Navigation in swirling winds: Path optimization for gliders using reinforcement learning

In this study a path planning algorithm for gliders and drones based on machine learning is developed. The algorithm considers disturbance effects from low altitude winds. The movements of gliders and drones are influenced by various physical disturbances in air environments, such as wind, terrain and atmosphere composition. In the present study, the effects of local winds are the primary consideration. A kinematic model is used to incorporate the nonholonomic motion characteristics of a glider, and several reinforcement learning algorithms are compared for path optimization. The proposed approaches determine a near-optimal path that connects the start and goal points with a reasonable computational cost when the map and air current field data are provided. To verify the optimality and validity of the proposed algorithms, a set of simulations were performed in simulated and actual atmospheric conditions, and their results are presented.

I. INTRODUCTION

The motion of aeronautical vehicles is affected by many external factors such as wind speed, wind direction, atmospheric drag, and air composition. Many of these factors are characterized by chaotic behaviour and evolution, which makes path planing a very non-trivial task. For example, a low flying glider or drone, traveling from Varna to Lovech must consider the strong Foehn winds originating from Stara Planina if certain meteorological conditions are satisfied. [1] When transporting medical equipment or goods, drones and gliders must minimize the travel time and reach their target as soon as possible. This optimization problem was first studied by E. Zermelo in 1931. [2] In the cited paper, he studies the problem in the form of a boat navigating on a body of water, originating from a point **A** to a destination point **B**. The boat is capable of a certain maximum speed, and the goal is to derive the best possible control to reach **B** in the least possible time. Zermelo derived a solution to the general case in the form of a partial differential equation, known as Zermelo's equation. It is usually impossible to find an exact solution in most cases, so numerical approaches are required.
Other search based techniques or dynamic programming can be used to find optimal paths, but these suffer from very large computational costs in systems with large state dimension.[3] The computational complexity can be reduced by reducing the system's state dimension, but this often results in an inaccurate representation of the environment and yields infeasable paths. This final project compares several path planning algorithms based on reinforcement learning (RL) [3], which generate a physically realizable path at a reasonable computational cost for a glider navigating in a turbulent atmosphere.

II. METHODS

A. Path planning via reinforcement learning

The application of RL in complex environments is as follows:

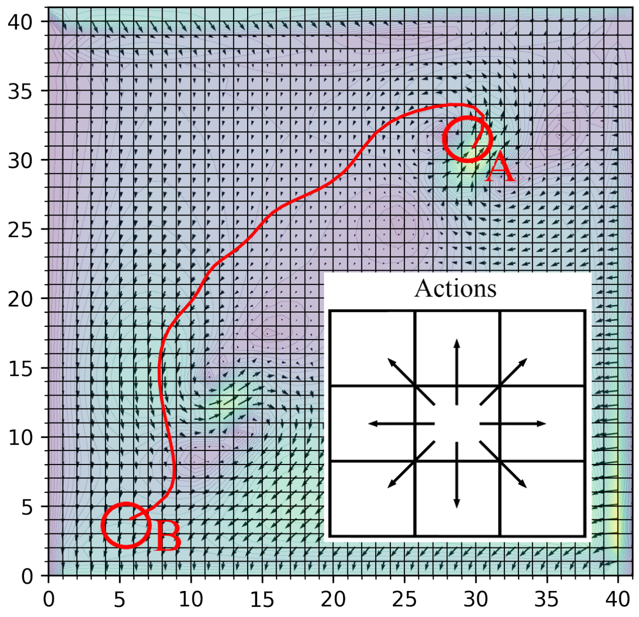

Figure 1: The goal of the agent is to go from a 3 by 3 region **A** to another 3 by 3 region **B** for the least amount of time. The agent navigates in a 41 by 41 gridworld and can take 8 different actions shown in the panel on the lower right corner. The velocity of the fluid is shown as a small black arrow inside each state.

An agent represented by a parametrized policy $\pi(a|s)$ can, through sampling of an unknown environment, find the optimal parameters for the policy that maximize the total received per episode reward:

$$r_{tot} = \sum_{n=1}^{\infty} r_n \quad (1)$$

The agent takes an action from a discrete ensemble of actions.In this study, at each current state the actions are always represented by the 8 different paths the agent can decide to take as shown in Figure 1. The used environments are 2d gridworlds where each state is represented by a location on the grid. A vector of the fluid velocity is associated with each state. This vector is stationary in the case of time independent flow and varies with time for

the time-dependent flow. The fluid velocity at the point that characterizes state s plus the agent's chosen action at that time determine the agent's next state. The characteristics of the environments are as follows: The grid is of size 41 x 41, where each square represents a possible state, and the number of states (Ns) are 1681. Two important parameters are introduced - T_{max} and Δt. T_{max} is the maximum amount of time steps the agent is allowed to explore before terminating the process. This is done to prevent the agent getting stuck in a certain region with no escape due to limitations in his own velocity, to speed up computational time, and to make sure that the sum (1) is always finite. This parameter is introduced because in every studied case, the optimal solutions are between $T_{max}/6$ to about $T_{max}/2.5$. Thus, limiting the agent is not an issue.
The next important parameter is Δt. It is used in the time dependent case of the flow - it sets how many time steps does the environment change for each action of the agent. The environment changes together with the agent moving, but it also changes sufficiently ($\Delta t >> dt$, where dt is the infinitesimal change in time) before the agent has the time to take an action again. This parameter is introduced because realistically no vessel could make a move as fast as the fluid is changing. For the solution simulated using Navier-Stokes we allow the agent to make a move after every 10 time steps of the fluid, meaning $\Delta t = 10$, and for the environment simulated using real data Δt is 20 minutes.
In the simulations we introduce a normalized velocity V_s, which is the median of the velocities of the fluid. The value of 1 for V_s means exactly the median of the fluid velocity and 0.5 means half of it. This will be the agent's own velocity. For the purposes of this research the authors decided to use two RL approaches, which are compared with each other with a goal to find the optimal for navigation strategy in a turbulent flow.

B. RL approach based on an actor-critic algorithm

To identify a time-optimal trajectory a potential based reward shaping is used at each time t during the learning process, for both agents. [4]

$$r = -\Delta t + \frac{|x_B - \mathbf{X}_{t-\Delta t}|}{V_s} - \frac{|x_B - \mathbf{X}_t|}{V_s} \tag{2}$$

Here x_B is the center of the final square the agent wants to reach, V_s is the magnitude of the velocity of the agent, $\mathbf{X}_{t-\Delta t}$ is the position of the agent at the previous step and $\mathbf{X}_t$ is the current position. The first term in Eq. (3) acts as a penalty if the agent takes to long to reach the final target and as a stimulus to find a path that requires the least actions. The second and third terms induce improvement in the distance from the desired state, and stimulate the agent to take paths that would over time increase its proximity to the final state. This kind of reward is known to preserve the optimal policy and help the algorithm to converge faster [4]. The minus in front of the third term means that if the agent got farther from the target he is penalized, and rewarded if he got closer. An episode can be finalized in three different ways: First if the agent moves within 1 square of the desired state the episode is terminated and the agent receives a reward of r=100. Second if the agent hits the borders of the environment, then he does not change his current position and receives a big penalty r=-100. And the third way is as mentioned before if the agent takes too long and takes more than T_{max} actions the episode is terminate and the agent receives the normal reward (3) for his final action.
In order to converge to policies that are robust against small perturbations of the initial condition, which is important in a chaotic environment, each episode is started with a uniformly random position around the initial state, or more precisely there is an equal probability to add a +1,-1 or a 0 to is initial x,y coordinates at the start of each episode.
For this agent, the policy is parametrized by the soft max distribution defined as:

$$\pi(a_j|s_i;\mathbf{q}) = \frac{\exp\ \beta h(s_i, a_j, \mathbf{q})}{\sum_{k=1}^{N_a} \exp\ h(s_i, a_k, \mathbf{q})} \tag{3}$$

Here β is a temperature parameter with an associated increase rate, which is used to lower the exploration at later stages and help the policy converge to a deterministic one. And $h(s_i, a_j, \mathbf{q}) = q_{ij}$ and characterizes the likelihood of taking action a_j at state s_i. During the training phase, the expected total future reward needs to be estimated (1). This agent follows the one-step actor-critic method [3] based on a gradient ascent in the policy parametrization. The critic approach circumvents the need to generate a big number of trial episodes by introducing the estimation of the the state-value function $\hat{v}(s_i, w)$:

$$\hat{v}(s_i, w) = \sum_{i'=1}^{N_s} w_{i'} y_{i'}(s_i) \tag{4}$$

Here $y_i'(s_i) = \delta_{ii'}$. $\hat{v}(s_t)$ is used to estimate the future expected reward $\hat{r}'(t)$, in the gradient ascent algorithm:

$$\hat{r}_{t+\Delta t} = r_{t+\Delta t} + \hat{v}(s_{t+\Delta t}, w) \tag{5}$$

$\hat{r}_{t+\Delta t}$ is used to estimate β_t (6), which is the future expected reward minus the state-value function, used as baseline.

$$\beta_t = [\hat{r}_{t+\Delta t} - \hat{v}(s_t, w_t)] \tag{6}$$

And the update rule for parameterizations of the policy and the state-value functions every time the environment is sampled is as follows:

$$\begin{cases} \mathbf{q}_{t+\Delta t} = \mathbf{q}_t + \alpha_t \beta_t \nabla_{\mathbf{q}} ln(\pi(a_t|s_t, \mathbf{q}_t)) \\ w_{t+\Delta t} = w_t + \alpha_t' \beta_t \nabla_w \hat{v}(s_t, w_t) \end{cases} \tag{7}$$

Where the learning rates α_t, α_t' , follow the Adam algorithm [5] to improve the convergence performance over standard stochastic gradient descent. Both gradients in (15) can be computed manually and are reduced to the following simplified expressions. All state action pairs for the past state are updated. if the state action pair includes the action the agent decided to take the following update is made:

$$q(s_t, a_i) \leftarrow q(s_t, a_i) + \alpha_t \beta_t (1 - \sum_{j=1}^{N_a} \pi(a_j | s_t, q) \qquad (8)$$

In all other cases for the other 7 actions we do the following update:

$$q(s_t, a_i) \leftarrow q(s_t, a_i) - \alpha_t \beta_t \sum_{j=1}^{N_a} \pi(a_j | s_t, q) \qquad (9)$$

In both cases the w values are update in the following way:

$$w(s_t) \leftarrow w(s_t) + \alpha'_t \beta_t \qquad (10)$$

C. Second agent based on Probabilistic Q-learning(PQL)

Based on the work of [6, 7], an agent based on a probabilistic Q-Learning method is introduced. For this method, the same reward as in the previous agent is used, because the same arguments are valid for its applicability. Each episode is also started with a uniformly random position, again because the goal is to get to a solution that does not depend on small perturbations in the initial conditions.

The main characteristic of PQL is that we associate to each state action pair to distinct values. A value for the Q function [3] and a value of the policy, in this case the policy P(just as the Q function) is a 41x41x8 matrix where $P(i, j, m)$ is the probability from 0 to 1, to take action m at state s with coordinates on the 2d grid i, j. The definition is equivalent to the following expression[8]:

$$a_s^\pi = f^\pi(s) = \begin{cases} a_1 \text{ with probability } p^\pi(s, a_1) \\ a_2 \text{ with probability } p^\pi(s, a_2) \\ \vdots \\ a_m \text{ with probability } p^\pi(s, a_m) \end{cases} \qquad (11)$$

As can be seen we take action a_j with probability $p(s, a_j)$. Of course in order for P to have the meaning of probability we need to impose the following requirement:

$$\sum_{A \in A_{(s)}} p(s, a) = 1 \qquad (12)$$

The one-step updating rule of PQL for $Q(s, a)$ is the same as that of QL:

$$Q(s_t, a_t) \leftarrow (1-a_t)Q(s_t, a_t) + a_t(r_{t+1} + \gamma \text{ max }_{a'} Q(s_{t+1}, a')). \qquad (13)$$

Where α_t is the learning which Is a hyper parameter. And$\gamma \in [0, 1)$ is a discount factor. Besides the updating of $Q(s, a)$, the policy is also updated for each action of the agent. After the agent takes an action at s_t , the corresponding probability $p(s_t, a_t)$ is updated according to the reward the agent received - r_{t+1} and the maximum value of $Q(s_{t+1}, a)$ for the state the agent ended up after taking the action $s' = s_{t+1}$

$$P(s_t, a_t) \leftarrow P(s_t, a_t) + k(r_{t+1} + \max_{a'} Q(s_{t+1}, a')) \qquad (14)$$

Where k ($k \geq 0$) is an updating step size.The probability distribution of actions at state $s = s_t[p(s, a_1), p(s, a_2), ..., p(s, a_m)]$ is normalized after each update. One of the main advantages of PQL is its weak dependence on hyper parameters [8]. This is due to the fact that the variation of k in a relatively large range will only slightly affect the learning process because the probability distribution is normalized after each step. The PQL algorithm can be summarized by the following pseudo code:

```
1:  Initialize $Q(s, a)$ arbitralrly
2:  Initialize the policy $\pi : P^\pi = (p^\pi(s, a)_{nxm}$ to be evaluated
3:  repeat (for each episode):
4:      Initialize $t = 1, s_t$,
5:      repeat for each step of episode
6:          $a_t \leftarrow$ action $a_i$ with probability $p(s_t, a_i)$ for $s_t$
7:          Take action $a_t$, observe reward $r_{t+1}$, and next state $s_{t+1}$
8:          $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a_t \delta^Q_{t+1}$
9:          Where $\delta^Q_{t+1} = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$
10:         $P(s_t, a_t) \leftarrow P(s_t, a_t) + k(r_{t+1} + \max_{a'} Q(s_{t+1}, a'))$
11:         Normalize $P(s_t, a_i)|_{i=1,2,...,m}$
12:     until $s_{t+1}$ is terminal
13: until the learning process ends
```

Another reason why PQL is a good approach in a chaotic environment is that there is no need to introduce a superficial hyper parameter that would regulate the exploration at different times (β in the actor critic approach). In such a way if a change occurs in the environment at later times the algorithm would naturally start to once again explore by updating the policy, and its exploration won't be reduced by external mechanisms. On the other hand as will be shown in Section III when a quasi-optimal solution is found, the policy rapidly converges to an almost deterministic one.

D. Simulation of a viscous fluid using the Navier-Stokes equation

The Zermelo problem in this study is applied in two environments. The first one is a viscous fluid simulated by solving the Navier-Stokes equation:

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \vec{v} + f \qquad (15)$$

Where $\nu = \frac{\mu}{\rho_0}$ is called the kinematic viscosity. v is the velocity of the fluid p is the pressure of the fluid. ρ is its density and f is a source of the the fluid. In our case we solve the equation with varying time dependent boundary conditions which are specified in the software repository of [9] The two sources added also vary in time, but have e specific location. The equation is solved using finite difference method, the main theoretical reasoning behind such a solution can be found in [10], where the stability and error are discussed in detail. The parameters required to solve the equation dx, dy, dt (the discretization of both time and space) are taken from [9]. A part of the code provided in the cited paper is reused for our needs.

III. TESTS AND COMPARISONS

This section is broken in to three main subsections. Comparison with the trivial policy, comparison with the optimal solution where an applicable numerical method for its derivation exists, and comparisons of the two agent with each other where the other two methods do not yield solutions.

A. Trivial policy

The trivial policy is a policy where the agent always takes the action that points in the direction of the desired target location. Because of the discretization this direction is rarely available to the agent and that's why he takes the action that is closest to his desired direction. The trivial policy outperforms our agents in certain very simple environments as shown in Figure 2 due to the exploring nature of the agents, but performs extremely poorly in dynamic situations precisely due to the lack of exploration.

B. Comparison with analytical solutions

Due to limitations in our computational power we were not able to apply zermelo's solution. The problem arose when calibrating the initial conditions of the differential equations one is required to solve . The authors of [11] explore in debt the reason behind the instability and unreliability of zermelo's solution when one is required to identify an initial steering angle. That is why we performed only a single comparison with an analytical solution provided by the following differential equation [12]:

$$y'(x) = \frac{c_0 v_v}{\sqrt{1 - c_0^2(v_v^2 - v_f^2)}} \tag{16}$$

Here c_0 is a constant which determines the boundary condition, v_v is the velocity of the agent, v_f is the fluid velocity. Results from this comparison are shown in Figure 2.

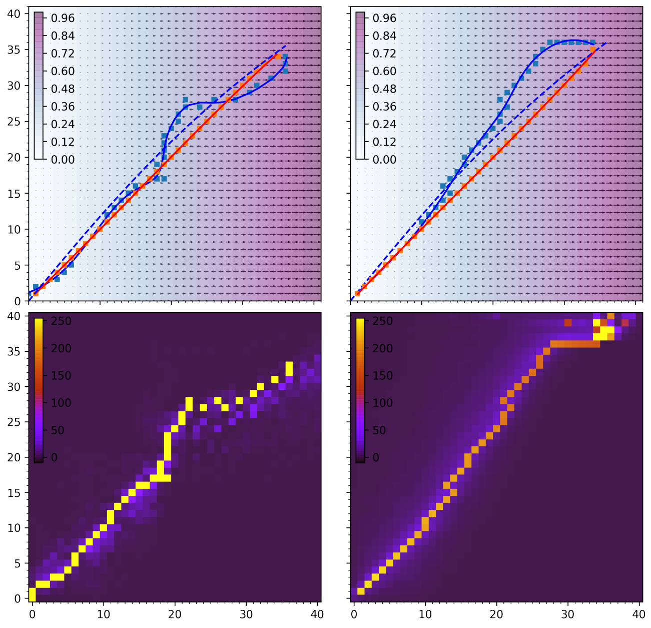

Figure 2: Right column contains the trajectory taken by the actor-critic agent, blue dashed line is the analytical solution. Left column contains the trajectory of the PQL agent. The bottom graphs represent the density of visited states along the training process. The red line corresponds to the path taken by the trivial policy agent.

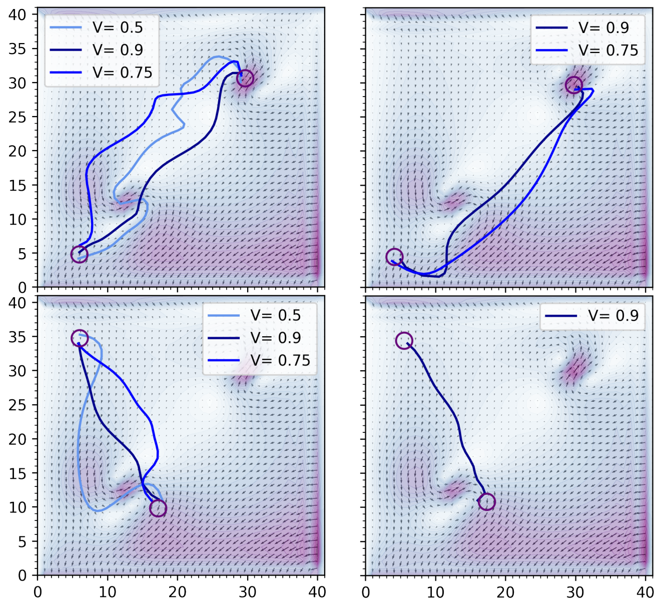

Figure 3: On the left are shown the trajectories of the PQL agent with 3 different speeds. On the right is the actor-critic agent.

C. Behaviour in the simulated fluid

The above results show the quasi-optimal paths that both agents were able to converge to for different speeds

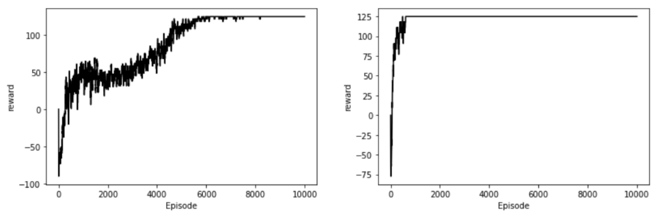

Figure 4: Right graph is the learning curve of the PQL agent, the left is the learning curve of the actor-critic agent.

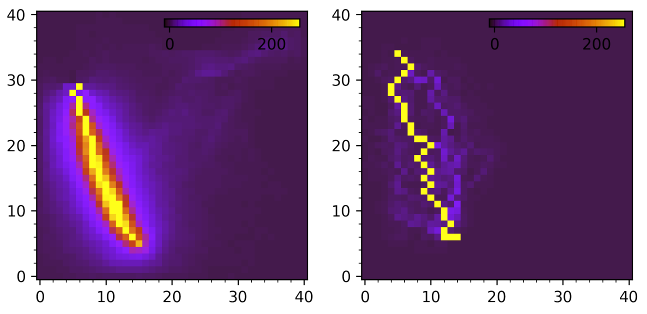

Figure 5: Density of visited states in the time dependant version of the simulated flow. On the left are the trajectories of the PQL agent and on the right are the actor-critic agent trajectories.

V_s. The missing paths on Figure 3 are due to the agent not being able to consistently reach the desired location. From the data we can conclude that the PQL agent strongly outperforms the actor critic one, first by consistently finding a solution to the problem with all speeds it was tested for and secondly due to its learning rate. From Figure 3 which shows the evolution of the reward for both agents in the case of $V_s = 0.9$ in the upper start and final configuration, one can see how the PQL agent ramps up the reward much faster and once the quasi optimal path is reached consistently follows it. Tests on the time dependent version of this environment show the same perfomance for both agents, and the actor-critic model yields slightly better results.

PQL was applied to path planning in a real-world environment using actual wind speed data to examine the effectiveness and practical utility of the approach. Actual wind data were taken from 35 meteo-stations across Bulgaria. Data were acquired from a free online weather data provider. [13]. The data downloaded consists of 12 consequent measurements of the wind speed and direction for each station. Then the data is interpolated in time to a total of 124 values for each component of the velocity vector of the wind. To make this data compatible with the RL framework of this work, the geographical coordinates of the weather stations were linearly transformed to coordinates on a 41 by 41 grid. This transformation ignores the curvature of Earth, but due to the small size of the region, and its position on the globe, this is an acceptable approximation. Thus, the final generated data

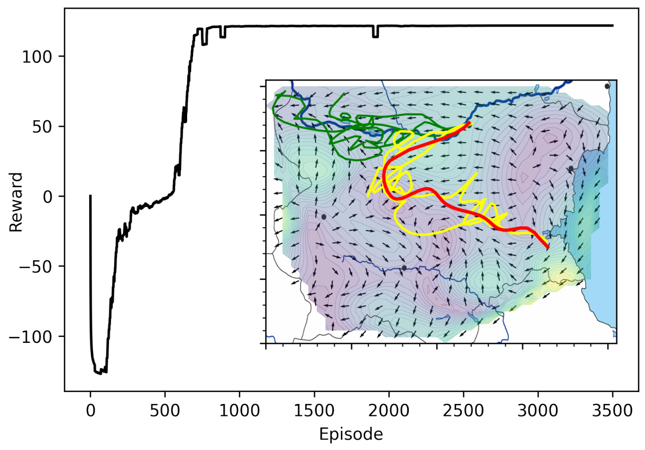

Figure 6: This figure shows different trajectories taken by the QPL agent through its learning process in the time independent case. The starting point of the agent is Rousse, and the end point is Burgas. The green lines are trajectories of the agent during episodes 50-100, the yellow lines are episodes 400-600, and the red line is the final path taken at episode 2500.

consists of 124 different 41 by 41 grids representing time steps, where each state is associated with a vector representing the wind.
To test the performance of the agent in this environment, only one snapshot in time was first used. Results are shown in Figure 6. In the first 200 episodes the agent is going out of the boundaries, because of the wind and is punished. By episode 400 the agent has learned not to go out of the boundaries and steer towards the goal point. At episode 1000 the agent has learned the trajectory which yealds the biggest reward.

For the time dependent case of this environment an animation of the flow was made and can be found in the supplementary material. A comparison between the Trivial policy (green line) and the PQL agent (red line) for $V_s = 0.5$ is made. On this clip, the trivial policy fails to get to the goal point, and the PQL agent finds a pseudo-optimal path.

IV. CONCLUSIONS

In conclusion our project was able to reproduce the main results in [11] and expand on them with the addition of a second agent. The PQL agent outperformed the actor-critic algorithm in certain scenarios most notably in complicated time independent environments, and in the time dependent case for the environment simulated with real data. But the actor critic agent did better in the more simple environments and in the time dependent Navier-Stokes equation for at least one configuration, that's why we can't claim with certainty that one agent is clearly better than the other one. But nonetheless we showed that further work must be done in this field to find an optimally suited RL approach to tackle Zermelo's problem, because it is clear that the methods presented in [11] can be outperformed.

We were also able to apply the agent on a "real" environment and show its applicability in real world problems which the authors of the original paper did not attempt to do. As it was mentioned in the beginning the solving of Zermelo's problem has significant implications especially in optimal control theory. Techniques developed to solve this problem may be applicable not only to gliders but also to optimizing air traffic for airplanes [14] and sea navigation for long and short distance marine routes [15].

Further work on this project must ensure that more results are compared with existing numerical solutions and more complicated RL algorithms are applied to the problem. The authors expect to see Deep RL being applied in this sphere in the future because to the best of the authors knowledge no such algorithms have been applied to the problem.

V. ACKNOWLEDGMENTS

We would like to express gratitude to Assistant Professor Galin Guylchev for providing materials regarding the numerical solution of Navier-Stokes equation and advice on improving our current model.

VI. SUPPLEMENTARY MATERIAL

The following link contains the supplementary material to this project.

https://github.com/tongenspace/Navigation-in-swirling-winds-Path-optimization-for-gliders-using-reinforcement-learning/find/main

[1] Векилска, *ОБЩА КЛИМАТОЛОГИЯ* (УИ " Св. Климент Охридски ", София, 2012).

[2] E. Zermelo, ZAMM - Journal of Applied Mathematics and Mechanics **11**, 114 (1931).

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, MA, 2017).

[4] A. Y. Ng, D. Harada, and S. J. Russell, in *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999) p. 278–287.

[5] D. P. Kingma and J. Ba, arXiv e-prints , arXiv:1412.6980 (2014), arXiv:1412.6980 [cs.LG] .

[6] D. Dong, C. Chen, H. Li, and T. Tarn, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **38**, 1207 (2008).

[7] C. Chen, D. Dong, and Z. Chen, International Journal of Quantum Information **4**, 1071 (2006).

[8] C. Chen, D. Dong, H.-X. Li, J. Chu, and T.-J. Tarn, Neural Networks and Learning Systems, IEEE Transactions on **25**, 920 (2014).

[9] L. Barba and G. Forsyth, Journal of Open Source Education **1**, 21 (2018).

[10] J. Anderson, "Mathematical properties of the fluid dynamic equations," in *Computational Fluid Dynamics*, edited by J. F. Wendt (Springer Berlin Heidelberg, Berlin, Heidelberg, 2009) pp. 77–86.

[11] L. Biferale, F. Bonaccorso, M. Buzzicotti, P. Clark Di Leoni, and K. Gustavsson, Chaos: An Interdisciplinary Journal of Nonlinear Science **29**, 103138 (2019), https://doi.org/10.1063/1.5120370 .

[12] B. Liebchen and H. Löwen, arXiv e-prints , arXiv:1901.08382 (2019), arXiv:1901.08382 [cond-mat.soft] .

[13] METEOBLUE, "Weather history download," (2021).

[14] D. González Arribas, M. Soler, and M. Sanjurjo Rivo, Journal of Guidance, Control, and Dynamics **41** (2017), 10.2514/1.G002928.

[15] J.-B. Caillau, S. Maslovskaya, T. Mensch, T. Moulinier, and J.-B. Pomet, in *CDC 2019 - 58th IEEE Conference on Decision and Control* (Nice, France, 2019).