three-dimensional (helical trajectories with varying angular velocities). The RL algorithm learns a long-term strategy that dynamically guides the robot trajectories along paths with reduced collision risk. Experimental results demonstrate that the joint space velocity control strategy learned by RL is capable of performing *intelligent* path corrections (corrections that proactively reduce collision risk in the long-term) in the presence of highly dynamic obstacles (with linear speeds up to 2.5 m/s and angular velocities up to 8π rad/s) that repeatedly obstruct the robot's trajectory. The constant target obstructions and high velocities simulated in the experiment serve as a stress test for the path planning strategy, demonstrating its reliability in dynamic environments.

II. Background

A. Reinforcement Learning

1) A brief introduction to RL framework: For a discrete-time finite RL problem, at each step t, the agent (learner) observes a representation of the environment state (S_t) and interacts with it by executing an action ($\mathrm{A_t}$), causing a transition into a new state (S_{t+1}) with a scalar reward (R_{t+1}) at the next timestep, as shown in Figure 1. The process repeats until a terminal condition is reached, marking the end of an episode. The agent's objective is to learn a policy π (mapping from observed states to actions) that maximizes the expected cumulative reward over time [5]. The user-defined reward functions guide the learning process by reinforcing (or penalizing) actions based on the quality of resulting state transitions, assigning positive feedback to those that help the agent toward its goal, and vice versa.

2) Soft Actor-Critic (SAC) algorithm: Unlike standard RL, which maximizes the expected sum of rewards, the SAC maximizes a *soft* expectation of rewards augmented by the

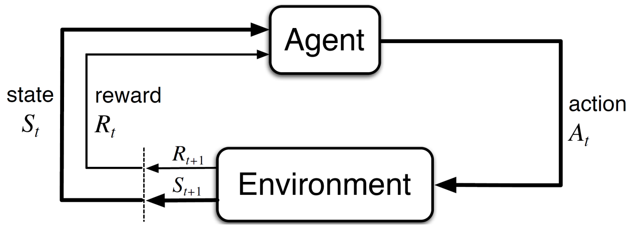


Fig. 1: Reinforcement learning framework [5].

expected entropy of the policy, improving exploration and preventing premature convergence into suboptimal solutions [6]. Figure 2 shows the pseudo-code for SAC implementation with state-value(V), state-action-value (Q), and policy(π) functions parameterized by neural networks ψ, θ, and ϕ, respectively. $\mathcal{D}$ denotes the replay buffer, whereas ∇J and λ represent the gradient of the objective function and the learning rate, respectively. A detailed explanation of the code can be found in [7].

Algorithm 1 Soft Actor-Critic

```
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
for each iteration do
    for each environment step do
        $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
        $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
    end for
    for each gradient step do
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
        $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
    end for
end for
```

Fig. 2: Soft Actor-Critic Algorithm [7].

B. Artificial Potential Field (APF) method

The APF method uses user-defined virtual potential fields around the goal (attractive field, as given in Equation (1)) and the obstacles (repulsive field, as given in Equation (2)) for path planning. The attractive potential draws the robot configuration (q) towards goal configuration (q_{goal}), while the repulsive potential pushes it away from the obstacle (q_{obs}) within a radius of influence (d_0). APF models robot motion as a gradient descend (i.e., following the vector $-\nabla(U_{\text{att}} + U_{\text{rep}})$) in the combined field. For cartesian space implementation, the attractive and repulsive forces calculated in the cartesian space are converted to joint space velocities (or torques) to generate robot actuator commands.

$$U_{att}(q) = \frac{1}{2}k_{att}\|q - q_{goal}\|^2 \tag{1}$$

$$U_{rep}(q) = \begin{cases} \frac{1}{2}k_{rep}\left(\frac{1}{\|q - q_{obs}\|} - \frac{1}{d_0}\right)^2, & \text{if } \|q - q_{obs}\| \le d_0, \\ 0, & \text{if } \|q - q_{obs}\| > d_0. \end{cases} \tag{2}$$

III. Related Works

Multiple RL methods, algorithms, network architectures, and reward structures have been explored for robotic path planning in recent research [2]–[4], [8]. Some of the recent works in RL-based path planning with manipulators are briefly summarized below. In [9], Deep Deterministic Policy Gradient (DDPG) algorithm is combined with Hindsight Experience Replay (HER) to avoid obstacles that dynamically appear (but, stationary after appearance) in the trajectory of the robot. In [10], Twin Delayed Deep Deterministic (TD3) algorithm is combined with a potential field for path planning around spherical obstacles. However, it uses a simplified one-dimensional obstacle motion with constant velocity. [11] uses Soft Actor-Critic (SAC) with HER and a potential field inspired reward structure to avoid obstacles moving at the robot in a straight line with a velocity between 0.1-0.3 m/s. Similarly, [12] uses SAC with Prioritized Experience Replay (PER) to avoid an obstacle moving randomly across the target, however, with a single overlap between the robot and obstacle trajectories. In [13], Deep Q-Network (DQN) was combined with a guiding neural network to avoid objects moving along a conveyor belt between the robot and the target. [14] uses a gating mechanism with SAC and DDPG to avoid obstacles moving on a plane.

The methods described above primarily handle obstacles following a fixed trajectory type that momentarily obstruct the target at relatively low speeds, but do not address trajectories of varying dimensions and attributes. This study addresses the limitation by incorporating obstacle trajectories into the training phase of RL planner. In the context of Industry 5.0, the next generation of path planners must be capable of handling complex, persistent obstacle interactions at higher speeds.

IV. Methodology

A. Path Planning using RL

The RL framework can be used for manipulator path planning, wherein the agent observes the state of the robot and its environment to generate joint actuation commands, as shown in Figure 3. Our experiments used a task-specific reward function (refer to Section IV-B) and a control frequency of 100 Hz. For the experiments, the SAC algorithm was chosen as it has been shown to (generally) outperform other RL algorithms for continuous control tasks, including OpenAI gym benchmarks [6].

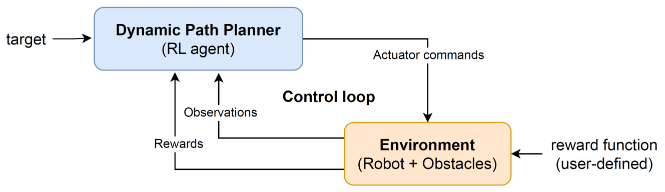


Fig. 3: Path planning with RL agent.

B. *Experimental Setup*

1) Simulation Environment: Universal Robot's UR5e, obstacles, and scenes were simulated in MuJoCo [15] for the experiments. MuJoCo was chosen because of its speed and parallelization capability. The simulated robot was wrapped in the standard *gymnasium* structure for RL experiments. *Stable-baselines3* implementation [16] of SAC was used for experiments on an Intel i7 processor (CPU-only) with 10 parallel vectorized environments for accelerated sample generation and training.

During an episode, the agent attempts to reach a target point in the workspace (goal) without colliding with a dynamic obstacle modeled as a sphere, which represents the bounding spherical volume of the obstacle in the robot's workspace. The distance from the robot body to the obstacle (measured using reference points on the robot shown in Figure 4) and the robot state were the observations fed into the neural networks of SAC for decision making. Tracking multiple reference points on the robot body helps the RL agent's decision making by enabling it to learn distinct adaptive actuations for different robot links based on their proximity to the obstacle. In our experiment, the actuation signals are the robot's six joint velocities (actions). To improve the robot's response to immediate changes in its surroundings, the joint velocity (lowest control level via standard Universal Robot interfaces) was used instead of joint position.

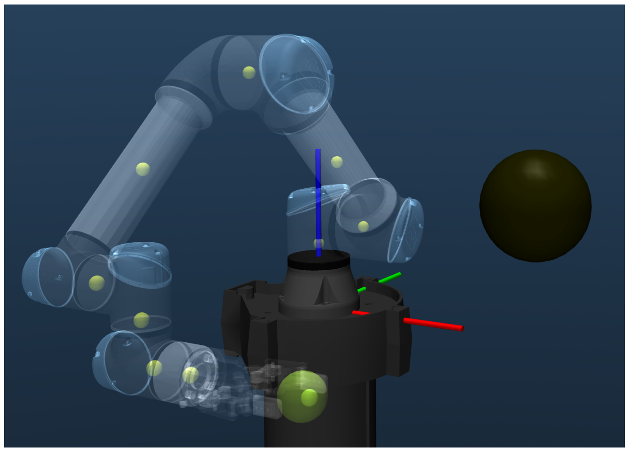


Fig. 4: Reference points on the robot.

2) RL components: The observation space of the RL agent comprises of the following: current joint angles, end effector position, distance to the obstacle from the reference points (Figure 4), and dimension of bounding sphere- a combined observation dimension of 20. The action space comprises of angular velocities of six UR5e joints. The reward structure used for the experiments, consisting of continuous and sparse components, is described by Equation (3). The continuous reward component was introduced to guide the agent's behavior incrementally, as using a sparse reward (rewards only at the end of an episode) generally leads to slow convergence because of its infrequent occurrence. Overall, the RL agent gets a continuous reward based on its distance to the target (incentivizing movement towards the target) and a terminal reward based on the termination condition (incentivizing goal achievement). The parameters α, β, γ, δ, and ϵ determine the relative weights of reward components. A virtual boundary, violation of which leads to a penalty weighted by ϵ, was placed on the robot during training to discourage exploration into unpromising state space regions. The weights of the reward components were experimentally determined.

$$R = \begin{cases} \alpha(1-d)^{2n+1}, & \text{after each step,} \\ \beta(T-t)+\gamma, & \text{if target achieved,} \\ -\delta(T-t), & \text{if collided,} \\ -\epsilon(T-t), & \text{if boundary violated.} \end{cases} \tag{3}$$

where,

T	is the maximum number of steps in an episode,
t	is the current step number (elapsed steps),
d	is the distance between end-effector and target,
α	is the proximity reward weight,
β, γ	are the success bonus weights,
δ, ϵ	are the adjustable penalty weights,
n	is a parameter (integer) to control reward scaling.

3) Network architecture: Two fully connected deep neural networks with two hidden layers of 512 and 256 neurons were used as actors and critics for policy and value estimation, respectively. A twin critic Q-learning approach (which uses the minimum of Q-values predicted by two separately instantiated identical networks) was used to reduce overestimation bias and improve learning stability [7]. Each Q-network (and its target counterpart) takes 26-dimensional input (combination of observation and action) and outputs a single Q-value, whereas the policy network takes in 20-dimensional observation and outputs six action values (joint velocity commands).

4) Dataset generation: The dataset consisted of five obstacle motion types (linear with constant velocity, linear with triangular velocity profile, circular, elliptical, and helical motion) in 1D, 2D, and 3D motion domains. The trajectories for all the motion types were generated using two key spatial points: the target point (that the robot is trying to reach) and the end effector point at the beginning of path planning. These two points define the direction or the plane of the cyclical obstacle motion. A third point is sampled from the workspace to establish the local coordinate system at the target for trajectory generation of 2D and 3D profiles. Specifically, the obstacle either exhibits linear to-and-fro movement along the direction defined from the target point towards the end effector (for 1D motion) or rotational movement around the plane/volume defined by the three points (for 2D and 3D motion). For instance, the helical trajectory is generated as a combination of oscillatory (sinusoidal) motion along the screw axis and a circular motion along the plane containing the other two basis vectors in the local coordinates, which is then transformed into the global coordinates. Figure 5

TABLE I: Overview of motion types and parameters

Dimensions	Motion Type	Trajectory Dimensions (l/r/h)	Motion Attributes (v/a)
1D	Linear – Constant Velocity	$l = 0.25$–0.4	$v = 0.5$–1.5
	Linear – Triangular Profile	$l = 0.25$–0.4	$a = 5.0$–15.0
2D	Circular	$r = 0.15$	$v = \pi$–8π
	Elliptical	$r_1 = 0.12,\ r_2 = 0.17$	$v = \pi$–8π
3D	Helical	$r = 0.15,\ h = 0.1$	$v = \pi$–8π

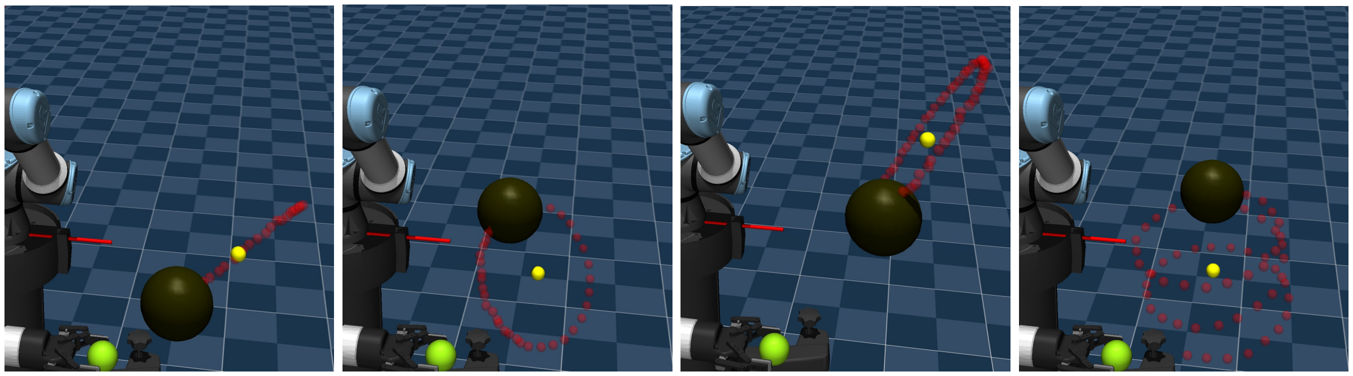


Fig. 5: Instances from training dataset (from left to right): (1) Linear, (2) Circular, (3) Elliptical, and (4) Helical trajectories. Key scene elements (colored spheres) are: obstacle (black), its trajectory (red), end effector (green), and target (yellow).

depicts examples of different motion types from the experimental dataset, whereas Table I describes the variations in the parameters (length(l), radius(r), height(h), linear/angular velocity(v), and acceleration(a)) for each motion type. All the parameters in the table are in SI units.

C. Training and Validation

The control loop runs for a maximum of 15 seconds at a frequency of 100 Hz, producing a maximum of 1500 timesteps during an episode. During each episode, an instance of obstacle trajectory (cf. Table I) was executed by the obstacle across/around a target point sampled from the interaction zone ($x \in (0.25, 0.85)$, $y \in (-0.7, 0.0)$, and $z \in (-0.1, 0.7)$ m). During training, the SAC agent collected experiences from the environments (rollouts) to improve its value estimation and policy (action choices) with high entropy until the average cumulative reward converged to a satisfactory success rate on the training dataset. Agent's performance during training was tracked using three key metrics: (1) average cumulative reward (average sum of rewards collected by the agent during an episode), (2) average success rate (proportion/percentage of successful runs over total runs), and (3) average path length (number of timesteps the agent required to complete an episode). These performance indicators were aggregated over a predefined log interval (50 recent episodes in our implementation) to estimate the agent's current performance.

During the evaluation, the planner was stochastically tested with obstacle trajectories of unseen attributes from the range defined in Table I. The performance of the RL agent was primarily compared against the APF planner with second-order potential fields (cf. Equations (1) and (2)), as it can instantaneously plan based on the system's state, similar to the RL planners. Alternatively, the Rapidly-exploring Random Tree (RRT) was considered for comparison, but was excluded due to its inadequate performance at high obstacle speeds. Unlike the RL planner, the parameters of the APF planner were optimized separately for each test dataset using a guided search with Tree-structured Parzen Estimator (TPE) from *Optuna* [17]. The implementations discussed in Section III, which are intended for a single motion profile, are also excluded from comparison, as they cannot be directly applied to the combined dataset.

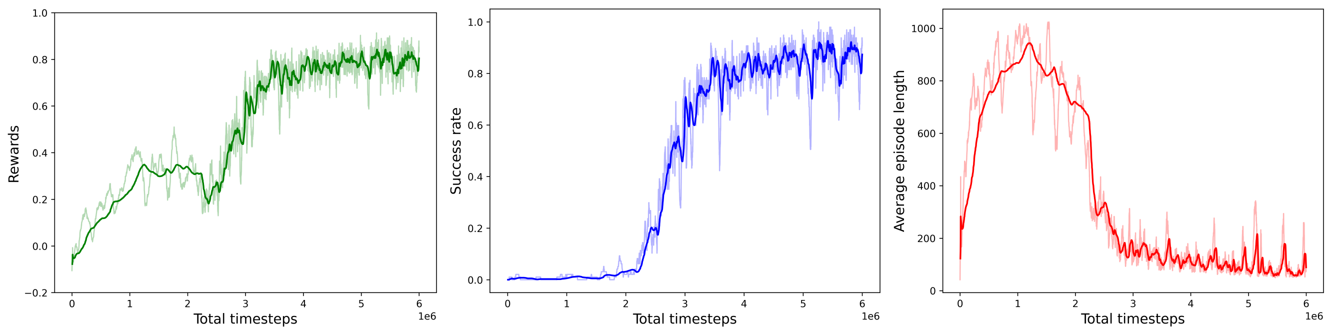
V. Experiments, Results and Discussion

The primary objective of the experiment was to learn obstacle avoidance and path planning strategies for close proximity interaction between the robot and a highly dynamic obstacle. The RL agent was trained on the combined dataset (which includes 1D, 2D and 3D) for obstacle motions. During the training and evaluation process, the target points, obstacle motion type, trajectory orientation and parameters were varied for each episode to simulate dynamic obstacle behaviors. Such a dynamic obstacle behaviour ensures that the robot's path planner can not preplan a trajectory to the target, as the direct path from the end effector position to the target point is persistently interrupted, forcing the planner to continuously replan and adapt its trajectory. Additionally, a zero-mean Gaussian noise was added to simulate real-world inaccuracies in sensors and actuators.

During the training, the rewards collected by the agent and task success rate increased, whereas the average path

TABLE II: Performance comparison for different motion types

	1-D Dataset	**2-D Dataset**	**3-D Dataset**	**Combined Dataset**
Artificial Potential Field				
Success Rate (%)	57.10 ± 3.28	71.70 ± 2.58	74.10 ± 4.12	57.50 ± 2.37
Collision Rate (%)	39.90 ± 3.18	28.30 ± 2.58	25.90 ± 4.12	42.20 ± 2.70
Failure Rate (%)	**3.00 ± 1.41**	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.67 ± 0.30**
Reinforcement Learning (ours)				
Success Rate (%)	**75.50 ± 1.96**	**96.30 ± 1.06**	**86.80 ± 2.25**	**84.70 ± 2.41**
Collision Rate (%)	**19.70 ± 2.49**	**3.60 ± 1.17**	**12.90 ± 2.47**	**13.40 ± 2.76**
Failure Rate (%)	4.70 ± 1.16	0.10 ± 0.32	0.30 ± 0.67	1.90 ± 1.45



(a) Rewards (b) Success Rate (c) Episode Length

Fig. 6: Graphs showing (a) cumulative rewards, (b) success rate, and (c) episode length over the training duration.

length reduced, indicating a positive learning of the task (refer to Figure 6). The performance comparison between the RL and APF planners on validation dataset, given in Table II, shows that the RL planner trained solely on the combined dataset consistently outperformed the APF planner fine-tuned for individual test cases. The RL planner handled the combination of trajectories of different dimensionality, whereas the APF struggled even with optimized parameters. The collision rates were significantly reduced by the RL planner, while the failure rates remained similar. Note that the mean failure rate is lower than its standard deviation (in certain cases) due to near-zero values from the experiment seeds. Both planners show relatively lower performance for 1D dataset compared to the 2D and 3D datasets, since the short trajectory length across the target reduces the available approach time and increases the collision chances.

The experiments demonstrate the advantage of RL path planners over conventional planners, especially for handling complex obstacle trajectories at higher speeds. The improved performance achieved by the RL path planner can be attributed to its ability to learn long-term strategies that increase (expected) future rewards. Figure 7 shows a qualitative comparison between paths planned by APF (top row) and RL (bottom row) for 1D, 2D, and 3D obstacle motion (columns 1-3, from left to right, respectively). In the figure, the curves marked by red and green spheres denote the trajectory of the obstacle and the robot's end effector, respectively, whereas the yellow sphere marks the target point. For 1D and 2D obstacle motion, APF produces an oscillatory path to the target, often along the line/plane of the obstacle motion which tends to collide at higher speeds. On the other hand, the RL agent redirects the robot's trajectory away from the line/plane of the obstacle motion, reducing the number of collisions. For the helical obstacle trajectories, the RL agent tends to recognize the motion type and approach the target along the screw axis while the obstacle is moving away (cf. Figure 7). Overall, the RL path planner demonstrated the ability to learn *intelligent* adaptation based on obstacle motion characteristics, dynamically guiding the end effector trajectories along paths with reduced collision risk.

VI. Limitations and Future Work

Despite the advantages of the proposed RL-based path planning approach, the use of deep neural networks introduces limitations such as computational overheads and a lack of interpretability. Another limiting factor is the sample inefficiency of the model-free RL approach. In this regard, incorporating prior baseline strategies and using model-based RL approaches to improve sample efficiency is an interesting avenue for future investigations. Another promising research direction is the use of temporal neural networks (such as Recurrent Neural Networks(RNNs) [18] or Long Short-Term Memory networks(LSTMs) [19]) or transformer networks with attention [20] to extend the proposed approach to complex obstacle geometries and multiple obstacle interactions.

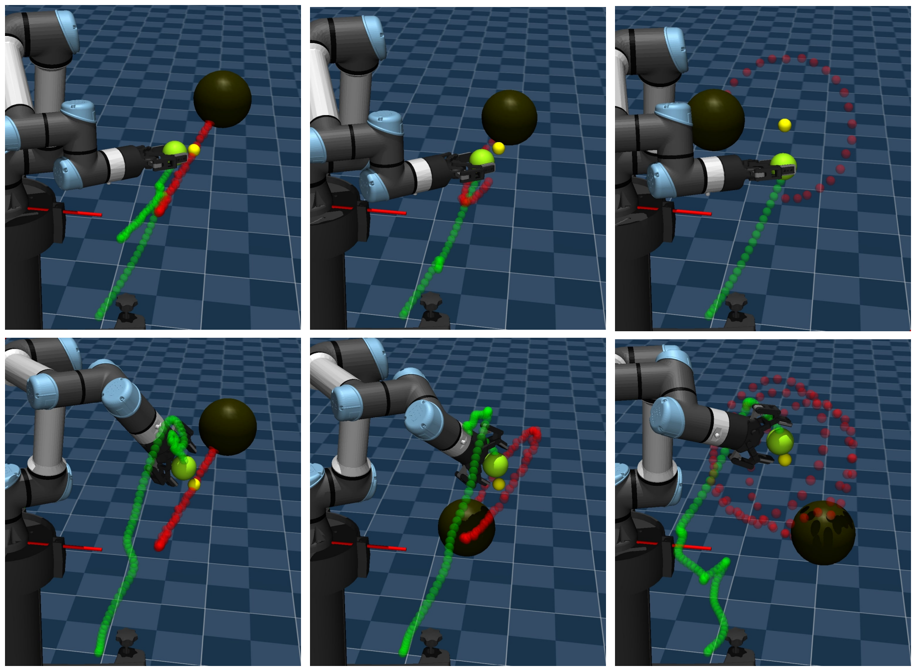


Fig. 7: Path-planning comparison: APF (top row) versus RL (bottom row) for 1D, 2D, and 3D (left to right) trajectories.

VII. Conclusion

In this study, an RL-based path planner that adapts the robot's trajectory based on the motion characteristics of obstacles in the robot's workspace is proposed. The experimental validation confirmed the proposed approach's adaptability on obstacle trajectories of varying dimensionality with repeated high-speed path obstructions. Furthermore, additional obstacle trajectory perturbations were introduced to test the effectiveness of the RL path planner. Thus, this work demonstrates the capability of RL path planners to recognize and adapt to diverse motion profiles, contributing towards the development of *intelligent* path correction required for collaborative manufacturing.

References

[1] P. K. R. Maddikunta, Q.-V. Pham, B. Prabadevi, N. Deepa, K. Dev, T. R. Gadekallu, R. Ruby, and M. Liyanage, "Industry 5.0: A survey on enabling technologies and potential applications," *Journal of industrial information integration*, vol. 26, p. 100257, 2022.

[2] M. G. Tamizi, M. Yaghoubi, and H. Najjaran, "A review of recent trend in motion planning of industrial robots," *International Journal of Intelligent Robotics and Applications*, vol. 7, no. 2, pp. 253–274, 2023.

[3] Y. Zhang, W. Zhao, J. Wang, and Y. Yuan, "Recent progress, challenges and future prospects of applied deep reinforcement learning: A practical perspective in path planning," *Neurocomputing*, vol. 608, p. 128423, 2024.

[4] D. Rawat, M. K. Gupta, and A. Sharma, "Intelligent control of robotic manipulators: a comprehensive review," *Spatial Information Research*, vol. 31, no. 3, pp. 345–357, 2023.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2018.

[6] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[8] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, vol. 23, no. 7, p. 3762, 2023.

[9] T. Lindner and A. Milecki, "Reinforcement learning-based algorithm to avoid obstacles by the anthropomorphic robotic arm," *Applied Sciences*, vol. 12, no. 13, p. 6629, 2022.

[10] Q. Xu, T. Zhang, K. Zhou, Y. Lin, and W. Ju, "Active collision avoidance for robotic arm based on artificial potential field and deep reinforcement learning," *Applied Sciences*, vol. 14, no. 11, p. 4936, 2024.

[11] J. Hu, J. Mao, X. Zhou, and C. Zhang, "Real-time obstacle avoidance and pathfinding for robot manipulators based on deep reinforcement learning," in *International Conference on Intelligent Robotics and Applications*. Springer, 2024, pp. 154–166.

[12] P. Chen, J. Pei, W. Lu, and M. Li, "A deep reinforcement learning based method for real-time path planning and dynamic obstacle avoidance," *Neurocomputing*, vol. 497, pp. 64–75, 2022.

[13] Z. Liu and S. Liu, "Improved residual reinforcement learning for dynamic obstacle avoidance of robotic arm," in *2024 IEEE 13th Data Driven Control and Learning Systems Conference*. IEEE, 2024, pp. 1941–1946.

[14] P. Wu, H. Su, H. Dong, T. Liu, M. Li, and Z. Chen, "An obstacle avoidance method for robotic arm based on reinforcement learning," *Industrial Robot: the international journal of robotics research and application*, vol. 52, no. 1, pp. 9–17, 2025.

[15] DeepMind, "Mujoco: Multi-joint dynamics with contact," 2024, accessed: 27-Mar-2025. [Online]. Available: https://mujoco.org/

[16] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of machine learning research*, vol. 22, no. 268, pp. 1–8, 2021.

[17] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[18] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

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

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.