RAIL: Recursive A* and Imitation Learning-Based Approach for Collision-Free Path Estimation

Subhadip Chandra
IEM Centre of Excellence for Data Science, Department of CSE (AIML), Institute of Engineering and Management, University of Engineering & Management,
Kolkata-700091, India.
subhadipchnadra@gmail.com

Kamlesh Kumar Dubey
Department of Applied Sciences and Humanities,
Invertis University
Bareilly, UP, India
kkdubey4maths@gmail.com

Gaurav Agarwal
Department of Computer Science & Engineering,
Invertis University
Bareilly, UP, India
gaurav.a1@invertis.org

Anumoy Pathak
Department of CSBS,
Institute of Engineering and Management, University of Engineering &Management,
Kolkata-700091, India.
malapathak.india@gmail.com

2025 8th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech) | 979-8-3315-2076-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/IEMENTECH65115.2025.10959614

Abstract— **This paper discusses collision-free path planning for autonomous vehicles, a crucial aspect of their development. The authors propose a method to detect objects of different sizes, both dynamic and static, and compare algorithms for path planning and avoidance of collision. They use traditional path-finding algorithms like A* Algorithm, which include heuristics and Rapidly Exploring Random Tree (RRT) algorithms for real-time analysis. The authors discuss the challenges of ensuring path-planning algorithms scale to large environments and operate in real-time, and the need for future work on improving the robustness of path-planning algorithms in dynamic and unpredictable scenarios.**

The methodology involves creating a dataset, analyzing images, and training neural networks to imitate the collision-free path- finding algorithm. Static obstacles change their coordinates with time, making working in dynamic environments difficult. Recursive A* and Imitation Learning (RAIL) based approaches analyze obstacles frame by frame, making them static at a particular time-instant. The Generative Adversarial Network (GAN) and Inverse Reinforcement Learning are used to train an imitation learning model for faster trajectory finding in autonomous vehicles. Future development could involve using the Internet of Things (IoT) to share the trained trajectory-finding model results.

***Keywords*— collision free path planning, imitation learning, heuristic, a* algorithm, obstacle detection**

I. INTRODUCTION

The appeal of autonomous vehicles in manufacturing, transportation, and routine works and work in hazardous environments is increasing every day. For the progress

in the domain of autonomous vehicles proper trajectory finding for loco-motion is important. Autonomous vehicular transport is not possible without proper path planning. Overcoming the obstacles in a dynamic environment is a major issue. Hence collision-free trajectory finding algorithms become very important. Over the years many path-finding algorithms have been developed and implemented in different areas. Dijkstra's algorithm [1] is one of the best shortest path-finding algorithms for static environments. According to Foed, A*[2] is one of the best approaches to finding a path.

According to Noreen, Rapidly Exploring Random Trees (RRT) [3], RRT* are the algorithms that have optimized the way of path finding. But the problem arises when it's a dynamic environment, where the obstacles are changing their position at every instant. All the above-mentioned algorithms are fruitful only in the case of a static environment. Autonomous vehicles need real-time analysis to find the perfect trajectory. Newer approaches such as imitation learning are one of the most researched areas for finding paths for autonomous vehicles. But such kind of deep learning model requires huge training data. It's also not possible to demonstrate every complex situation and feed it to the model. So, in this article, we use a traditional path-finding algorithm like A* and modify it so that its behavior can be imitated for proper trajectory finding in real-time. In this paper, we propose a novel approach that offers the following:

- Detects obstacles in a dynamic environment.
- Imitates the behavior of A* (Recursive)search algorithm .
- Finds a collision-free trajectory for autonomous vehicular system.

II. LITERATURE SURVEY

Collision-free path planning is a basic problem in autonomous vehicular navigation. Here we need to

979-8-3315-2076-2/25/$31.00 ©2025 IEEE

determine a trajectory from a starting point to a destination while avoiding obstacles. The challenge grows with the complexity of the environment and the need for real-time computation. The creation of novel reinforcement learning algorithms, like actor- critic, for constrained Markov decision processes (CMDPs) has been the focus of recent research [4,5,6]. Although these techniques are appreciated for their simplicity and wide applicability, they frequently call for a model or simply attempt to satisfy safety requirements in a probabilistic way [7]. As an alternative, some strategies concentrate on creating control barrier functions (CBFs) that can assure safety. Usually, a safety filter is integrated into a reference controller to create these safe controllers. In the obstacle avoidance problem, the use of CBFs in conjunction with model predictive control (MPC) has shown notable outcomes [8, 9, 10]. There are, in fact, controllers that combine control barrier functions and reinforcement learning [11, 12]. Nevertheless, these controllers treat the CBF and RL as two distinct controllers and usually do not explicitly account for the action adjustments made by the CBF layer within the loss function of the RL algorithm.

A. Path Planning Techniques

Dijkstra's Algorithm is one of the earliest methods used for path finding, Dijkstra's algorithm computes the shortest path between nodes in a graph, considering edge weight. It is applicable to static environments but may fail in dynamic obstacles environment. Whereas an extension of Dijkstra's, the A* algorithm includes heuristics to select paths that are more promising. It combines the cost to reach the node and the estimated cost to the goal, making it effective for real-time ap- plications. RRT algorithm is designed to explore large spaces quickly. They incrementally build a tree of feasible paths and are particularly useful for complex, high-dimensional spaces. Variants like RRT* improve path quality by optimizing paths over time.

B. Collision Avoidance Strategies

According to Wei, the behavioural planning [13], approach involves predicting the behaviour of other agents and adjusting the vehicle's path accordingly. It incorporates models of traffic rules and human-like decision-making processes. According to Ho, imitation learning [14] models are trained to mimic the behaviour of human drivers or expert systems. This method leverages demonstrations to learn complex driving policies and improve decision-making in various scenarios.

C. Scalability and Real-Time Performance

Ensuring that path-planning algorithms scale to large environments and operate in real time remains a challenge. Techniques like parallel computing and hardware acceleration are being explored to address this.

D. Dynamic Environments

Handling dynamic obstacles and changing environments is a major research area. Future work involves improving the robustness of path-planning algorithms in highly dynamic and unpredictable scenarios.

III. Proposed Methodology

At the very beginning, the images of roads and traffic are collected from various traffic datasets available on the internet. In the next phase, data pre-processing is done. At first, the nature of the environment is taken into account. Then the RGB images are converted to grayscale images. After that, the grayscale images are masked contours are plotted and the contour area is calculated. Objects with considerable area are considered. Thus various obstacles are detected from the images. Following it obstacle matrix is created. Then Iterative A* algorithm is implemented on the matrix to find the optimal path. After that, a GAN model is trained to imitate the behaviour of Iterative A* and apply it on test dataset. Thus, collision free path finding algorithm is implemented.

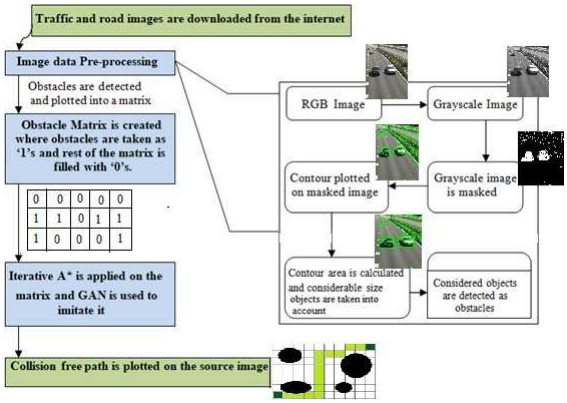


Fig. 1 Overall Flow of the Proposed Methodology

A. Dynamic Environments

The environment is one of the key aspects. Environments are of two types: a) Static and b) Dynamic. Fig. 2(a) represents static obstacles on which do not change with time and Fig. 2(b) represents a dynamic obstacle which changes its coordinates with time. For autonomous vehicles the dynamic environment is to be considered. In dynamic environment the obstacles change their coordinates with respect to time but working in a dynamic environment is difficult. In Fig. 2 F(x, y) represents a 2D plane with obstacles having x and y co-ordinates.

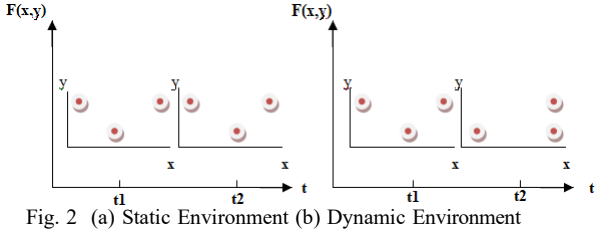


Fig. 2 (a) Static Environment (b) Dynamic Environment

B. Obstacle Detection

Recursive A* and Imitation Learning (RAIL) based approach for collision-free path estimation, analyses obstacles frame by frame. Thus, making obstacles static at a particular time-instant. First, a frame is masked and contours are plotted. The area of each contour is calculated. Considerable-sized objects are marked and analysed. Fig. 3 represents a masked frame. The background is removed. As clear from Fig. 3 the obstacles appear in white. In Fig. 4 contours are plotted on a frame. The green spots which are visible in Fig. 4 are the contours; the area of such spots is calculated to find the size of obstacles.

Contour Area Calculation: According to the discrete version of Green's Theorem [15] bivariate difference calculus provides a general and unifying framework for the description and generation of incremental algorithms. It may be used to compute various statistics about regions bounded by a finite and closed polygonal path. Typically, we use Green's theorem as an alternative way to calculate a line integral CF(ds). If, for example, we are in 2D space, C is a simple closed curve, and F(x, y) is defined everywhere inside C, we can apply Green's theorem to convert the line integral into a double integral. Instead of calculating line integral CF(ds) directly, we calculate the double integral

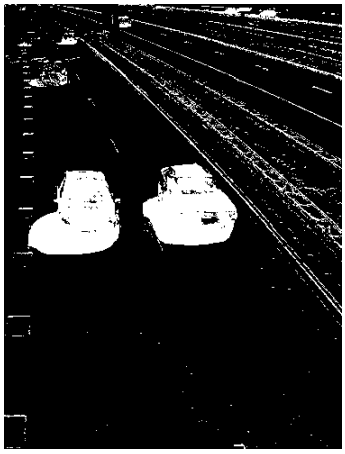


Fig. 3 Masked Frame (Image)

$$D((F2) \times (F1)y)dA \quad (1)$$

If we are given the double integral Df(x, y)dA, we can use Green's theorem only if there happens to be a vector field F(x, y) so that, f(x, y) = (F2)×(F1)y. The area of a region D is equal to the double integral of f(x, y) = 1 over D. Now using this formula we individually calculate the area of all the objects detected in the masked image. The tire sizes that are mostly used range from 185/65 R15 to 265/70R17. Considering the lower dimension, which is 185/65 R15, the diameter of the tire is 19.68 inches (15+ (0.65*185*0.039)). Now keeping the camera resolution and the tire size in mind, any object with a contour area greater than 100 sq. inches is treated as an obstacle. The coordinates of such obstacles in the frame are pushed into an obstacle list.

C. Obstacle Matrix

A 2-D matrix is created, having similar dimensions as that of the image. The coordinates from the obstacle lists are popped out one by one. Then the coordinates of the 2-D matrix which are matching with the popped-out coordinates are marked with "1"s and the rest with "0"s. The same is represented in Fig. 5. Fig. 5(a) shows the obstacles identified in a frame. Fig. 5(b) represents the respective obstacle matrix where obstacles are represented as "1".

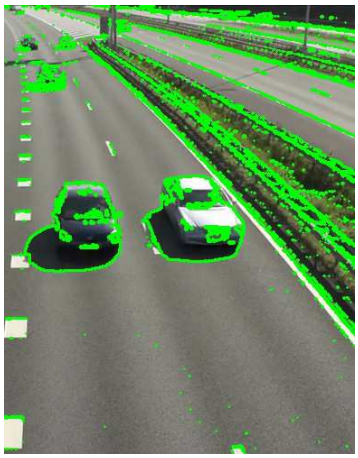
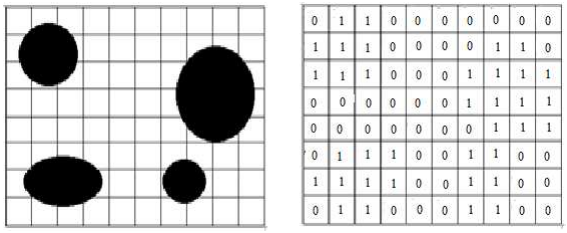


Fig. 4 Contour Plotted Frame (Image)



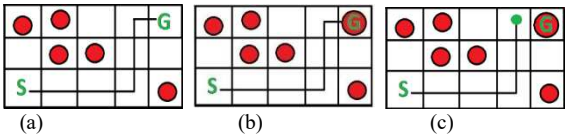
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	1	0
1	1	1	0	0	0	1	1	1	1
0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	1	1	1
0	1	1	1	0	0	1	1	0	0
1	1	1	1	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0	0

(a) (b)

Fig. 5 (a) Detected Obstacles (b) Obstacle Matrix

D. Recursive A* Search Algorithm

When it comes to search algorithms, A* has served a lot of purposes. It is a heuristic method to find path avoiding obstacles. A* has the following functions: g(n): The cost of the path from the start node to the current node n. h(n): The heuristic estimate of the cost from n to the goal. f(n) = g(n) + h(n): The total estimated cost of the cheapest solution through n. A* finds the path between a starting node(S) and a goal node (G). But A* fails in a dynamic environment if the goal node overlaps with any kind of obstacles which was not there previously. Fig. 6(a), (b), and (c) show a grid where red dots are obstacles and 'S' and 'G' are starts and goals respectively.



(a) (b) (c)

Fig. 6 (a) A* Algorithmic Approach, (b) Fail Case of A* Algorithm (c) Recursive A*Algorithmic Approach

In Fig. 6(a), we see that, using A* algorithm we are able to reach the destination. But problem occurs when obstacles reside over goal as seen in Fig. 6(b). In that case A* fails to find the collision free path. So we come up with recursive A* algorithm which goes as close as possible up to the goal as evident from Fig. 6(c). In case of recursive A*, the immediate surroundings of the start node are taken as the goal (pseudo goal). If path is possible then that goal becomes the pseudo start node and its immediate surrounding becomes the goal. This process is continued until the desired goal is reached or at least goes close enough. In this case left and forward direction is preferred for analysing the immediate surroundings. So, RAIL uses A* in a recursive manner. In Recursive-A*(RA*) each next node from the starting node is treated as a pseudo goal until the real goal is reached. This approach helps to get closer to the real goal in case of any obstacle overlap.

E. Imitation Learning

According to Choi Imitation learning (IL) [16] is a machine learning paradigm in which an agent learns a policy by observing and mimicking expert demonstrations, without needing explicit programming or reward signals as in traditional reinforcement learning. It is commonly used in robotics, self-driving cars, and game playing, where learning directly from demonstrations can be more efficient than learning purely from trial and error. The mathematical foundations behind imitation learning can be understood by examining its core components: supervised learning, policy learning, and inverse reinforcement learning. Here we use Generative Adversarial Network (GAN) and Inverse Reinforcement Learning (IRL). GAN is a deep learning architecture that trains two neural networks competing against each other. One network generates new data by taking input data and modifying it as much as possible. The other network checks whether the generated data resembles the original or not. This process of generating and checking is continued until the predicting network fails to distinguish between the fake and original data. Thus, a policy is learned. Reinforcement learning involves an agent, reward function, and policy. The agent learns to maximize rewards in an environment. The reward function (R(s, a, s)) quantifies the

desirability of a state transition (s → s) caused by an action a. The policy ((as)) maps states to actions. In the case of Inverse

Reinforcement Learning there is a given set of trajectories (state-action pairs) = (s1, a1), (s2, a2), . . . = (s1, a1), (s2, a2), . . . generated by an expert's policy * and the goal is to infer the reward function R(s, a, s) that explains *. The input to IRL is a set of expert trajectories:

$$\tau = (s0, a0), (s1, a1) \ldots (sT, aT) \quad (2)$$

The expert's policy (as) maximizes the expected cumulative reward:

$$J(\pi) = E[\sum_{t=0}^{\infty} \gamma^{t} R(st,at)|\pi] \quad (3)$$

$$\pi^{*} = \arg\max_{\pi} J(\pi) \quad (4)$$

$$R^{*}=\text{argmax}_{R}(L(R;\tau)) \quad (5)$$

Here L is likelihood function, measuring how well R explains the expert's behavior. Therefore, IRL is used to uncover the underlying preferences of an expert and to use inferred rewards for training new agents or transferring behavior to different environments. Imitation learning involves imitating the behavior of an expert. It infers the reward function R(s) that explains the expert's behavior using IRL and trains a policy to maximize the inferred reward function. In the case of autonomous vehicles, training an imitation learning model requires a huge training time. So, the RA* is used as the expert trainer whose behaviour is imitated for faster trajectory finding.

IV. ALGORITHM OF RAIL

***Input*:** Real-time video of the road captured by a car

***Output*:** Collision-free trajectory for the real-time scenario

1. Extract Frames are extracted from the video.
2. **While** frames are available in the video **do**
3. Convert the RGB frame into a grayscale frame.
4. Remove the background of the frame and mask the frame.
5. Plot the contours on the masked frame and calculate the contour area.
6. Detect objects with considerable size.
7. Detect the coordinates of pixels as obstacles and append them into an obstacle list *obs*
8. Create a 2-D matrix of similar dimension of the frame and fill that with **0s**
9. **While** *obs* list is not empty **do**
10. Pop coordinates and store in *C*
11. Assign a value *l* to the coordinate of the matrix matching *C*
12. **End While**
13. Set *start, pstart* = (frame_height, frame_height/2)
14. Set *goal* = (frame_width/2, frame_width/2),
15. Set *pgoal* = (frame_height – 1, frame_width/2)
16. Create a list named *path* and Set *i* = 1
17. **While** *pstart*(x coordinate) > goal(x coordinate) **do**
18. Create *closed set* and *open set*
19. *Open set* = nodes to be evaluated, initialized with the start node
20. *Closed set* = nodes that have already been evaluated, initially empty
21. Create a list named *ppath*
22. Initialize *g*(*pstart*) = 0//*g*(*n*): The cost of the path from the start node to node *n*
23. Initialize *g*(other nodes) = ∞
24. Initialize *f*(*pstart*) = *h*(*pstart*) // *h*(*n*): A heuristic function estimating the cost from node *n* to the goal and *f*(*n*) = *g*(*n*) + *h*(*n*): The estimated total cost of the cheapest solution through *n*
25. **While** *Open Set* is not empty **do**
26. Select the node *n* in the *Open Set*

with the smallest $f(n)$ value
27. **If** n is the *pgoal* node **then**
Reconstruct and store the path by tracing back through parent pointers in *ppath*
29. **End If**
30. Remove n from the *Open Set* and add it to the *Closed Set*
31. **For** each neighbor n' of n
32. **If** n' is in the *Closed Set*, skip it
33. Calculate the tentative cost $g\_tentative(n) = g(n) + \text{cost}(n, n')$
34. **If** n' is not in the *Open Set* or $g\_tentative(n) < g(n')$ then
35. Update $g(n') = g\_tentative(n)$, $f(n') = g(n') + h(n')$
36. Set the parent of n' to n
37. **If** n' is not in the *Open Set*, add it to the *Open Set*
38. **End If**
39. **End For**
40. **End While**
41. **If** *Open set* is not empty **then**
42. Update $i = i+1$
43. Reverse *ppath* and append into *path*
44. Update *pstart* = *pgoal,* Update *pgoal* = (frame_height – *i*, frame_width/2)
45. **Else**
46. Update $i = i+1$
47. Update *pgoal* = (frame_height – *i*, frame_width/2)
48. **End While**
49. Train Imitation learning model to learn the environment and path.
50. Display path on the frame.
51. **End While**

V. Result

Table. 1, depicts the success rate achieved by various algorithms on the same test cases in a dynamic environment (where the position of the obstacles is not fixed). It is clear from the table that RAIL is more successful as compared to A* and Dijkstra's algorithms. Table. 2. shows the execution time taken by the different algorithms in different grid sizes such as 25×25, 100×100, 250×250 and 500×500. With respect to execution time, the proposed methodology executes faster compared to other conventional approaches. Fig. 7 reflects a bar plot to compare between RAIL and A*. Dijkstra's algorithm in not considered in this figure as it has a huge execution time as compared to the other two.

Table. 1. Success Rate comparison with other collision free path estimation algorithms.

Sl. No.	Algorithm	Success Rate
1	Dijkstra	0.0001(0.01 %)
2	A*	0.1(10%)
3	RAIL	0.812(81.2)

Table. 2. Time Efficiency Comparison with other collision free path estimation algorithms

Algorithm	Grid Size (rows x columns)	Execution Time(in ns)
Dijkstra	(25×25)	621
	(100×100)	9972
	(250×250)	62430
	(500×500)	249999
A*	(25×25)	49
	(100×100)	199
	(250×250)	499
	(500×500)	999
RAIL	(25×25)	48
	(100×100)	198
	(250×250)	498
	(500×500)	998

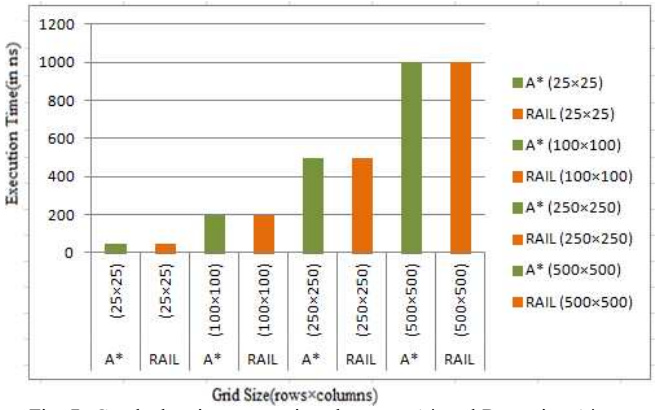


Fig. 7 Graph showing comparison between A* and Recursive A*

VI. Conclusion

Usage of a graph-based algorithm such as A* in a dynamic environment is not feasible. But using it in an imitation learning model is something new. We imitate the behavior of A* algorithm and implement it in real time. Collision-free path planning and autonomous vehicle navigation are rapidly evolving fields. Advances in algorithms, sensor technologies, and machine learning are driving improvements in safety, efficiency, and robustness. Continued research and development are needed to address remaining challenges and enable broader adoption of autonomous vehicles.

References

[1] Edsger W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, (1959)

[2] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. Procedia Computer Science, 179:507–514, (2021).

[3] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*- smart path planning algorithms. International Journal of Computer Science and Network Security (IJCSNS), 16(10):20, (2016).

[4] Cheng, R.; Orosz, G.; Murray, R.M.; Burdick, J.W. End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks. Proc. Aaai Conf. Artif. Intell. 33, 3387–3395. (2019).

[5] Tessler, C.; Mankowitz, D.J.; Mannor, S. Reward constrained policy optimization, (2018).

[6] Achiam, J.; Held, D.; Tamar, A.; Abbeel, P. Constrained Policy Optimization. In Proceed- ings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August, pp. 22–31. (2017).

[7] Gu, S.; Kuba, J.G.; Chen, Y.; Du, Y.; Yang, L.; Knoll, A.; Yang, Y. Safe Multi-Agent Re- inforcement Learning for Multi-Robot Control. Artif. Intell. , 319, 103905. (2023).

[8] Du, D.; Han, S.; Qi, N.; Ammar, H.B.; Wang, J.; Pan, W. Reinforcement Learning for Safe Robot Control Using Control Lyapunov Barrier Functions. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, (2023).

[9] Zeng, J.; Zhang, B.; Sreenath, K. Safety-Critical Model Predictive Control with Discrete- Time Control Barrier Function. In Proceedings of the 2021 American Control Conference (ACC), New Orleans, LA, USA, (2021).

[10] Thirugnanam, A.; Zeng, J.; Sreenath, K. Safety-Critical Control and Planning for Obstacle Avoidance between Polytopes with Control Barrier Functions. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA) 2022, Philadelphia, PA, USA, (2022).

[11] Xue, H.; Lai, Y.H.; Sun, K. Human-like constraint-adaptive model predictive control with risk-tunable control barrier functions for autonomous ships. Ocean. Eng. (2024).

[12] Cohen, M.H.; Belta, C. Safe Exploration in Model-Based Reinforcement Learning Using Control Barrier Functions. Automatica 2023, 147, 110684.

[13] Junqing Wei, Jarrod M Snider, Tianyu Gu, John M Dolan, and Bakhtiar Litkouhi. A behavioral planning framework for autonomous driving. In 2014 IEEE Intelligent Vehicles Symposium Proceedings, pages 458–464. IEEE, (2014).

[14] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learn- ing. Advances in neural information processing systems, 29, (2016).

[15] Srecko Brlek, Gilbert Labelle, and Annie Lacasse. The discrete green theorem and some applications in discrete geometry. Theoretical Computer Science, 346(2-3):200– 225, (2005).

[16] Seongjin Choi, Jiwon Kim, and Hwasoo Yeo. Trajgail: Generating urban vehicle trajectories using generative adversarial imitation learning. Transportation Research Part C: Emerging Technologies, 128:103091, (2021).