Diffusion Models for Control of Underactuated Agents Hitching Ocean Currents

Lukas Ketzer, Bear Häon, Mustafa Suman

***Abstract*—In dynamic flow fields like ocean currents, an agent can navigate by going with the flow, using minimal propulsion to guide itself into advantageous flows. This hitchhiking flows navigation approach is highly energy-efficient. However, reliable navigation remains a challenge in this context, as typically only forecasts are available, which can significantly differ from the actual currents. The forecast error can exceed what the agent's actuation can handle. This project examines the potential of diffusion-model based controlers that allows reliable navigation of underactuated agents hitchhiking ocean currents.**

I. Introduction

Reducing the atmospheric carbon dioxide concentration is a critical challenge of the twenty-first century. Carbon sequestration is critical to mitigating the effects of climate change. As part of a natural process, seaweed growing in coastal regions sequesters carbon into the deep ocean. While growing, this seaweed fixates dissolved atmospheric CO2 into biomass. Part of the seaweed's biomass reaches the deep ocean, where the carbon is confined for millennia. However, this natural process already exists and is fairly limited due to the finite coastline. Robotic seaweed farms could autonomously grow and deposit seaweed in the open ocean. These farms should maximize seaweed growth conditions and solar exposure during the growth cycle. Afterward, they deposit the biomass at specific locations in the ocean, optimizing the confinement of the biomass. These robotic seaweed farms rely on station keeping in favorable areas and navigation to more promising growth areas or deposit locations.

The overactuated navigation approaches used in ships require significant power to overcome the drag forces inherent in navigating through fluids. The power required scales with $P = F_{Drag} \cdot \frac{\delta_s}{\delta_t} \propto A_C \cdot v^3$, where A_C is the cross sectional area and v the velocity relative to the surrounding fluid. This makes overactuated control prohibitively expensive for energy constraint applications such as long duration floating structure with large cross-sectional area. An energy-efficient steering paradigm leveraging the ocean currents is needed: **navigating agents by hitchhiking complex flows**. By using the drift of these nonlinear, time-varying flows for propulsion, only a minimal amount of energy is required to nudge the agents into beneficial flows.

A. Agent and Flow Dynamics

We consider an agent operating in a general time-varying, non-linear flow field $v(x, s) \to R^n$, where $x \in R^2$ represents the state and s the time.

Report submitted Dec 13, 2023

The agent's actuation signal, denoted by u, comes from a bounded set $U \in R^{n_u}$ where n_u is the dimensionality of the control. The dynamics of the system, given certain initial conditions, are governed by an Ordinary Differential Equation (ODE) as:

$$\dot{\xi}(s) = f(\xi(s), u(s), s) = v(\xi(s), s) + g(\xi(s), u(s), s), \quad s \in [0, T]$$

where ξ symbolizes the trajectory and $\xi(s) \in R^n$ the state at time s. The agent's movement in the flow is influenced by its control $g(x(s), u(s), s)$ and the drift of the surrounding flow $v(x(s), s)$.

B. Problem Setting

The objective is for the agent to reliably navigate from an initial state x_0 to a target region $T \in R^n$, being underactuated such that:

$$\max_u \|g(x(s), u(s), s)\|_2 \ll \|v(x(s), s)\|_2$$

most of the time. Throughout its operation, the agent receives a flow forecast $\hat{v}(x, s)$, which deviates from the true flow $v(x, s)$ due to a stochastic forecast error $\delta(x, s; \omega)$ where ω is a random variable. The agent obtains a new forecast at regular intervals.

The goal is to enable underactuated agents to reliably navigate in complex, natural flows. Reliability is measured as a probabilistic bound, meaning the agent arrives at the target with a high likelihood. Thus, we measure the controllers success rate empirically navigating from a start to a target T over a mission set M in realistic flows. Success is defined by the agent reaching T within a maximum allowed time T_{max}; otherwise, it's deemed a failure. In realistic settings, only deterministic flow forecasts are available to the agent.

C. Previous solutions and novel approaches

1) Dynamic programming [1]: The first method is based on the Model Predictive Control (MPC) paradigm, where regular replanning with deterministic dynamics is used to compensate for imperfect knowledge of the dynamics and achieve reliable navigation. The key insight is that the value function of the backwards Hamilton-Jacobi (HJ) reachability can be used for closed-loop control. The optimal control that minimizes the distance of the agent to the target set at a terminal time can be extracted for every state and time in the domain.

The initial solution follows a dynamic programming algorithm, Multi-Time Reachability, as it yields a controller that that reaches the target in the minimum possible time, if feasible, or approaches as close to the target as possible, if

not. For this, the cost function J to achieve this behavior is defined:

$$J(x, u(\cdot), t) = d(\xi_{u,t,x}(T), T) - \int_t^T I_T(\xi_{u,t,x}(s))\, ds \quad (1)$$

where $d(x, T)$ is a distance metric from a point x to the target set T, $\xi_{u,t,x}(\cdot)(s)$ is the position of the agent at time s when starting at x at time t. With $I_T(x)$ being the boolean indicator whether the agent is at the target T. This cost function implies that, if reachable, the agent should reach the destination as quickly as possible. Otherwise, the optimal control should try to minimize the terminal distance to the target.

Given this cost function, one derives the HJ Partial Differential Equation (PDE) whose solution is given by:

$$\frac{\partial J^*(x,t)}{\partial t} = \begin{cases} (1 - \min_u[\nabla_x J^* \cdot f(x,u,t)]) & \text{if } (x,t) \notin T \\ J^*(x,T) = d(x,T) & \text{otherwise} \end{cases}$$

From the value of J^*, which contains information about the minimum time required to reach the destination, we can extract an informative time-to-reach map D^* as follows:

$$D^*(x,t) = T + J^*(x,t) - t, \forall (x,t) \text{ such that } J^*(x,t) \leq 0$$

If the target is reachable from x starting at t (implied by $J^*(x,t) \leq 0$), then $D^*(x,t)$ represents the minimum duration needed to reach T. Inside the target, $D^*(x,t) = 0$ for all t, $x \in T$. The optimal control $u^*(x,t)$ minimizes the Hamiltonian:

$$\begin{aligned} u^*(x,t) &= \arg\min_{u \in U} f(x,u,t) \cdot \nabla_x J^*(x,t) \\ &= \arg\min_{u \in U} g(x,u,t) \cdot \nabla_x J^*(x,t) \end{aligned}$$

where the second equation follows from the first because $v(x,t)$ is not a function of u. Note that the control obtained from this value function provides the time-optimal control at all states, making it more useful for reliable navigation.

Algorithm 1 Multi-Time HJ Closed-loop Schema

Input: Forecast Flow(s) $\hat{v}(x(s), s)$, $t = 0$, $x_t = x_0$
while $t \leq T_{\max}$ and $x_t \notin T$ **do**
  if new forecast available **then**
    compute time-to-reach map D^*
    Use latest D^* for control
    $u_t^* = \arg\min_{u \in U} g(x_t, u, t) \cdot \nabla_x D^*$
    $x_{t+1} = x_t + \int_t^{t+1} f(u, x(s), s)\, ds$
  end if
end while

The authors assess the method's effectiveness in realistic ocean currents over an extensive set of multi-day start-to-target missions across the Gulf of Mexico and temporally over four months. Using HYCOM forecasts and simulating true currents with HYCOM hindcasts, this method achieves a 99% success rate. Furthermore, the method is evaluated in a more challenging setting, using forecast errors that are more representative of real-world operations: planning with HYCOM forecasts and simulating the true currents with Copernicus hindcasts. In this setting, the method achieves a 82.3% success rate. The considerable difference between the 99% and 82.3% success rates in the two simulation settings emphasizes the importance of accurate forecasts for reliability.

2) Online Deep Reinforcement Learning [2]: As seen in the previous section, inaccurate, error-prone forecasts of ocean currents pose a significant challenge for the controller. Dynamic programming-based path planning on coarse and inaccurate forecasts still builds a solid baseline for this challenge. However, it is limited by the accuracy of the forecasts, fails to take advantage of local measurements and cannot learn from experience. This section introduces a deep reinforcement learning approach that builds on the results of the previous section. The goal is to:

- learn from local measurements
- learn patterns from failed and suboptimal experience
- learn risk-averse behavior

The agents observed state consists of the time-to-reach-map and an extrapoloation of the local current dynamics from a gaussian process. As a learning algorithm, a double-Q-learning algorithm using fully-connected layers. The action space is descretisized to move along 8 radiant angels along the plane with a fixed propulsion. To stabilize training experience replay is used. Interestingly, a dense reward function is chosen that also leverages the time-to-reach-map is chosen.

However, the deep reinforcement learning agent is outperforming the Multi-Time Hamilton Jacobi agent by +0.6 p.p. success rate on the test missions.

D. Data-Driven Offline Reinforcement Learning

1) Reinforcement learning via Sequence Modeling [3]: The transformer architecture is an emerging paradigm that has shown great potential in sequence prediction tasks of complex data structures, such as natural language. The success of this method in context dependent sequence prediction motivate the reformulation of reinforcement learning problems as sequence prediction task. Every episode in a reinforcement learning task may be reformulated as a sequence of state, action, and reward. This leads to the trajectory representation

$$(\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, ...)$$

where, for $i \in N$, s_i are the states, a_i the agent's actions and $\hat{R}_i = \sum_{t'=i}^{T} r_t$ is the sum of rewards-to-go until the episode ends at T. In essence, the transformer architectures convert the RL-based optimal path planning problem into a supervised learning problem via sequence modeling. Following an offline learning paradigm, and agents trajectory can be recorded in this data structure. The transformer would then be trained to predict the next token given a context length of K steps. Most interestingly, this approach has shown great performance in imitation learning tasks in its original implementation of typical ATARI games.

2) Transformers for Trajectory Planning [4]: However, building upon this idea of reinforcement learning as a sequence prediction task, recent research has picked up this idea to solve the optimal path planning problem in ocean currents. In their

toy examples the authors use the above approach to guide an agent through simplified flow fields.

II. Method

A. Simulator

Inspired by the environment in [1], we developed a custom simulator adhering to the gym API framework which represents a 20 x 20 field with a single gyre mocking a simple ocean current in the center of that respective field. In that environment it is possible to generate custom start points, target points, agent magnitudes and mission times. Further, it is possible to set the distance to consider a mission as succesful. Depending on the pre-selected setting, for the agent it is possible to act continuously in that environment (i.e. indicating the action as a fraction of a magnitude in x and y direction) or in a discrete way, (i.e. choosing 1 of equidistant 8 angles with a fixed given magnitude).

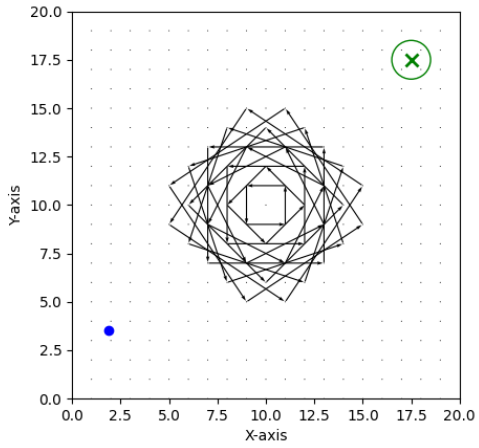

Fig. 1. Custom simulator modeling a Single Gyre. The blue dot represents the start of an example mission start where as the green cross represents a goal in a mission. The circle indicates the area the agent has to reach to consider a mission as successful (in our experiments we chose a threshold of 1).

B. Simple Diffuser

We employ a diffusion-based generative process for trajectory synthesis, leveraging a U-Net based denoising model and a discrete-time Denoising Diffusion Probabilistic Model (DDPM) scheduler. We initialize a U-Net 1D model with specified input, output channels, and other hyperparameters tailored to our simulator-generated dataset - which is normalized and loaded using custom Dataset classes for training and sampling.

During training (Algorithm 2), trajectories are first corrupted with Gaussian noise whose variance is managed by the DDPM scheduler. The denoising model then learns to reverse this process, iteratively refining noisy inputs towards clean data. A sample of genereated output is seen in Figure 2. In the sampling phase (Algorithm 3), we generate trajectories by reversing the diffusion process, starting from noise and progressively denoising it.

Algorithm 2 Training the Diffusion Model

Input: Training dataset, Denoising model, Noise scheduler
Initialize Parameters: Set batch size, horizon, dimensions, learning rates
Load training dataset
Initialize denoising model and noise scheduler
Initialize optimizer
for each epoch **do**
  for each batch in dataset **do**
    Generate noisy trajectories
    Predict and denoise using the model
    Compute loss and backpropagate
    Update model parameters
  end for
end for

Algorithm 3 Sampling from the Diffusion Model

Input: Trained denoising model, Noise scheduler
Initialize Sampling Configuration: Set batch size, horizon, noise levels
Load the trained denoising model
Initialize noise scheduler
Generate initial random noise x
for each timestep in reverse **do**
  Apply the denoising model to x
  Optionally add Gaussian noise
end for

C. Value-Selection Diffuser

Here, we combine a denoising model and a value network to guide the diffusion process in generating trajectories with higher estimated values (Algorithm 4). The key components of this process are the **Denoising Model (UNet):** This component is responsible for predicting the noise in the data at each timestep, the **Noise Scheduler:** It controls the variance of the noise during the diffusion process, and the **Value Network (RNN):** This network estimates the values

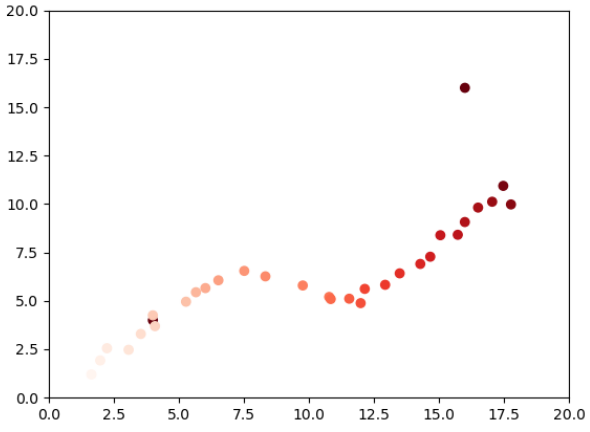

Fig. 2. Sample: Diffusion Trajectory generation

of the trajectories.

The mathematical addition to part B is as follows:

Value Estimation:

$$V(x) = \text{RNN}(x)$$

where $V(x)$ is the value of the trajectory x, estimated by the RNN.

Algorithm 4 Denoising and Value-Guided Diffusion Process

Input: Denoising model, value model, noise scheduler
Initialize Parameters: Set batch size, horizon, dimensions, learning rates
Initialize denoising model and value model
Initialize noise scheduler
Generate initial random noise x
while not end of diffusion process **do**
  for each timestep i **do**
    Predict residual noise using the denoising model
    Denoise x using the predicted noise and scheduler's step function
    Optionally add Gaussian noise based on scheduler's variance
    if step i is a multiple of ϵ **then**
      Apply the value model
    end if
    Select the trajectory with the highest value
  end for
end while

D. CQL Baseline

Conservative Q-Learning (CQL) [5] was introduced within the scope of Offline-RL to improve training and performance. Conventional off-policy RL techniques often struggle Offline-RL scenarios owing to the distribution mismatch between the dataset and the policy being learned. CQL suggests the adoption of a conservative Q-function that aims to provide a lower estimate of the actual value. Empirically, CQL has recently demonstrated strong performance, making it a highly suitable baseline for comparison. In our case, we make use of CQL applied to Double-Q learning.

We use a discretized action space with 8 symmetrically arranged angles to facilitate the training process of the CQL algorithm.

III. Experiments

Method	Avg. Cum. Reward	Avg. Final Dist.
Diffusion	-405.023	5.742
Guided Diffusion	-402.310	4.941
CQL	-404.774	5.764

TABLE I
Experimental results comparison

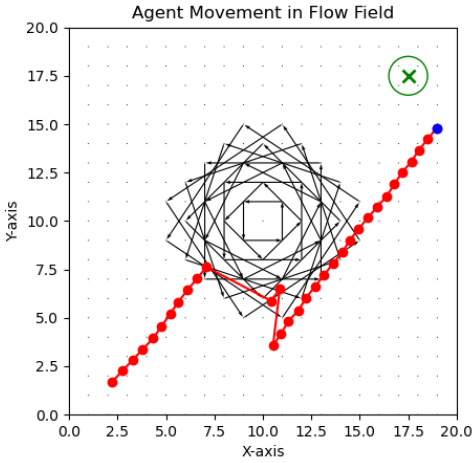

Fig. 3. Diffusion

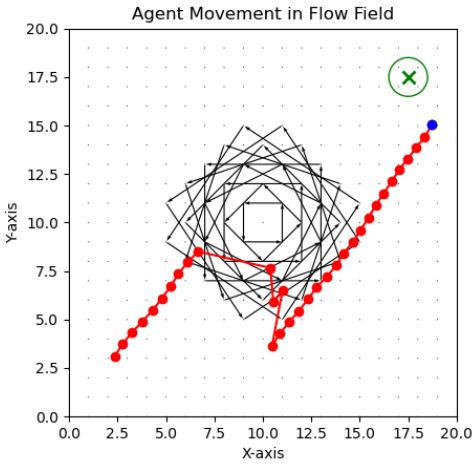

Fig. 4. Value Guided Diffusion

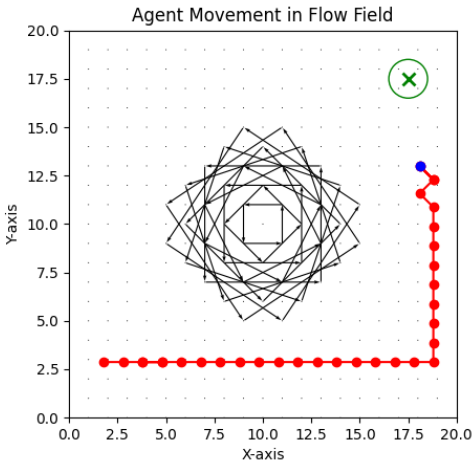

Fig. 5. CQL Baseline

A. Data Generation

The dynamics of the gyre current in our simulator are consistent throughout each rollout. Data collection was executed using an agent operating under a completely randomized policy, ensuring diverse sampling of the state space. A rollout ended at a maximum of $T = 100$ timesteps, or when the agent moved outside the field. Finally paths of with at least 40 transitions were kept.

B. Methods

In our experiments, we evaluated the performance of a diffusion-based trajectory generation model, focusing on navigational tasks within a simulated flow field environment. The core of our experimental setup involved agents with different strategies: the Diffusor Agent, the Diffusor Agent with Value Guidance, and a Conservative Q-Learning (CQL) Agent. Regarding the CQL Agent, our hyperparameter-tuning (i.e. grid search) led to a cql_alpha value of 0.4, a hidden_size of 64, num_layers of 2, 20.000 training steps, a batch size of 128, a cql_temperature value of 1, an greedy epsilon value of 0.04 and a discount of 0.98. The above mentioned agents were tasked with navigating in a Single Gyre Flow Field. Each agent's performance was quantified through a series of 100 missions, each starting from randomly generated coordinates within defined bounds, targeting a fixed destination. Key metrics for evaluation included cumulative reward and ultimate distance from the target at mission completion. Each mission was limited to a maximum of 32 steps - where each agent could take a continuous action. Visualized results are seen in Figure 3, Figure 4 and Figure 5.

IV. Conclusion

In this study, we have successfully implemented and evaluated a diffusion-based controller for a path planning problem in a simulated ocean current environment. Our custom simulator, designed to mimic a single gyre, provided a robust platform for testing various control strategies. The data-driven approach, particularly the use of a denoising diffusion probabilistic model, demonstrated its efficacy in generating viable trajectories under uncertain and dynamic conditions. The experimental results demonstrate that the diffusion-based model can generate realistic trajectories and learn from the trajectories in the offline replay buffer. Nonetheless, further work is required to test the performance of the diffusor in more realistic settings, as for example in fully simulated ocean current environments.

Next Steps and Further Work

Simulator Include realistic ocean currents based on actual hindcasts, similar to work in [1].
Architecture Optimize diffusor architecture and to further improve denoising process and increasing kernel width.

V. Individual Contributions

- **Lukas Ketzer**: Conceptualization, screening of methodologies, development of toy problem simulator, implementation of diffusion model based path planner
- **Bear Häon**: Conceptualization, screening of methodologies, implementation of value-guided diffusion algorithm
- **Mustafa Suman**: Conceptualization, development of toy problem simulator, implementation of CQL-Algorithm (including fine-tuning), screening of methodologies

REFERENCES

[1] M. Wiggert, M. Doshi, P. F. J. Lermusiaux, and C. J. Tomlin, "Navigating underactuated agents by hitchhiking forecast flows," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 2417–2424.

[2] J. Jeannin and M. Wiggert, "Reinforcement learning for under-actuated current-based navigation," Master's thesis, ETH Zurich, 2022, unpublished master's thesis.

[3] L. Chen *et al.*, "Decision transformer: Reinforcement learning via sequence modeling," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 15 084–15 097.

[4] R. Chowdhury and R. Murugan, "Intelligent onboard routing in stochastic dynamic environments using transformers," in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2023.

[5]