Velocity Planning with Multi-Objectives in Displacement-Time Graphs Using Deep Reinforcement Learning

Liyang Wang[1,*], and Rui Zeng[1]

***Abstract*— This paper presents a novel velocity planning method in displacement-time graphs with multiple constraints and optimization goals using deep reinforcement learning. The method formulates the velocity planning problem as a reinforcement learning task with state representation including time, position, velocity, acceleration, and distances to each obstacle triangle representative. The action space is discretized within allowable accelerations, and the kinematics ensure velocity constraints during state transitions. The advantage of this method lies in its independence from scene-specific tuning, and exhibiting robustness in various complex scenarios. Comparative analysis demonstrates a 100% success rate, along with superior computational efficiency when contrasted with the baseline approach, while also exhibiting better comfort performance. It offers a valuable alternative for velocity planning in robotics and autonomous vehicles, showcasing deep reinforcement learning's potential in practical robotics applications.**

I. INTRODUCTION

Velocity profile planning is a fundamental and critical aspect of trajectory planning in numerous robotics applications, including autonomous driving car [1, 2, 3], unmanned aerial vehicles (UAVs) [4, 5], and civil aviation aircraft [6, 7]. This problem is challenging due to several factors. First, it requires avoiding moving obstacles, such as autonomous vehicles avoiding pedestrians and surrounding cars, or aircraft avoiding birds and other airborne vehicles. Second, it involves satisfying the robot's physical constraints, such as speed and acceleration limits. Finally, there is a need to optimize objectives such as minimizing travel time and enhancing passenger comfort. These multiple objectives make velocity profile planning difficult in robotics applications.

In this work, we focus on the problem of velocity profile planning under multiple optimization goals and constraint conditions, leveraging the concept of Displacement-time graph (S-T graph). S-T graph is widely used since velocity profile planning can be converted into a path planning problem in a 2D space [8]. An example of velocity planning in S-T graph is shown in Fig. 1. A point mass robot moves along a specified path and avoids moving obstacles by planning its velocity. In the S-T graph, robot starts from the origin, finds a velocity profile leading to any valid point on the top, which represents the destination.

Smoothed velocity profile traditionally can be created through convex-optimization-based or sampling-based methods [9]. In [3], researchers generate results using a quadratic programming (QP) framework. However, QP solver could fail in complex dynamic environments with multi-objectives due to the S-T graph may not be convex. If an initial result is provided, CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [10] and STOMP (Stochastic trajectory optimization for motion planning) [11] are frequently employed since they can handle multiple objectives by setting appropriate cost functions [12]. However, CHOMP and STOMP rely on good initial result to reduce computation time, which can be challenging to obtain in practical scenarios. Moreover, designing cost functions that properly reflect the desired objectives while accounting for multiple constraints is a complex undertaking.

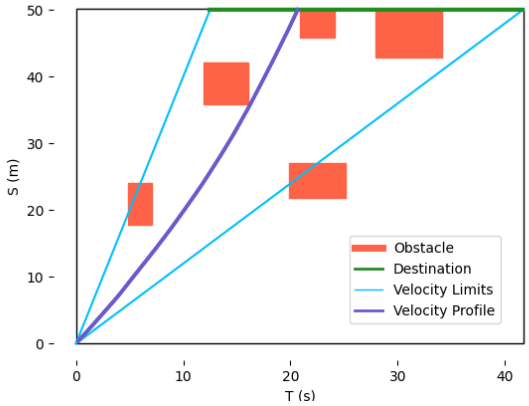


Fig. 1. An illustration of velocity planning in S-T graph. In this example, the total length of the path is $50m$, the upper and lower limits of velocity are $4.0m/s$ and $1.2m/s$, the upper and lower limits of acceleration are $0.5m/s^2$ and $-0.3m/s^2$. Moving obstacles are shown as orange blocks, representing occupancy of the path segment during the corresponding time frame. The green bar on the top represents the path destination. The two light blue lines are calculated based on the upper and lower limits of the velocity, representing the problem boundary. A planned velocity profile example is shown as a purple line.

We aim to address these challenges by exploring a novel deep reinforcement learning (DRL)-based approach. In recent years, DRL has achieved remarkable success, particularly with methods based on policy gradients, such as Deep Deterministic Policy Gradient (DDPG) [13], Advantage Actor-Critic (A2C) [14], and Proximal Policy Optimization (PPO) [15]. These algorithms are widely used in robotics tasks like navigation [16], path planning [17], and collision avoidance [18, 19].

In this paper, we tackle the problem of velocity planning with multi-objectives in S-T graph. We assume that the S-T graph is given since robot path is already determined and

*This work was not supported by any organization

[1]Liyang Wang and Rui Zeng are with COMAC Beijing Aircraft Technology Research Institute, North 1st Street, Future City, Beijing, 102209, China. Emails: wangliyang, zengrui@comac.cc

* Corresponding author

the trajectories of moving obstacles are known. The objective is to find a path in the S-T graph that allows the robot to reach the destination while avoiding all moving obstacles. The planned velocity profile requires satisfying the upper and lower limits of the robot's velocity and acceleration, while minimizing acceleration changes to improve comfort.

In our method, the problem is modeled as a Markov Decision Process (MDP) and utilizes PPO algorithm to obtain velocity planning results. The state representation includes the robot's position, velocity, acceleration, time, and distances to each obstacle representative on the S-T graph. Acceleration is discretized within the upper and lower limits, forming the action space. The state transition is based on the robot's kinematic equations. The reward of each step is the progress of the robot made. The results generated by our proposed method can always guarantee satisfaction of all constraints. The proposed method handles non-convex environments and does not require initial guess or parameter tuning based on different scenarios. By training the PPO agent in different S-T graph settings, we demonstrate that this method enables the robot to adaptively plan its velocity based on the environment and achieve safe and comfort velocity profile in complex scenarios with moving obstacles. In addition, we designed and implemented a baseline solution employing breadth-first search (BFS) within a grid generated from the S-T graph, while meticulously considering all constraints. When compared to the baseline, the proposed method showcases a 100% success rate and outperforms the baseline in terms of computational efficiency. Moreover, it demonstrates better comfort performance.

II. METHODOLOGY

The scenarios represented by the S-T graph vary significantly depending on factors such as constraints, the number and positions of obstacles. In this section, we introduce our method using DRL to handle various scenarios. Currently, for each scenario, the model is trained independently.

A. State Representation

We divide the agent's state S into two parts, that is $S = [S_a, S_o]$. $S_a = [s, v, a, t]$ represents the agent's current position s, velocity v, acceleration a, and time t. S_o is obstacles related states. Given t and s, in S-T graph, the distance to each obstacle can be calculated. We denote $S_o = [d^1, d^2, ..., d^n]$, where d^i indicates the distance to the i_{th} obstacle representative in S-T graph.

Obstacles in the S-T graph are rectangular. However, directly using the distance of a point (t, s) to a rectangular obstacle as a state in the S-T graph can be problematic. Such kind of distance calculation method could generate local minima, leading the agent stuck at an obstacle and not able to reach the destination.

To solve this problem, we propose a concept of *companion triangle* to replace the rectangular representation of obstacles. Use the same setting in Fig. 1, Fig. 2 shows companion triangle generation results based on obstacle rectangular. The first two vertices of the companion triangle are the upper left and lower right corners of the obstacle rectangle. The third vertex is determined by upper and lower limits of velocity. Take the upper left corner point and the maximum velocity as the slope to determine the first straight line, take the lower right corner point and the minimum velocity as the slope to determine the second straight line, and the intersection of the two straight lines is the third vertex of the companion triangle. It can be seen from Fig. 2 that, if the (t, s) is located within the companion triangle area, due to the upper and lower limits of the velocity, the corresponding obstacle cannot be avoided although no collision at current time t.

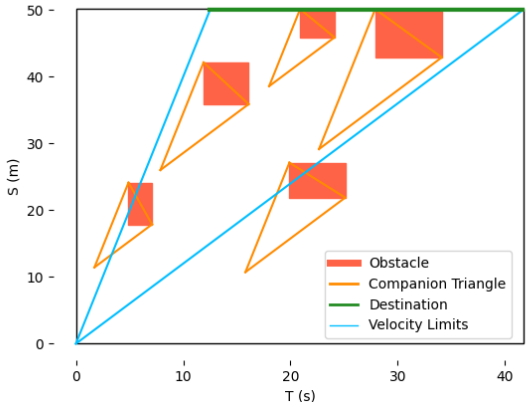
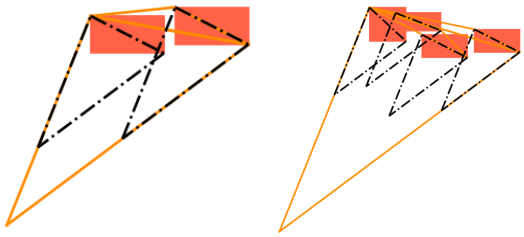


Fig. 2. Companion triangle generation based on obstacle rectangular.



(a) Merge two triangles (b) Merge multiple triangles

Fig. 3. Illustration of companion triangles merge when have overlapping.

We also employ a triangle merging algorithm to avoid getting stuck in local optima. If two triangles have overlapping areas and may cause local optima, we merge them into one larger triangle, which can cover this local optimal region. We repeat the merging process until not able to merge. Fig. 3 illustrates how companion triangles are merged. The black dashed triangles are original companion triangles of corresponding obstacles, and the larger orange triangles are merged results. In Fig. 3a, two triangles are merged, while in Fig. 3b, the merge process repeats until all four associated triangles are merged. Triangles will be removed if they are merged. We call the triangles after the merging process as representatives of the obstacles in S-T graph. The state d^i in S_o is then the distance to the i_{th} representative.

B. Action Space

The constraints of the upper and lower limits of the acceleration, which are denoted as a_{max} and a_{min}, are guaranteed by the design of action space A. Given total

number of actions n, the action incremental Δa is calculated as follows:

$$\Delta a = \frac{a_{max} - a_{min}}{n-1} \tag{1}$$

The action space is then defined as:

$$A = [a_{min}, a_{min} + \Delta a, a_{min} + 2\Delta a, ..., a_{max}] \tag{2}$$

The agent picks an acceleration $\bar{a} \in A$ at each time step i, which is also set as the state a in S.

C. State Transition

We formulate the problem as deterministic. The state transition is based on kinematic equations, and the upper and lower limits of the velocity, which are denoted as v_{max} and v_{min}, are guaranteed. The time incremental between two steps is fixed as Δt. State transition law for S_a is described in Eqn. 3. Once new position (t_{i+1}, s_{i+1}) in S-T graph is determined, S_o at step $i+1$ can also be updated.

$$\begin{aligned} t_{i+1} &= t_i + \Delta t \\ a_{i+1} &= \bar{a}_i \\ v_{i+1} &= \begin{cases} v_{min} & \text{if } v_i + a_{i+1}\Delta t < v_{min} \\ v_{max} & \text{if } v_i + a_{i+1}\Delta t > v_{max} \\ v_i + a_{i+1}\Delta t & \text{other case} \end{cases} \\ s_{i+1} &= s_i + 0.5(v_i + v_{i+1})\Delta t \end{aligned} \tag{3}$$

The MDP in this work is episodic. There are two conditions of termination. The first is when the agent reaches the destination s_{max}, as $s_i \geq s_{max}$. The second is when (t_i, s_i) is located inside one of the companion triangles.

D. Reward Function

The agent gets reward r_i at each step is defined as the progress it makes at that step, which encourages the agent to reach s_{max}. The reward function is shown in Eqn. 4.

$$r_i = \frac{100(s_{i+1} - s_i)}{s_{max}} \tag{4}$$

In order to avoid local optima, we did not add time cost in the reward design. The optimization goal of using as little time as possible can be addressed by setting the discount factor to less than 1, which encourages the agent to make as much progress as possible in the most recent steps. Comfort optimization goal, usually expressed in terms of reducing acceleration variation, is also not included in the reward design. From the experiments presented in Section III, we found that the results produced by the DRL method are inherently smooth, and there is no sudden acceleration change phenomenon. At the same time, adding acceleration change punishment to the reward usually causes failure.

E. Deep Reinforcement Learning Configurations

The PPO algorithm is chosen in this work due to its high efficiency and robustness. In order to avoid falling into the gradient cliff, the PPO algorithm uses the method of KL divergence or gradient clip. According to [15], the two methods have similar performance and the gradient clip method is easier to implement. In this work, we apply a gradient clip version of PPO with clip coefficient $\epsilon = 0.2$.

The deep neural networks of the actor and critic are set independently. They both consist of 8 fully connected layers. The input to both the actor and critic network is the state vector S. The output of the actor network is a probability distribution of the n actions, while the output of the critic network is the state's evaluation represented by a scalar. Training follows the generalized policy iteration (GPI) [20] loop, with transition buffer size equal to 4096 and batch size equal to 64. We set the number of actions $n = 12$. Adam is used for gradient descent. The learning rates are set equal to 10^{-4}. We train 5 epochs for evaluation and 2 epochs for improvement in each GPI loop.

Bootstrap step size b and discount factor γ are two important hyperparameters that affect the performance significantly, where b balances between bias and variance of the value estimation, and γ controls how much the agent values long-term returns. To find the best choice of b and γ, we pick one complex scenario and apply different b and γ pairs. Their values are determined by evaluating the corresponding model performance. Details are presented in Section III-B.

III. RESULTS

A. Scenarios Collection

The goal of scenarios collection is to obtain S-T graph environments that cover a wide distribution. To do this, we utilize the actual flight trajectory record data of China's civil aviation aircraft for the whole day on April 19, 2021, to generate sufficient and diverse experimental scenarios. In the experiment, a virtual aircraft serves as the agent and is required to plan its velocity along the provided route to avoid surrounding flights, while ensuring its capability and optimizing passenger comfort as much as possible. Given the location and heading angle of the virtual aircraft, the destination location, and the virtual no-fly zone, a RRT* [21] planner and B-Spline smoother is used to generate a 2D path in the area as the provided route. Then, S-T graph space can be generated based on initial time and the path, by analyzing occupancy of the path using real flight trajectories in the area.

For stochasticity, we pick the virtual aircraft's capability, location, and heading angle randomly for each scenario. The aircraft's capability includes v_{min}, v_{max}, a_{min}, and a_{max}. Each randomly drawn variable follows a uniform distribution. In this work, the boundaries of them are shown in Table I.

We set the total time of each scenario to be $2230s$, time interval $\Delta t = 10s$, initial velocity $v_0 = 188m/s$, and acceleration $a_0 = 0m/s^2$. The state $[s_2, v_2, a_2, t_2]$ can be calculated using Eqn. 3, assume that $a_1 = a_0$. Consider S_2 as the first state of the episode, the total steps number is 220.

With v_{min}, we can calculate the maximum travel distance $s_{max} = 2230v_{min}$. In 2D, taking the current position as a vector's initial point, the heading angle as the vector's direction, and s_{max} as the vector's magnitude, the vector's endpoint is considered as a virtual destination. Then, a virtual circular no-fly zone with radius $r = 0.2s_{max}$ is placed in the

TABLE I

RANDOM VARIABLE BOUNDARIES FOR TEST SCENARIOS COLLECTION

Random Variables	v_{min} (m/s)	v_{max} (m/s)	a_{min} (m/s^2)	a_{max} (m/s^2)	Longitude (°E)	Latitude (°W)	Heading (rad)
Minimum	75.0	255.0	-4.0	3.0	102.0	20.0	$-\pi$
Maximum	120.0	300.0	-2.5	9.0	123.0	41.0	π

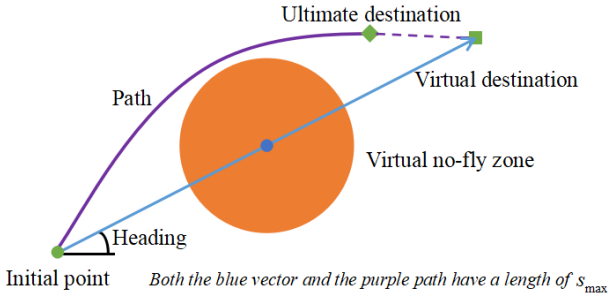


Fig. 4. An illustration of the path generation process. The entire purple line from the initial point to the virtual destination is the result of the RRT* planner and B-Spline smoother. The solid purple line from the initial point to the ultimate destination is the path used as the provided route with length equal to s_{max}. This path generation process ensures stochasticity and makes the scenario for experiments more realistic.

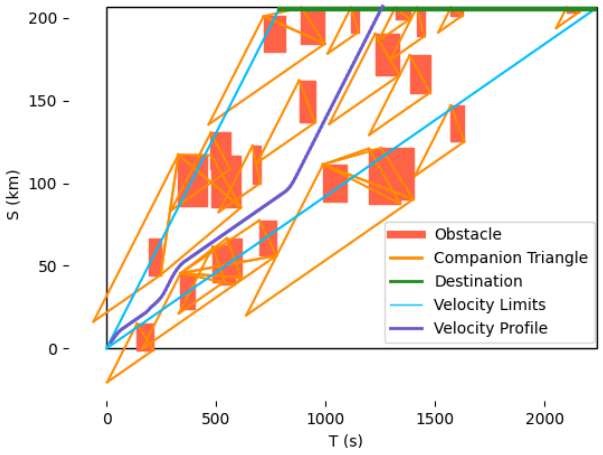


Fig. 5. Scenario used for bootstrap step size b and discount factor γ tuning. The velocity profile shown in purple line is the planning result of the fully trained model using $b = 64$ and $\gamma = 1.0$.

middle of the vector. We can get a path from current position to the virtual destination using RRT* planner and B-Spline smoother. Finally, on the overall path, starting from the initial point, a segment of length s_{max} is extracted to form the final path, thereby determining the ultimate destination as well. This process is illustrated in Fig. 4.

The initial time is set at 5:00 PM on that day, which is typically busy in the airspace, resulting in more complex scenarios. Matching the actual aircraft trajectory data within the vicinity airspace during the corresponding time interval with the generated path enables the identification of time slots occupied by the path and the segments of the path that intersect with existing trajectories. S-T graph is created through this process.

Based on the real flight trajectory data and the collection method presented above, we collect a total of 500 random scenarios. In these scenarios, we remove the scenarios without any obstacles in the planable area on the S-T graph, because this is trivial in the realistic. We also apply baseline method presented in Section III-C on each scenario with $\eta = 0.25$ (in Eqn. 5). We remove all scenarios where the baseline cannot find results, because this means that there is possibly no solution for this scenario. Finally, we have 352 qualifying scenarios that are being utilized for experimentation.

B. Hyperparameters Tuning

As discussed in Section II-E, we chose a relatively complex scenario from the pool of 352 experimental scenarios for b and γ tuning. The selected scenario is shown in Fig. 5. We test b of 10, 12, 16, 64, and 220, as from TD(9) to Monte Carlo [20]. We also test γ of 0.9, 0.96, 0.99, and 1.0. The b and γ are paired in combinations, resulting in a total of 20 experimental runs. For each run, we train 80 GPI loops from scratch. In each loop, the data collection phase includes multiple episodes, and for each episode, the agent's progress is quantified by calculating the percentage of the traveled distance relative to the total distance. We take the overall average progress of all episodes as the evaluation metric to determine the best b and γ. And the best progress of all episodes is also counted and used as a reference.

The overall average progress results of the 20 experiments are shown in Fig. 6, and the corresponding best progress results are shown in Table II. From Fig. 6, an agent gets the highest overall average progress when $b = 64$ and $\gamma = 1.0$, and those values are picked for the comparative analysis presented in Section III-D. The agents who successfully reach the destination at least once in all episodes are those shown as 100 in Table II. We also recorded the average total return of 100 consecutive episodes during the training process of each experiment, shown in Fig. 7. Based on the results, bias, and variance are best balanced at $b = 64$, and $\gamma = 1.0$ achieves the best results because, in our problem formulation, long-term progress is just as important as current progress.

TABLE II

BEST PROGRESS RESULTS

$\gamma \backslash b$	**10**	**12**	**16**	**64**	**220**
0.9	17.58	17.82	27.77	17.56	18.94
0.96	17.65	19.53	17.12	22.20	**100**
0.99	18.47	25.16	17.09	**100**	**100**
1.0	23.99	**100**	**100**	**100**	**100**

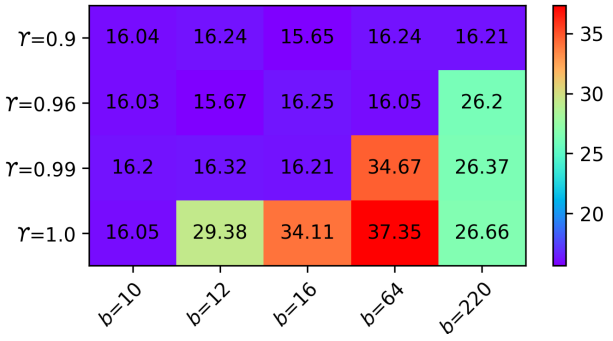


Fig. 6. Overall average progress results of the 20 experiments for bootstrap step size b and discount factor γ tuning.

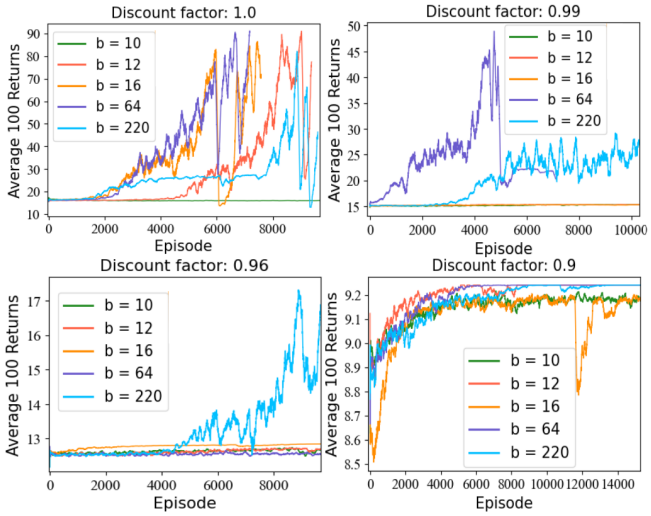


Fig. 7. Average total return of 100 consecutive episodes during the training process of each experiment.

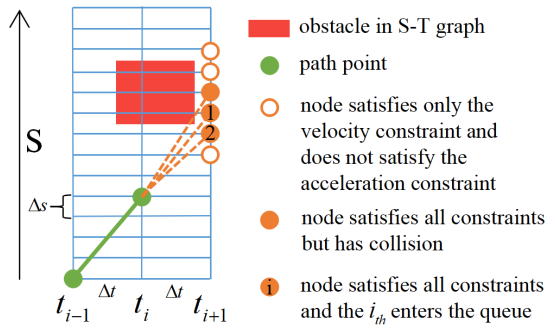


Fig. 8. Illustration of node selection and the order in which nodes enter the queue of the BFS algorithm.

C. Baseline Approach

A graph-search-based method with all constraints considered is designed and implemented as the baseline approach. This method first generates grids on the given S-T graph, then uses BFS to find the shortest path from the origin to the top of the S-T graph. During the search, the velocity and acceleration limits are fulfilled by controlling the next nodes that are pushed into the queue [22] of the BFS algorithm. To optimize the time to the destination, nodes making more progress are pushed first. In the baseline method, all collision check is against the original rectangular obstacle, not the companion triangle.

The grid based on the S-T graph has a horizontal spacing equal to the specified time interval Δt, and the vertical spacing is determined by Eqn. 5, where η controls the density of the node points.

$$\Delta s = \eta v_{min} \Delta t \tag{5}$$

With the generated grid and current path end at t_i, the node candidates to push to the queue are located at t_{i+1}. By evaluating the connection between each candidate node and the current path endpoint node, eliminating all nodes that involve collisions or fail to satisfy velocity and acceleration constraints, the remaining candidate nodes are then sequentially pushed into the queue in a top-to-bottom manner. Fig. 8 illustrates the node selection and the order in which nodes enter the queue.

The proposed baseline method requires the establishment of dense grid (small η in Eqn. 5) to ensure a solution, which, due to the time complexity of BFS, results in a significant consumption of computational time. We also attempted to apply another baseline method. First, establish a sparse grid and use BFS without acceleration limits considered, then apply B-spline smoothing technique [23] to obtain a smooth path as initial velocity profile. Finally, utilize a STOMP optimizer[11] to achieve an optimal velocity profile planning result that satisfies all constraints. However, this approach did not yield success because constructing the cost functions of this problem for STOMP is extremely challenging, resulting in convergence difficulties. We don't use RRT* as a baseline because constraints are very hard to be guaranteed.

D. Comparative Analysis

The results generated by both methods ensure compliance with all constraints. With the 352 scenarios picked by the baseline method, we first test the planning success rate of the presented DRL method and then analyze computation time and comfort for both methods. We define *first result* mode and *fully trained* mode for DRL. In the first result mode, the agent is trained from scratch, and the planning result is obtained from the first successful reaching of the destination in the data collection phase of GPI loops. In fully trained mode, we take the best model obtained after training 80 GPI loops from scratch, then get the planning results by starting from the initial state on the S-T graph, and query the actor model to navigate until the episode ends.

We apply the first result mode of the DRL method on all 352 scenarios and achieve 100% success rate. This outcome demonstrates that situations in which the baseline method finds a result are also successfully handled by the DRL method. Conversely, we posit that scenarios where the DRL method finds a path may not necessarily lead to success with the baseline approach. This is due to the baseline's heavy reliance on the density of grid points, necessitating individual adjustments for each scenario to reduce computational time.

TABLE III

EXPERIMENTAL RESULTS ON REPRESENTATIVE COMPLEX SCENARIOS FOR COMPARATIVE ANALYSIS

Scenarios	**Computation time of baseline**(s)	**Computation time of DRL first result**(s)	**Comfort metric of baseline**(m/s^2)	**Comfort metric of DRL fully trained**(m/s^2)	**Comfort metric of DRL first result**(m/s^2)
1	144.09	76.11 / 188.50 / 66.87	112.44	82.23 / 84.77 / 78.63	205.46 / 153.75 / 158.59
2	552.81	92.11 / 95.30 / 72.21	98.74	70.6 / 93.50 / 72.99	138.22 / 146.42 / 147.25
3	57.75	57.96 / 86.38 / 35.17	82.18	63.53 / 74.77 / 70.03	131.98 / 114.80 / 94.63
4	29.33	5.10 / 33.52 / 19.36	83.07	72.33 / 43.14 / 53.16	155.41 / 132.87 / 122.13
5	84.70	174.54 / 102.48 / 150.12	78.60	77.90 / 66.18 / 71.56	192.98 / 179.36 / 175.10
6	298.15	267.3 / 257.71 / 243.04	126.71	53.36 / 68.24 / 61.49	122.31 / 123.08 / 104.65
7	334.77	1.96 / 12.82 / 4.05	117.77	91.35 / 68.71 / 87.66	111.77 / 124.16 / 105.08
8	40.68	7.11 / 4.84 / 28.83	68.18	28.62 / 40.93 / 34.34	133.17 / 152.81 / 117.60

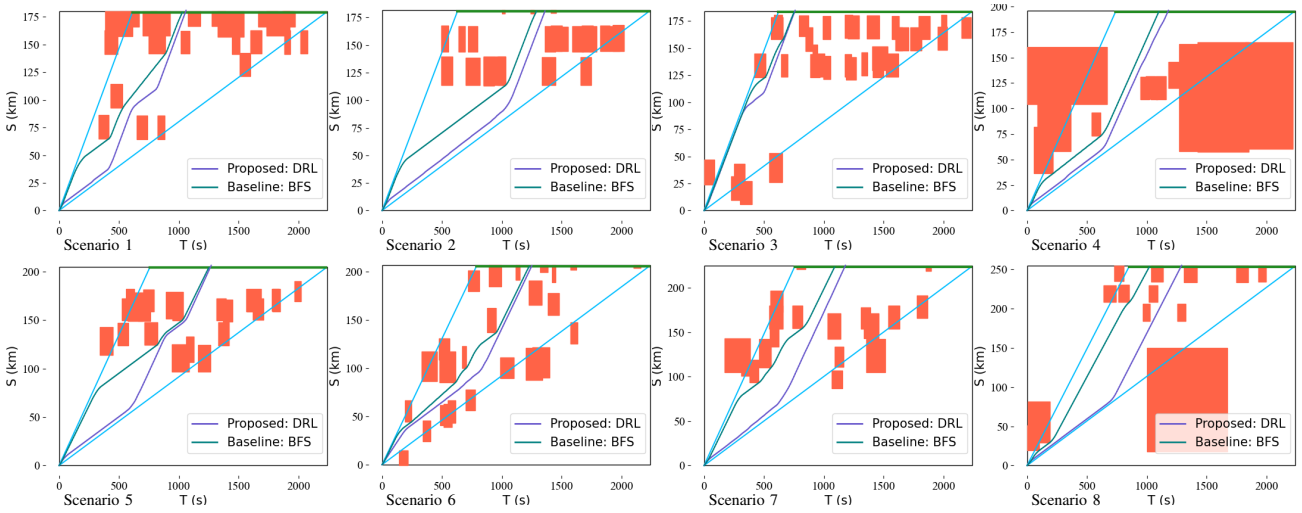


Fig. 9. Representative complex scenarios and the planning results of baseline method and DRL fully trained actor.

If we set $\eta = 0.5$, the baseline only succeeded in 191 scenarios within the original 352 scenarios.

In the experiments, the computation time by both methods to yield results varies significantly based on the complexity of the scenario. The average computation time for the baseline method is $61.10s$, the average computation time for the first result mode DRL is $31.81s$, and DRL exhibits lower computation time compared to the baseline method in 94.9% of the scenarios. Computation time for fully trained mode DRL is not collected since it generates results almost instantly.

We conduct a more in-depth analysis by selecting 8 representative complex scenarios. Due to stochasticity, each relevant DRL test is conducted through 3 independent experiments. The metric for comfort evaluation is defined as the accumulated acceleration of the entire velocity profile, and the lower the value the better the comfort. Detailed test results are listed in Table III, which demonstrates that compared with the baseline method, in general, the first result mode DRL takes less computation time, and the fully trained mode DRL has better comfort. In Fig. 9 we plot velocity profile results for the 8 scenarios of baseline method and fully trained mode DRL. Based on the discussion in Section II-D, due to $\gamma = 1.0$ and the absence of time-related rewards, the velocity planning generated by DRL typically takes more time compared to the baseline method.

IV. CONCLUSION

In conclusion, we present a comprehensive framework for multi-objective velocity planning. Our approach utilizes the S-T graph concept and formulates the problem as a MDP and employs the DRL algorithm to ensure efficient, collision-free velocity planning, and adherence to velocity and acceleration constraints while prioritizing passenger comfort. Comparative analysis against the introduced baseline method highlights the superiority of our proposed approach, achieving a 100% success rate and outperforming the baseline in terms of computational efficiency and comfort.

In the future, we would like to extend our work to minimize the planned velocity profile total time by designing appropriate rewards. Currently, while there is no need for individual scenario-specific tuning like the baseline, the presented DRL method requires training from scratch for each scenario due to varying state vector lengths across different scenes. Based on the fact that a fully trained model generates an optimized velocity profile almost instantly, we would like to investigate universal model that can be applied to any scenario. A potential approach is convert a sequence of continuous S-T graphs into images and use them as the input state representations. Correspondingly, the model could incorporate well-established convolutional neural networks (CNN), allowing the agent to autonomously identify optimal features. Then universal actor could be obtained by training on multiple scenarios simultaneously.

REFERENCES

[1] K. Zhang, X. Feng, L. Wu, and Z. He, "Trajectory prediction for autonomous driving using spatial-temporal graph attention transformer," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 22 343–22 353, 2022.

[2] A. Said, R. Talj, C. Francis, and H. Shraim, "Local trajectory planning for autonomous vehicle with static and dynamic obstacles avoidance," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2021, pp. 410–416.

[3] T. Wang, D. Qu, H. Song, and S. Dai, "A hierarchical framework of decision making and trajectory tracking control for autonomous vehicles," *Sustainability*, vol. 15, no. 8, p. 6375, 2023.

[4] J. Hu, X. Yang, W. Wang, P. Wei, L. Ying, and Y. Liu, "Obstacle avoidance for uas in continuous action space using deep reinforcement learning," *IEEE Access*, vol. 10, pp. 90 623–90 634, 2022.

[5] T. Wakabayashi, Y. Suzuki, and S. Suzuki, "Dynamic obstacle avoidance for multi-rotor uav using chance-constraints based on obstacle velocity," *Robotics and Autonomous Systems*, vol. 160, p. 104 320, 2023.

[6] W. Zeng, X. Chu, Z. Xu, Y. Liu, and Z. Quan, "Aircraft 4d trajectory prediction in civil aviation: A review," *Aerospace*, vol. 9, no. 2, p. 91, 2022.

[7] W. Dai, B. Pang, and K. H. Low, "Conflict-free four-dimensional path planning for urban air mobility considering airspace occupancy," *Aerospace Science and Technology*, vol. 119, p. 107 154, 2021.

[8] J. Johnson and K. Hauser, "Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 2035–2041.

[9] Y. Zhu, L. Wang, and L. Zhang, "Excavation of fragmented rocks with multi-modal model-based reinforcement learning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 6523–6530.

[10] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE international conference on robotics and automation*, IEEE, 2009, pp. 489–494.

[11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 4569–4574.

[12] Q. Guo, Z. Ye, L. Wang, and L. Zhang, "Imitation learning and model integrated excavator trajectory planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 5737–5743.

[13] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[14] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.

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

[16] O. Bouhamed, H. Ghazzai, H. Besbes, and Y. Massoud, "Autonomous uav navigation: A ddpg-based deep reinforcement learning approach," in *2020 IEEE International Symposium on circuits and systems (ISCAS)*, IEEE, 2020, pp. 1–5.

[17] B. Li and Y. Wu, "Path planning for uav ground target tracking via deep reinforcement learning," *IEEE access*, vol. 8, pp. 29 064–29 074, 2020.

[18] S. Ouahouah, M. Bagaa, J. Prados-Garzon, and T. Taleb, "Deep-reinforcement-learning-based collision avoidance in uav environment," *IEEE Internet of Things Journal*, vol. 9, no. 6, pp. 4015–4030, 2021.

[19] X. Wu *et al.*, "The autonomous navigation and obstacle avoidance for usvs with anoa deep reinforcement learning method," *Knowledge-Based Systems*, vol. 196, p. 105 201, 2020.

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

[21] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: A survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[22] D. C. Kozen and D. C. Kozen, "Depth-first and breadth-first search," *The design and analysis of algorithms*, pp. 19–24, 1992.

[23] D. Jung and P. Tsiotras, "On-line path generation for unmanned aerial vehicles using b-spline path templates," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 6, pp. 1642–1653, 2013.