

# Robust Unmanned Surface Vehicle Navigation with Distributional Reinforcement Learning

Xi Lin, John McConnell and Brendan Englot

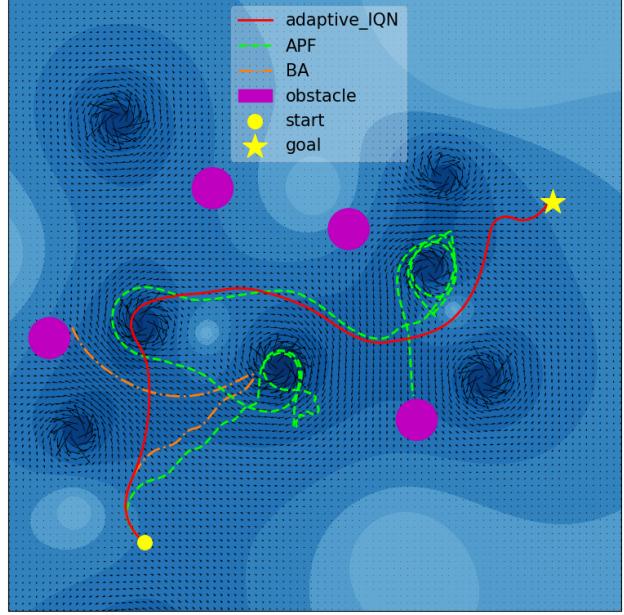
**Abstract**— Autonomous navigation of Unmanned Surface Vehicles (USV) in marine environments with current flows is challenging, and few prior works have addressed the sensor-based navigation problem in such environments under no prior knowledge of the current flow and obstacles. We propose a Distributional Reinforcement Learning (RL) based local path planner that learns return distributions which capture the uncertainty of action outcomes, and an adaptive algorithm that automatically tunes the level of sensitivity to the risk in the environment. The proposed planner achieves a more stable learning performance and converges to safer policies than a traditional RL based planner. Computational experiments demonstrate that comparing to a traditional RL based planner and classical local planning methods such as Artificial Potential Fields and the Bug Algorithm, the proposed planner is robust against environmental flows, and is able to plan trajectories that are superior in safety, time and energy consumption.

## I. INTRODUCTION

The operation of autonomous vehicles in marine environments is sensitive to currents [1], and navigating safely and efficiently under the influence of current flow is challenging. In recent years, several methods [2], [3], [4] have been proposed to deal with the global path planning problem for Unmanned Surface Vehicles (USV) given knowledge of the current flow field. However, few prior works have focused on the local path planning problem for USVs in environments with unknown current flows. In this work, inspired by recent USV competitions [5] and by the challenges of navigating in river rapids, we consider the sensor-based local navigation problem for USVs in simulated marine environments with no prior knowledge of the current flow and obstacles.

Reinforcement Learning (RL) shows the ability to acquire high-performance policies to operate in unseen environments by learning through experiences of interaction with training environments given no prior information [6]. In the last decade, Deep Reinforcement Learning (DRL) combining RL with deep neural network architectures has been widely used to solve practical problems with high-dimensional sensory inputs, which can be efficiently executed via neural network inference. Compared to a traditional DRL method that only learns the expected return, a Distributional Reinforcement Learning (Distributional RL) method is shown to provide a more stable learning behavior in environments with high uncertainty as it learns return distributions [7]. In addition, risk measures can be applied to adjust the level of sensitivity

Xi Lin, John McConnell and Brendan Englot are with the Department of Mechanical Engineering, Stevens Institute of Technology, 1 Castle Point Terrace, Hoboken, NJ 07030, USA, {xlin26, jmcconn1, benglot}@stevens.edu



**Fig. 1: Comparison of planned trajectories.** Planned trajectories of the proposed Distributional RL based path planner and classical planners in a simulated marine environment with no prior knowledge of the current flow field and obstacles, where darker color indicates faster current flows. Arrows show the velocity of the current in specific locations.

to aleatoric uncertainty in learned distributions and enhance the safety performance of a Distributional RL agent [8].

We propose a Distributional RL based local path planner to address USV sensor-based navigation tasks under unknown obstacles and current flows, as well as an algorithm that adjusts the level of sensitivity towards collision risk and further improves the task success rate. The proposed method exhibits a more stable learning performance and converges to safer policies compared to a traditional DRL-based local planner. We compare the proposed method's performance to classical methods including Artificial Potential Fields [9], [10] and the Bug Algorithm [11], [12], which have been applied to solve the USV local path planning problem in stable marine environments with minimal current disturbances. It is shown in Figure 1 that the performances of Artificial Potential fields and the Bug Algorithm are vulnerable to unknown current flows, and result in unsafe, zig-zag trajectories. On the contrary, the proposed Distributional RL planner is able to plan a much smoother trajectory under current disturbances while keeping a safe distance from detected obstacles, thus

achieving superior performance in safety, time and energy consumption. Our contributions are summarized as follows:

- To our knowledge, the first Distributional RL based path planner for USV sensor-based navigation in environments with unknown current flows and obstacles.
- Simulated experiments that show superior performance in safety, time and energy consumption vs. traditional RL and classical reactive planning algorithms.
- The software implementation of our approach and a simulation environment for studying decision making in USV navigation (amidst unknown currents and obstacles) have been made freely available at [https://github.com/RobustFieldAutonomyLab/Distributional\\_RL\\_Navigation](https://github.com/RobustFieldAutonomyLab/Distributional_RL_Navigation).

The rest of this paper is organized as follows: Section II introduces related work, highlighting relevant local path planning methods; Section III describes the RL problem formulation; Section IV introduces the methodology of this work; Section V introduces baseline approaches and discusses experimental results; Section VI concludes the paper.

## II. RELATED WORKS

Khatib [13] proposed to perform mobile robot navigation with an Artificial Potential Field (APF), which creates attractive force towards the goal and repulsive force to avoid obstacles. However, the traditional APF method may fail when local minima are present, or the goal is close to obstacles, and solutions such as virtual local target [14], deterministic annealing [15], dynamic window [16], and additional repulsive force [17], [18] were proposed to address this problem. Some works focus on improving performance of the APF method in dynamic environments, by using a new potential function that considers the relative position and velocity of the robot with respect to the target and obstacles [19], or evolutionary algorithms [20], [21].

Bug Algorithms (BA) have been used for reactive mobile robot path planning since the 1980s [22]. Lumelsky et al. [23] proposed the Bug1 and Bug2 algorithms, which use a point robot model and zero-range sensor for obstacle detection, and exhibit two behaviors: the robot moves in a straight line to the goal and follows the obstacle boundary as soon as it encounters an obstacle. The Bug2 algorithm has been applied to the local navigation problem on platforms such as wheeled robots [24], quadrotors [25] and USVs [12]. Alg1 and Alg2 [26], Rev1 and Rev2 [27] improve the performance of bug algorithms by switching the obstacle following direction when the robot is in a previously visited location. VisBug [28], [29] and TangentBug [30] use range sensors for obstacle detection and find shorter paths to the goal [12].

Deep Reinforcement Learning (DRL) based planners have shown strong ability to navigate in unknown environments in recent years. Tai et al. [31] trained a DRL agent for navigation in unseen indoor environments without any collisions. Zhang et al. [32] and Josef et al. [33] showed robust performance of DRL planners on Unmanned Ground Vehicle (UGV) navigation in simulated unknown rough terrain where the surface normal could abruptly change. Cheng et al.

[34] proposed a DRL based path planner to control an underactuated USV under unknown environment dynamics. Xu et al. [35] developed a DRL agent for a USV to avoid dynamic obstacles such as boats.

More recently, Distributional RL [7] methods have been used to learn risk-aware navigation policies for enhanced safety. Choi et al. [36] proposed a novel Distributional RL algorithm for indoor navigation tasks, which shows superior performance in safety relative to traditional DRL methods, as well as adjustable sensitivity towards collision risks. Kamran et al. [37] demonstrated that a Distributional RL based policy reduces driving time compared to traditional DRL methods, and requires much less safety interference than rule-based policies in autonomous driving scenarios. Liu et al. [38] proposed an algorithm that automatically adjusts the level of sensitivity toward risk of a Distributional RL planner for nano drone navigation.

## III. PROBLEM FORMULATION

We formulate the USV navigation problem as a Markov Decision Process  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ , where  $\mathcal{S}$  and  $\mathcal{A}$  are the sets of states and actions. At each time step  $t$ , the agent receives an observation of the current state  $s_t \in \mathcal{S}$ , and selects an action  $a_t \in \mathcal{A}$ . This causes the agent to transition to the state  $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$ , and the agent receives an observation of  $s_{t+1}$ , as well as a reward  $r_{t+1} = R(s_{t+1}, a_{t+1})$ . A common way to find an optimal policy in RL is to maximize the expected future return  $Q^\pi$ .

$$Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \quad (1)$$

This is also known as the action-value function, which starts with a state-action pair  $(s_t, a_t)$  and follows the policy  $\pi$  thereafter. Discount factor  $\gamma \in [0, 1]$  controls the effect of future rewards.

$$Q^\pi(s, a) = \mathbb{E}[R(s, a)] + \gamma \mathbb{E}[Q^\pi(s', a')] \quad (2)$$

$$\mathcal{T}Q(s, a) := \mathbb{E}[R(s, a)] + \gamma \mathbb{E}[\max_{a'} Q(s', a')] \quad (3)$$

The relation between  $Q^\pi$  and its successors satisfies the Bellman equation (Eq. (2)), where  $s' \sim \mathcal{P}(\cdot|s, a)$ , and  $a' \sim \pi(\cdot|s')$ . An optimal policy can be derived using the Bellman optimality operator (Eq. (3)).

## IV. METHODOLOGY

### A. Traditional DRL

We choose DQN [39] as the traditional DRL baseline, and use the implementation from the Stable-Baselines3 project [40]. DQN parameterizes an approximate action-value function (Eq. (1)) as  $Q(s, a; \theta)$  with a neural network model, and learns an optimal policy by performing optimization on a loss function (Eq. (4)) based on Temporal Difference (TD) error, where  $(s, a, r, s')$  are samples from experiences.

$$\mathcal{L}_{\text{DQN}} = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \quad (4)$$

## B. Distributional RL

Instead of the expected return (Eq. (1)), Distributional RL algorithms [7] focus on the return distribution, which satisfies the distributional Bellman equation (Eq. (5)), where  $Z^\pi(s, a)$  is a random variable that satisfies  $Q^\pi(s, a) = \mathbb{E}[Z^\pi(s, a)]$ . Similarly, the distributional Bellman optimality equation (Eq. (6)) is used in this case.

$$Z^\pi(s, a) \stackrel{D}{=} R(s, a) + \gamma Z^\pi(s', a') \quad (5)$$

$$\mathcal{T}Z(s, a) \stackrel{D}{=} R(s, a) + \gamma Z^\pi(s', \text{argmax}_a \mathbb{E}[Z(s', a')]) \quad (6)$$

The proposed Distributional RL based path planner uses Implicit Quantile Networks (IQN) [8], and we use the implementation from [41]. IQN uses the quantile function  $Z$ , denoted as  $Z_\tau := F_Z^{-1}(\tau)$ , where  $\tau \sim U([0, 1])$ , and a distortion risk measure  $\beta : [0, 1] \rightarrow [0, 1]$  to define a distorted expectation.

$$Q_\beta(s, a) = \mathbb{E}_{\tau \sim U([0, 1])}[Z_{\beta(\tau)}(s, a)] \quad (7)$$

Then the risk-sensitive policy can be expressed as  $\pi_\beta(s) = \text{argmax}_a Q_\beta(s, a)$ , and approximated by  $K$  samples of  $\tilde{\tau} \sim U([0, 1])$ :

$$\tilde{\pi}_\beta(s) = \text{argmax}_a \frac{1}{K} \sum_{k=1}^K Z_{\beta(\tilde{\tau}_k)}(s, a). \quad (8)$$

The IQN loss function (Eq. (11)) is constructed with the sampled temporal difference (TD) error (Eq. (9)) and the quantile Huber loss (Eq. (10)).

$$\delta^{\tau_i, \tau'_j} = r + \gamma Z_{\tau'}(s', \pi_\beta(s')) - Z_\tau(s, a) \quad (9)$$

$$\rho_\tau^\kappa(u) = |\tau - \delta_{\{u < 0\}}|(\mathcal{L}_\kappa(u)/\kappa),$$

$$\text{where } \mathcal{L}_\kappa(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \leq \kappa \\ \kappa(|u| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases} \quad (10)$$

$$\mathcal{L}_{\text{IQN}} = \frac{1}{N'} \sum_{i=1}^N \sum_{j=1}^{N'} \rho_{\tau_i}^\kappa(\delta^{\tau_i, \tau'_j}) \quad (11)$$

## C. USV Navigation Environment

We design a simulated marine environment with environmental flows and static obstacles, where the robot is required to navigate under flow disturbances and reach the goal without colliding with any obstacles.

The Rankine vortex model [42] (Eq. (12)) is used to create the flows, where a rigid body rotation within the vortex core of radius  $r_0$  is assumed, and  $\Gamma$  is the circulation strength of the vortex.

$$v_r = 0, \quad v_\theta(r) = \frac{\Gamma}{2\pi} \begin{cases} r/r_0^2, & \text{if } r \leq r_0 \\ 1/r, & \text{if } r > r_0 \end{cases} \quad (12)$$

The angular velocity of the vortex core  $\Omega = \Gamma/(2\pi r_0^2)$ , and the linear velocity at the edge of vortex core  $v_{\text{edge}} = \Omega r_0$ . The flow velocity at a specific location is approximated by superimposing the effects of nearby vortices.

We use a simplified USV model, the pose of which can be represented as  $(x, y, \theta)$ , where  $(x, y)$  are the global Cartesian coordinates of the robot, and  $\theta$  is its orientation. To consider

the effects of current flow on robot motion, the kinematic model described in [43] is used, which treats the robot as a point particle, since its dimension is much smaller than the length of its trajectory and the scale of current flow.

$$\frac{d\mathbf{X}(t)}{dt} = \mathbf{V}(t) = \mathbf{V}_C(\mathbf{X}(t)) + \mathbf{V}_S(t) \quad (13)$$

In Eq. (13),  $\mathbf{V}(t)$  is the total velocity of the robot,  $\mathbf{V}_C(\mathbf{X}(t))$  is the current flow velocity at robot position  $\mathbf{X}(t)$ , and  $\mathbf{V}_S(t)$  is the robot steering velocity.

## D. States and Observations

The full state consists of robot position  $\mathbf{X}(t)$ , robot steering velocity  $\mathbf{V}_S(t)$ , current flow field  $\mathbf{V}_C(\mathbf{X})$ , the positions and sizes of all obstacles, and the goal location. In this work, the robot only receives a partial observation of the full state from onboard sensors,  $O_t = (O_{\text{velocity}}, O_{\text{goal}}, O_{\text{LiDAR}})$ , where  $O_{\text{velocity}}$  is the robot's seafloor-relative velocity measured by Doppler Velocity Log (DVL),  $O_{\text{goal}}$  is the goal position in the robot frame computed with the aid of an inertial measurement unit (IMU), compass and GPS, and  $O_{\text{LiDAR}}$  are LiDAR reflections indicating any obstacles ahead. The LiDAR range is  $d_0 = 10$  meters. These assumptions are inspired by the class of USV required to successfully detect and avoid obstacles in the Maritime RobotX competition [44]. An observation example is shown in Fig. 5.

## E. Actions

The action used to control the robot motion is the change in steering velocity  $\mathbf{V}_S$ . We assume that the rates of change in both the magnitude and direction of  $\mathbf{V}_S$  are constant across one control time step, hence an action is given in the form of  $a_t = (a, w)$ , where  $a$  is the rate of change in velocity magnitude, and  $w$  is the rate of change in velocity direction. For convenience,  $a$  and  $w$  are referred to as linear acceleration and angular velocity. We use  $a \in \{-0.4, 0.0, 0.4\}$  m/s<sup>2</sup>, and  $w \in \{-0.52, 0.0, 0.52\}$  rad/s. The forward speed is clipped to  $[0, v_{\text{max}}]$ .

## F. Reward

The reward function is designed to encourage the learning agent to move towards the goal while avoiding obstacles, which is shown in Equation (14).

$$r_t = \begin{cases} r_{\text{base}, t} + r_{\text{collision}}, & \text{if collide at } t \\ r_{\text{base}, t} + r_{\text{goal}}, & \text{if reach goal at } t \\ r_{\text{base}, t} = r_{\text{step}} + \alpha(d_{t-1} - d_t), & \text{otherwise} \end{cases} \quad (14)$$

$d_t$  represents the distance between the robot and the goal at  $t$ . We use  $r_{\text{step}} = -1.0$ ,  $r_{\text{collision}} = -50.0$ ,  $r_{\text{goal}} = 100.0$ , and  $\alpha = 1.0$  in our computational experiments.

## G. Network Architecture

The network architectures used by our DQN and IQN agents are shown in Figure 2. Observations from different sources are encoded separately as different features, which are then concatenated into a general one. Compared to the DQN network model, IQN employs an additional cosine function to embed the quantile input [8], and we use

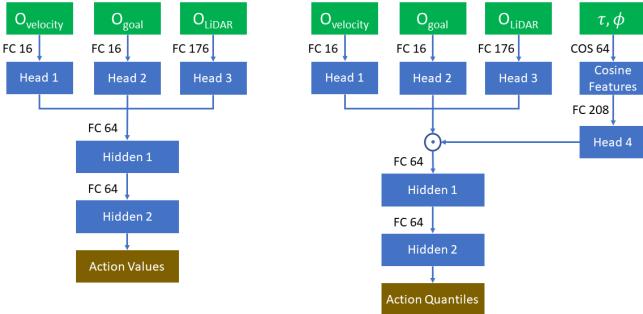


Fig. 2: **Network models.** Architectures used by DQN (left) and IQN (right). FC, COS and  $\odot$  stand for fully connected layer, cosine embedding layer and element-wise product.

TABLE I: **Hyperparameters** used by IQN and DQN during the learning process.

Parameter	Learning rate	Buffer size	Mini-batch size	Discount factor
Value	$1 \times 10^{-4}$	$1 \times 10^6$	32	0.99

conditional value-at-risk (CVaR, Eq. (15)) as the distortion function. The resulting cosine features are shown in Eq. (16).

$$f(\tau; \phi) = \phi\tau, \quad \phi \in (0, 1], \quad \tau \sim U([0, 1]) \quad (15)$$

$$[\cos(\pi \cdot 0 \cdot f(\tau; \phi)), \dots, \cos(\pi \cdot 63 \cdot f(\tau; \phi))] \quad (16)$$

During the training process,  $\phi = 1.0$ , the number of samples in Eq. (11) is  $N = N' = 8$ , and the number of samples in Eq. (8) is  $K = 32$ . Given an observation, outputs of the DQN network are action values corresponding to actions, and those of the IQN network are sets of quantile points that reflect the return distributions of their respective actions.

Both agents use  $\epsilon$ -greedy as their behavior policy. The control time step of an action is 1.0 seconds. The exploration rate starts at 1.0, and decreases linearly to 0.05 during the initial 10% of total training time steps, then stays at 0.05 thereafter. Other hyperparameters shared by both agents are shown in Table I.

### H. Model Training

As we focus on the local path planning problem, all simulation environments are of the size  $50m \times 50m$ . We employ the idea of curriculum training and gradually increase the level of difficulty in the training environment using the schedule shown in Table II. Example environments used during each phase are shown in Figure 3.

When initializing a training episode, vortices of random position, spinning direction, and strength, as well as circular obstacles of random position and size are generated in the training environment. The linear velocity at the edge of vortex core,  $v_{edge}$ , and the radius of obstacle are sampled uniformly from  $[5, 10]m/s$  and  $[1, 3]m$  respectively. The start and goal positions are randomly generated such that the distance between them is not smaller than the given threshold. The initial robot pose and forward speed are also randomly generated. An episode is terminated when the robot collides

TABLE II: **Training Schedule.** Environment hyperparameters used in curriculum training.

Parameter	1st million	2nd million	3rd million
Number of vortices	4	6	8
Number of obstacles	6	8	10
Min distance between start and goal	30.0	35.0	40.0

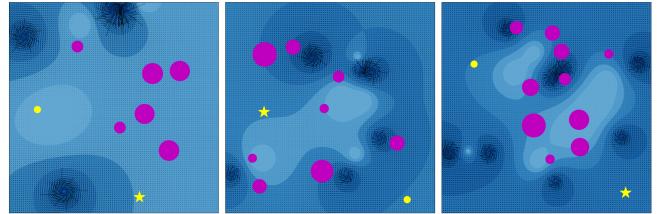


Fig. 3: **Training environments.** Examples of training environments randomly generated for use during the 1st, 2nd and 3rd million steps (shown from left to right).

with any obstacles, reaches the goal, or has moved for more than 1,000 steps, then a new episode is created.

To assess agents' performances during the training, thirty random environments are created in advance, which consist of three sets of ten environments generated according to parameters shown in each column in Table II, except the min distance between start and goal. Every evaluation environment uses the same start and goal located respectively at the lower left and top right of the map. Every agent is evaluated on all thirty environments after every 10,000 steps of training. For both IQN and DQN, we train thirty models with different random seeds, and visualize the overall learning performance in Figure 4.

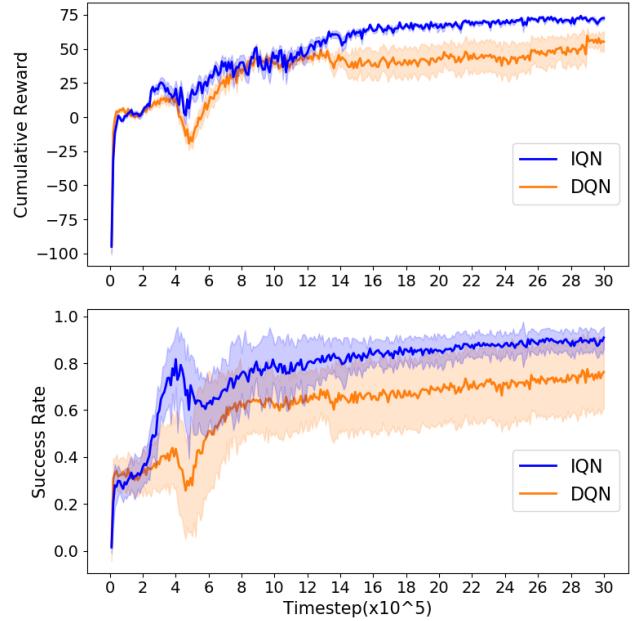


Fig. 4: **Learning performances of IQN and DQN.** In each curve, the solid line and bandwidth represent the mean and standard error of evaluation results over thirty models trained with different seeds.

The training processes of all models were run on an Nvidia RTX 3090 GPU. It can be seen that IQN exhibits a more stable learning performance and has higher scores in terms of cumulative reward and success rate.

## V. EXPERIMENTS

To obtain a more comprehensive understanding of the performance of our trained agents, two sets of evaluation experiments using the number of vortexes and obstacles corresponding to the lowest and highest level of difficulty in Table II are performed, denoted as Test Case 1 and Test Case 2, and the results are summarized in Table III. Each set of experiments uses 500 randomly generated environments, and similar to the evaluation environments during the training process, each environment has a fixed start and goal located respectively at the lower left and top right of the map. The control time step of an action is set to 0.5 seconds. A square boundary is set in each environment, and the robot moving outside the bounded area is considered an out of bounds failure. The energy consumption is computed by summing up the magnitude of all action commands.

IQN and DQN agents used in the evaluation experiments are randomly selected from trained models. We evaluate the IQN agent with fixed CVaR threshold values  $\phi = \{0.25, 0.50, 0.75, 1.0\}$  that represent different levels of sensitivity towards risk. In addition, we also try an adaptive function of  $\phi$  shown in Eq. (17).

$$\phi = \begin{cases} \min(d(X, X_O))/d_0, & \text{if } \min(d(X, X_O)) \leq d_0 \\ 1.0, & \text{if } \min(d(X, X_O)) > d_0 \end{cases} \quad (17)$$

$X$  and  $X_O$  are positions of the robot and all obstacles. Thus the adaptive framework leads to a greedy policy when no obstacles are detected, and a risk sensitive policy (with the level of sensitivity being proportional to the minimum distance to obstacles) if any obstacles exist.

### A. Baseline approaches

We also choose two classical local planning methods as baselines. The first one is an Artificial Potential Field (APF) method described in [18]. An attractive potential field  $U_{\text{att}}$  and a repulsive potential field  $U_{\text{rep}}$  are constructed to generate forces that lead the robot to the goal and repel it from obstacles.

$$U_{\text{att}}(X) = \frac{1}{2} k_{\text{att}} \cdot d^2(X, X_g) \quad (18)$$

$$U_{\text{rep}}(X) = \begin{cases} U_o(X), & \text{if } d(X, X_o) \leq d_0 \\ 0, & \text{if } d(X, X_o) > d_0 \end{cases} \quad (19)$$

$$\text{where } U_o(X) = \frac{1}{2} k_{\text{rep}} \left( \frac{1}{d(X, X_o)} - \frac{1}{d_0} \right)^2 d^n(X, X_g)$$

In the above equations,  $X$ ,  $X_g$  and  $X_o$  are positions of the robot, goal and an obstacle respectively, and we use  $k_{\text{att}} = 50.0$ ,  $k_{\text{rep}} = 500.0$ , and  $n = 2$ . The total force  $F = -\nabla U_{\text{att}}(X) - \nabla U_{\text{rep}}(X)$ . Given an observation, each LiDAR reflection point is treated as an obstacle, and the repulsive force is the sum of contributions from all points. To output an action decision that is compatible with the robot, we compute the difference in angle between the total force

$F$  and the robot velocity, and map it to the angular velocity that causes the closest change in angle in one control step. We also project  $F$  to the direction of robot velocity, scale it by  $1/m$  with  $m = 500.0$ , and map it to the closest linear acceleration.

The second baseline is a Bug Algorithm (BA) method similar to VisBug [28]. It uses a simple navigation policy: when the robot hits an obstacle, it moves by following the boundary; as soon as the way to the goal is clear, the robot moves directly towards it. Given an observation, the tangent vector of the obstacle surface is approximately computed from LiDAR reflections, and the robot steers itself and moves in the same direction while keeping a safe standoff distance of 5 meters. Similarly, the angular velocity that minimizes the angle difference between desired steering direction and the robot velocity in one control step is chosen. The linear acceleration is selected such that the robot maintains a low speed when moving along obstacles, and the max speed when moving towards the goal.

### B. Results

It can be seen in Table III that all IQN agents except  $\phi = 0.25$  achieve a higher success rate than other methods, with lower average time and energy consumption. In addition, as the difficulty of environment increases, IQN agents show more robust performance in terms of success rate, with no significant increase in time and energy consumption. It can be seen in Figure 1 that the trajectories of APF and BA agents are highly affected by flow disturbances exerted by vortices. Hence compared to DRL agents, APF and BA have clearly higher time and energy consumption, especially in Test Case 2. The lower success rate in the case of  $\phi = 0.25$  is related to the significantly higher out of bounds rate, indicating that the IQN agent is so conservative that it tries moving far enough to avoid any risks. The adaptive IQN agent has a higher success rate than other IQN agents, with similar time and energy consumption.

Figure 5 visualizes a state where obstacles are detected and compares decisions made by the adaptive IQN agent and the greedy ( $\phi = 1.0$ ) IQN agent. It can be seen from the observation plot that the goal direction is close to the steer direction as well as the velocity direction, and LiDAR reflections show a clearance between two sets of obstacles. Hence sticking with the current direction may get to the goal quickly, which is also the action selected by the greedy IQN agent. The adaptive IQN agent lowers the CVaR threshold to increase the risk sensitivity, and focuses on the worst part of the whole distribution. In this case, turning left is a better choice as it has a higher expected return value in the tail distribution. Additionally, the trajectories of the greedy and adaptive IQN agents overlap initially, since the latter one maintains  $\phi = 1.0$  and efficiently navigates to the goal when no obstacles are detected.

The computation time per action decision for all methods is shown in Table IV. All experiments in this section were run on a AMD Ryzen threadripper 3970X CPU. Compared to the classical methods, IQN and DQN models require more

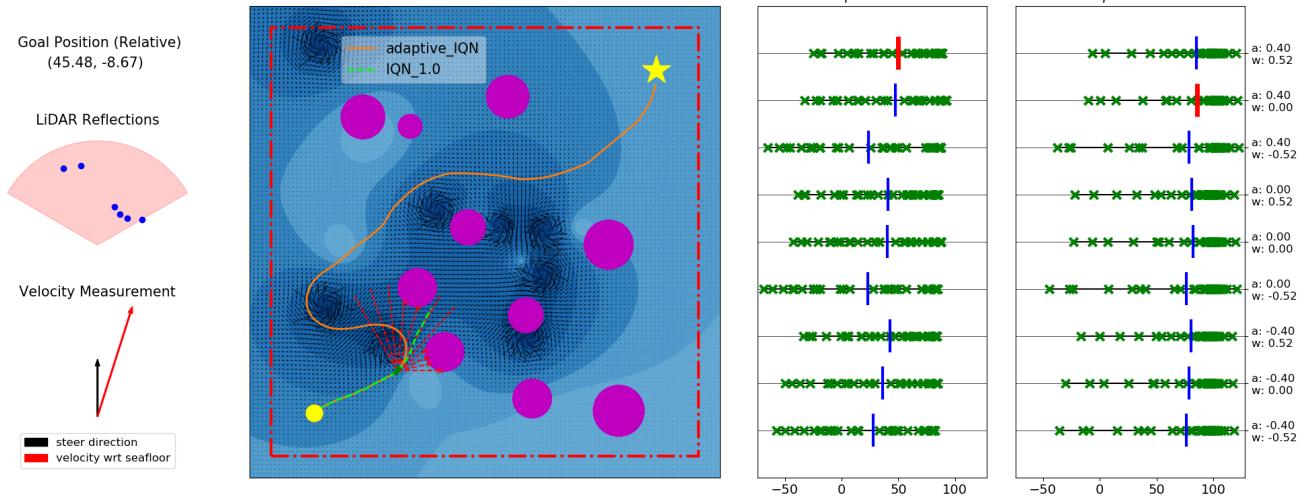


Fig. 5: **An example state where adaptive IQN agent and greedy ( $\phi = 1.0$ ) IQN agent make different decisions.** Observation inputs are shown at left. Trajectories of the two agents are shown in the map, where red dashed lines represent beams emitted by onboard LiDAR, and the red dashdot square is the boundary of the evaluation environment. In the distribution plots at right, green “x” markers show return values corresponding to different quantiles, and vertical lines indicate the mean value (red lines mark action selections).

TABLE III: **Experimental Results.** Out of bounds rate is the ratio of the number of out of bounds failures to that of all episodes. Average time and energy are computed using the data of successful episodes only.

(a) **Test Case 1** (4 vortices and 6 obstacles)

Agent		success rate	out of bounds rate	average time (s)	average energy
IQN	adaptive	0.95	0.01	35.32	85.34
	$\phi = 0.25$	0.87	0.11	36.80	86.46
	$\phi = 0.50$	0.94	0.02	35.35	83.62
	$\phi = 0.75$	0.94	0.02	35.43	83.14
	$\phi = 1.0$	0.94	0.01	34.73	82.80
	DQN	0.88	0.02	35.58	106.16
APF	APF	0.90	0.01	44.58	124.47
	BA	0.66	0.00	42.31	104.35

(b) **Test Case 2** (8 vortices and 10 obstacles)

Agent		success rate	out of bounds rate	average time (s)	average energy
IQN	adaptive	0.81	0.04	36.76	93.36
	$\phi = 0.25$	0.60	0.29	39.38	97.78
	$\phi = 0.50$	0.76	0.09	37.39	92.85
	$\phi = 0.75$	0.78	0.07	35.90	88.69
	$\phi = 1.0$	0.76	0.03	35.42	87.94
	DQN	0.61	0.04	34.67	106.19
APF	APF	0.58	0.07	53.72	165.03
	BA	0.37	0.01	49.73	133.15

computation in one forward pass, but their mean runtimes per action are still below 0.4 milliseconds. The maximum runtime per action of adaptive IQN is 13.12 milliseconds, which still allows a 75 Hz computation frequency.

## VI. CONCLUSION

In this paper, we propose an IQN based local path planner for sensor-based Unmanned Surface Vehicle (USV) navigation in marine environments with no prior knowledge of the

TABLE IV: **Runtime per action.** The mean and maximum runtime to compute an action decision.

Agent	mean (ms)	max (ms)
IQN	adaptive	0.31
	$\phi = 0.25$	0.28
	$\phi = 0.50$	0.28
	$\phi = 0.75$	0.28
	$\phi = 1.0$	0.28
	DQN	0.19
APF	APF	0.068
	BA	0.049
		1.81

current flow field and obstacles. Compared to a DQN based planner, the proposed method has a more stable learning performance and higher scores in the cumulative reward and success rate during the curriculum training process. Experimental results show that the proposed planner achieves superior performance in safety, time and energy consumption relative to other planners based on DQN, Artificial Potential Fields, and the Bug Algorithm. In future work, we hope to explore the multi-USV navigation problem with a similar approach, and the extent to which the policies learned in our simulator across different random obstacle and vortex fields can transfer to USV navigation with real robot hardware.

## ACKNOWLEDGMENT

This research was supported by the Office of Naval Research, grants N00014-20-1-2570 and N00014-21-1-2161.

## REFERENCES

- [1] T. Lolla, P. Haley Jr, and P. Lermusiaux, “Path planning in multi-scale ocean flows: Coordination and dynamic obstacles,” *Ocean Modelling*, vol. 94, pp. 46–66, 2015.
- [2] R. Song, Y. Liu, and R. Bucknall, “A multi-layered fast marching method for unmanned surface vehicle path planning in a time-variant maritime environment,” *Ocean Engineering*, vol. 129, pp. 301–317, 2017.

- [3] X. Guo, M. Ji, Z. Zhao, D. Wen, and W. Zhang, "Global path planning and multi-objective path control for unmanned surface vehicle based on modified particle swarm optimization (PSO) algorithm," *Ocean Engineering*, vol. 216, no. 107693, 2020.
- [4] J. Meng, Y. Liu, R. Bucknall, W. Guo, and Z. Ji, "Anisotropic GPMP2: a fast continuous-time Gaussian processes based motion planner for unmanned surface vehicles in environments with ocean currents," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3914–3931, 2022.
- [5] J. Woo, J. Lee, and N. Kim, "Obstacle avoidance and target search of an Autonomous Surface Vehicle for 2016 Maritime RobotX challenge," in *Proceedings of IEEE Underwater Technology (UT)*, 2017.
- [6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [7] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 449–458.
- [8] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 1096–1105.
- [9] J. Song, C. Hao, and J. Su, "Path planning for unmanned surface vehicle based on predictive artificial potential field," *International Journal of Advanced Robotic Systems*, vol. 17, no. 2, 2020.
- [10] H. Sang, Y. You, X. Sun, Y. Zhou, and F. Liu, "The hybrid path planning algorithm based on improved A\* and artificial potential field for unmanned surface vehicle formations," *Ocean Engineering*, vol. 223, no. 108709, 2021.
- [11] T. Wilson and S. B. Williams, "Adaptive path planning for depth-constrained bathymetric mapping with an autonomous surface vessel," *Journal of Field Robotics*, vol. 35, no. 3, pp. 345–358, 2018.
- [12] D. V. Lyridis, "An improved ant colony optimization algorithm for unmanned surface vehicle local path planning with multi-modality constraints," *Ocean Engineering*, vol. 241, no. 109890, 2021.
- [13] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.
- [14] G. Li, A. Yamashita, H. Asama, and Y. Tamura, "An efficient improved artificial potential field based regression search method for robot path planning," in *Proceedings of the IEEE International Conference on Mechatronics and Automation*, 2012, pp. 1227–1232.
- [15] N. S. F. Doria, E. O. Freire, and J. C. Basilio, "An algorithm inspired by the deterministic annealing approach to avoid local minima in artificial potential fields," in *Proceedings of the 16th International Conference on Advanced Robotics (ICAR)*, 2013, pp. 1–6.
- [16] J. Sun, G. Liu, G. Tian, and J. Zhang, "Smart obstacle avoidance using a danger index for a dynamic environment," *Applied Sciences*, vol. 9, no. 8, p. 1589, 2019.
- [17] T. Weerakoon, K. Ishii, and A. A. F. Nassiraei, "An artificial potential field based mobile robot navigation method to prevent from deadlock," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 5, no. 3, pp. 189–203, 2015.
- [18] X. Fan, Y. Guo, H. Liu, B. Wei, and W. Lyu, "Improved artificial potential field method applied for AUV path planning," *Mathematical Problems in Engineering*, vol. 2020, pp. 1–21, 2020.
- [19] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous robots*, vol. 13, pp. 207–222, 2002.
- [20] O. Montiel, R. Sepúlveda, and U. Orozco-Rosas, "Optimal path planning generation for mobile robots using parallel evolutionary artificial potential field," *Journal of Intelligent & Robotic Systems*, vol. 79, pp. 237–257, 2015.
- [21] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, "Mobile robot path planning using membrane evolutionary artificial potential field," *Applied Soft Computing*, vol. 77, pp. 236–251, 2019.
- [22] K. N. McGuire, G. C. de Croon, and K. Tuyls, "A comparative study of bug algorithms for robot navigation," *Robotics and Autonomous Systems*, vol. 121, no. 103261, 2019.
- [23] V. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE Transactions on Automatic Control*, vol. 31, no. 11, pp. 1058–1063, 1986.
- [24] Y. Zhu, T. Zhang, J. Song, and X. Li, "A new bug-type navigation algorithm considering practical implementation issues for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, 2010, pp. 531–536.
- [25] R. Marino, F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria, "A minimalistic quadrotor navigation strategy for indoor multi-floor scenarios," in *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*. Springer, 2016, pp. 1561–1570.
- [26] A. Sankaranarayanan and M. Vidyasagar, "A new path planning algorithm for moving a point object amidst unknown obstacles in a plane," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1990, pp. 1930–1936.
- [27] Y. Horiuchi and H. Noborio, "Evaluation of path length made in sensor-based path-planning with the alternative following," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2001, pp. 1728–1735.
- [28] V. Lumelsky and T. Skewis, "A paradigm for incorporating vision in the robot navigation function," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1988, pp. 734–739.
- [29] V. J. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 5, pp. 1058–1069, 1990.
- [30] I. Kamon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 1996, pp. 429–435.
- [31] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 31–36.
- [32] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot navigation of environments with unknown rough terrain using deep reinforcement learning," in *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2018, pp. 1–7.
- [33] S. Josef and A. Degani, "Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6748–6755, 2020.
- [34] Y. Cheng and W. Zhang, "Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels," *Neurocomputing*, vol. 272, pp. 63–73, 2018.
- [35] X. Xu, Y. Lu, X. Liu, and W. Zhang, "Intelligent collision avoidance algorithms for USVs via deep reinforcement learning under COLREGs," *Ocean Engineering*, vol. 217, no. 107704, 2020.
- [36] J. Choi, C. Dance, J.-E. Kim, S. Hwang, and K.-s. Park, "Risk-conditioned distributional soft actor-critic for risk-sensitive navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 8337–8344.
- [37] D. Kamran, T. Engelgele, M. Busch, J. Fischer, and C. Stiller, "Minimizing safety interference for safe and comfortable automated driving with distributional reinforcement learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1236–1243.
- [38] C. Liu, E.-J. van Kampen, and G. C. de Croon, "Adaptive risk tendency: Nano drone navigation in cluttered environments with distributional reinforcement learning," *arXiv preprint arXiv:2203.14749*, 2022.
- [39] 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.
- [40] 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. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>
- [41] S. Dittert, "Implicit quantile networks (IQN) for distributional reinforcement learning and extensions," <https://github.com/BY571/IQN>, 2020.
- [42] D. J. Acheson, *Elementary Fluid Dynamics*. Acoustical Society of America, 1991.
- [43] T. Lolla, P. F. Lermusiaux, M. P. Ueckermann, and P. J. Haley, "Time-optimal path planning in dynamic flows using level set equations: theory and schemes," *Ocean Dynamics*, vol. 64, pp. 1373–1397, 2014.
- [44] L. Stanislas and M. Dunbabin, "Multimodal Sensor Fusion for Robust Obstacle Detection and Classification in the Maritime RobotX Challenge," *IEEE Journal of Oceanic Engineering*, vol. 44, no. 2, pp. 343–351, 2019.