Imitation learning for an ASV path planner in complex marine environments:

A feasibility study

B. Rutteman

Delft University of Technology

TU Delft

Imitation learning for an ASV path planner in complex marine environments:

A feasibility study

by

B. Rutteman

TU Delft supervisor (Chair):	L. Ferranti
Daily supervisor (Roboat):	J. Klein Schiphorst
Project Duration:	september, 2024 - june, 2025
Faculty:	Faculty of Mechanical Engineering, Delft

Cover: Photograph of the founders of Roboat and their ASV watertaxi product ”Lucy”

TU Delft

Contents

1

Acknowledgments

For attaining the product of my thesis I would like to thank particularly both L. Ferranti and J. Klein Schiphorst for their supportive efforts. I would like to thank L. Ferranti specifically for providing guidance with respect to her academic/scientific expertise and professionalism to achieve the final product of this thesis. Additionally, I would like to thank J. Klein Schiphorst specifically for providing guidance in the process of constructing the final solution of this research, discussing interpretations of its corresponding results and for giving me the chance to work on the interesting topic that is ASV navigation by means of imitation learning.

B. Rutteman
Delft, June 2025

2

Introduction

This research article report is the second of two parts that comprises the thesis work performed by B. Rutteman in collaboration with Roboat. In this research article report, a research article that concerns a study towards the application of imitation learning (IL) algorithms Action Chunking Transformer (ACT) and diffusion policy as an autonomous surface vessel (ASV) path planning algorithm in complex marine environments has been documented. In collaboration with Roboat, the decision for conducting this particular research was made due to the growing interest in the application of imitation learning algorithms for autonomous vehicle (AV) navigation in popular literature. ACT and diffusion policy were selected to be investigated for the concerning application, out of the most prominent algorithm types used in the state-of-the-art literature regarding imitation learning and robotic path planning. This selection process has been described in the first part of the thesis work, which was the literature study. In this second part of the thesis work, a research was performed in which both model types were evaluated for the concerning application by leveraging two simulators: a simple 2D-grid simulator and a more complex GAZEEBO simulator of Roboat. Results from performed evaluative experiments led to the conclusion that the ACT model in combination with the trained on data sources is not adequate to achieve significant performance for the complex task that is realistic ASV navigation. Diffusion policy showed promise for this specific application, however could not be evaluated in desirable detail due to encountered hardware limitations for training and evaluating the model its performance.

3

Research paper

Imitation Learning for an ASV Path Planner in Complex Marine Environments

A Feasibility Study

Abstract — This thesis proposes a study towards the application of imitation learning (IL) algorithms Action Chunking Transformer (ACT) and diffusion policy as an autonomous surface vessel (ASV) path planner in complex marine environments. Rationale for conducting this research are the ubiquitous limitations regarding fine tuning cost- and or reward functions of conventional state of the art algorithms for ASV navigation in complex environments. In this study we trained both algorithm types on data sources collected from a basic 2D-grid simulator and a more realistic GAZEBO simulator. Subsequently we evaluated both algorithm's performances in each respective simulator in terms of the success rate for a standard navigation task. We found relatively high success rates for the 2D-grid simulator (0.98 for ACT and 0.53 for diffusion policy). For the GAZEBO simulator, we found poor performance for ACT (0.0 success rate) and for diffusion policy we could not establish the performance due to hardware limitations. For future work, the capabilities of both models could be investigated further by trying to bridge the gap between the simple 2D simulator and the GAZEBO simulator. Mainly the effect of task complexity and the quality and quantity of data used for training the models on the performances of the models can be investigated in these future work studies.

1. Introduction

Regarding Autonomous Surface Vessels (ASV's), ensuring a safe navigation of the vessel is paramount for real life deployment. The associated economical cost of collisions is a significant reason for this. Furthermore, especially for deployment in proximity of humans or when the ASV is designed for transport of human lives, navigation components are vital to be designed adequately so that no human can be physically harmed by the vessel.

State of the art solutions for (safe) ASV navigation differ wildly and concern both end-to-end or modular/hierarchical approaches [1]. Grid-based search methods such as Dijkstra's algorithm or the A* algorithm are commonly used in hierarchical approaches. End-to-end methods often apply artificial intelligence or neural networks to learn the entire navigation task [2] which can entail including motor control as well [3]. Additionally, model predictive control (MPC) based methods [4] and reinforcement learning (RL) based methods are applied regularly when designing an ASV navigation component [1].

However the more conventional methods of these solutions offer limited performance as ASV path planners in complex environments due to the requirement of carefully designing associated cost-functions (A*, Dijkstra's, MPC) or reward functions (RL) in order to define appropriate navigation behavior. Carefully designing these functions in order to define appropriate behavior for ASV navigation in complex marine environments can be very challenging due to the ubiquity of loose marine traffic regulations and the consequentially wide range of possible obstacle configurations in an ASV's environment. However, these strenuous optimization efforts might be avoided altogether by making use of a technique called imitation learning (IL) for solving the process of ASV navigation.

IL is an AI-based technique that allows a model to learn the behavior corresponding to a specific task by being trained on datasets containing proprioceptive data and corresponding responsive data by the agent to these observations collected during execution of the specific task by an expert. More technically phrased, the model tries to learn a direct mapping from each possible state to that state's optimal action for the concerning task based on the trained on dataset. Hence, IL allows for direct interpretation of desirable (navigation) behavior inherent in a task related dataset on which it is trained and thus does not require any manual programming of behavior or the development of a suitable reward- and/or cost function for defining this behavior [5]. Furthermore, IL might offer the ability of capturing appropriate behavior in situations of highly complex state- and action spaces, [6] which is a characteristic of the workspace of an ASV. This characteristic is even more pronounced when the ASV is deployed in complex marine environments.

The last few years there has been a surge in researches and applications that aim to apply imitation learning in order to capture the desirable behavior for AV trajectory planning and/or vehicle control [6]. In 2023 and 2024, two novel state of the art types of imitation learning models, respectively Action Chunking Transformers (ACT) and diffusion policy, were developed and have been received by the robotics community with great enthusiasm. Both

these models have been tested and evaluated for varying applications and consistently with very good results in terms of performance [7], [8], [9], [10]. This includes researches that cover applications in which vehicle navigation is of importance, such as automatic highway driving by means of diffusion policy [11] and learning multiple complex tasks and navigation between these tasks to a mobile robot by means of ACT [12].

Because of these aspects, in collaboration with ASV developing company Roboat, we propose to try and build on these imitation learning developments. In this work, we investigate the feasibility of applying ACT and diffusion policy models for ASV path planning in complex marine environments. In order to assess this feasibility, we leveraged two simulators. The first of these concerned a basic 2D-grid simulator designed from scratch containing a 2D-LQR controller for governing the navigation. Additionally, a more complex premade ASV GAZEBO simulator of Roboat was used which contained more realistic components, such as realistic ASV dynamics, ASV sensor plugins and a representation of the Marineterrein in Amsterdam. We expanded both simulators to allow for extracting datasets for the models to be trained on and for performing evaluative experiments. Due to leveraging two simulators of different complexities, the limitations of the models in terms of capturing the task of ASV navigation could be established in more detail. By researching the application of ACT and diffusion policy as an ASV path planner algorithm in complex marine environments, a promising research gap in the field of robotic path planning will be addressed. To the knowledge of the authors of this paper, no research has yet been published that has tested and evaluated the performance of both ACT and diffusion policy models for this specific application.

2. Related Work

In order to contextualize the reason for researching the research gap stated in the introduction, we investigated related work in popular research literature. This entails research towards imitation learning techniques, among which ACT and diffusion policy are considered, as well as the application of imitation learning as AV/ASV path planners.

2.1. Imitation Learning

According to a research overview concerning state of the art imitation learning techniques from 2023 from Zare et al. [6], the main imitation learning subcategories are: behavioral cloning (BC), inverse reinforcement learning (IRL) and adversarial imitation learning (AIL).

Behavioral Cloning

BC algorithms aim to learn directly from expert observations by identifying the correct actions for a specific state and basing their learned behavior on this. BC has the strength of being able to copy behavior directly and hence requires no knowledge of environment dynamics to train the algorithm. Furthermore, BC is a relatively computationally efficient algorithm to train and deploy [5]. The main drawback of using BC, generally lies in the phenomenon of covariate shift. Covariate shift means the model produces inadequate outputs regarding its state, which can lie out of the trained on distribution, and allows the model to create a chain of inadequate responses. By doing so the model "drifts off" from the trained on state distribution, making it even harder for the model to generate an adequate output. Covariate shift is the consequence of training the model on merely a limited distribution of possible states [5].

In order to overcome the problem of covariate shift, so-called interactive imitation learning (IIL) was developed. Interactive imitation learning (also called Dataset Aggregation or "DAgger" based algorithms) provides the most intuitive alternative for mitigating the covariate shift problem associated to traditional imitation learning. The discerning characteristic of interactive imitation learning is the availability of an active expert during the training phase to guide the process in the correct direction. This means that a human will give active feedback during policy iteration in order to collect new data in which the correct action is associated with a specific state. This data is then leveraged to retrain the policy at every iteration of this process. Because of this, covariate shift is mitigated and generalization capabilities of the model can be increased [5].

Inverse Reinforcement Learning

A different kind of imitation learning algorithm that does not require active expert feedback to prevent covariate shift, is called inverse reinforcement learning (IRL). This algorithm type is basically a backwards form of standard RL. IRL aims to learn a reward function based on expert demonstrations that inherently represent a specific task or behavior, and thus also the desirable policy. The learned reward function is then leveraged to determine the concerning policy function by evaluating and improving iteratively until the solution converges towards the optimal policy [13].

Adversarial Imitation Learning

Adversarial imitation learning (AIL) is another imitation learning technique that has shown great potential in solving trajectory planning tasks over the last few years. It borrows from the general concept of generative adversarial networks (GAN) in the sense that it also features an adversarial game between a discriminator and a generator in order to train the model. In a GAN adversarial game, both the discriminator and generator aim to win a competition in which the generator tries to generate images which the discriminator can not distinguish from real life images. Both components train themselves on successfully and failed attempts at these generated images. This process is iterated until the model has converged and can reproduce very real images that the discriminator simply can not distinguish from artificial ones. In AIL algorithms, this concept is used to train a model to generate real life-like expert behavior (state-action outputs) based on demonstrations. This means the model's generator will iteratively learn to come up with actions for any given state that are indistinguishable from expert produced ones by the discriminator.

2.2. Imitation learning for AV planners

Various IL alorithms have been successfully applied for AV planning purposes. IIL has been successfully applied with regard to the application of autonomous driving for

navigating structured environments by Yokoyoma et al. by applying an improved online sampling method that is able to distinguish moments of interventions from the rest of the dataset on which the model is trained [14]. In other researches such as the ones by Ahn et al, IIL has also been successfully applied for AV navigation in unstructured environments which showcases the generalization capabilties of IIL [15]. Ahn et al. achieves this by training its IIL algorithm on vision based occupancy grids maps, generated by making use of camera data collected during expert demonstrations of the task. Subsequently, the IIL algorithm separates drivable and non-drivable space based on these vision inputs in order to navigate towards its goal.

IRL algorithms have also been successfully applied for the purpose of AV navigation. In several researches the technique is applied for navigation in dynamic environments, such as in the research from Phan-Minh et al. [16], the researches from Huang et al. [17] [18], and the research from Gonzalez et al. [19]. In the research from Phan-Minh et al., the algorithm first constructs a set of trajectory proposals, subsequently removes the ones that are considered unsafe and as a last step evaluates the remaining trajectories and applies the highest scoring one for navigation. Huang et al. applies a similar hierarchical approach of generating a trajectory, however after the initially proposed trajectories are generated, Huang et al. predicts future paths of all dynamic objects in the environment and takes these into account for selecting the most suitable option out of the found proposals. Gonzalez et al. performs a lattice based search method in the state space of the ego vehicle for computing its trajectory. This allows the algorithm to only consider feasible robot motions, which allows faster inference and optimization of the selected path regarding dynamic objects in the environment due to incorporating the element of time in the optimization.

In terms of AIL, numerous algorithms have been developed for the purpose of AV navigation. In the research from Couto et al., a hierarchical GAIL-approach is applied [20]. During the first step of this hierarchy, a vanilla GAN-type network is used in order to generate a bird's eye view (BEV) image from raw image data collected during the training loop. Subsequently, these BEV-images are fed to the GAIL-network in order to learn the task of navigation in urban environments. In a research from Wang et al. AIL is applied for ASV navigation by combining the concepts of both IRL and AIL in the form of an adversarial inverse reinforcement learning (AIRL) algorithm [21]. This particular algorithm learns both the appropriate policy and the cost function of the task corresponding to the dataset it is trained on. The performance of the algorithm is further improved by assigning semantic rewards to the GAIL its generator reward function during the training process.

Diffusion policy has also been applied for AV navigation/planning. It has been used for mobile robotics [22] and even dense autonomous driving by leveraging a training procedure that applies reward gradient guided denoising [11]. This algorithm is called Diffusion-ES and its training procedure optimizes non-differentiable and black-box objectives.

2.3. Imitation learning for ASV planners

Some researches have also shown success in applying the technique of IL for ASV navigation. A 2023 research from Chaysri et al. implemented a GAIL-based algorithm in order to test the performance of ASV navigation from point A to point B with added environmental disturbances such as wind and current [23]. Associated experiments yielded good performance with a 100% success rate for all evaluated wind and current velocities in the research. In another research by Higaki et al, the authors tested the performance of a CNN-GAIL based network when applied as an ASV navigation component for traversing environments with dynamic obstacles. The model was trained on COLREG adhering data in order to test the ability of the model to learn the corresponding apropriate behavior. The model showed good performance with a 92-97% agreement rate in terms of action output between expert behavior and model behavior. However, the trained model showed limitation in handling situations with traffic vessel approaching angles close to the boundary angle that divides different action modalities. This being either a pass to the right or to the left with respect to the traffic vessel. At these approach angles, the model consistently chose a rightwards pass, even though this violated COLREG traffic conventions in certain situations [24].

3. Background

3.1. ACT

The original paper describing the concept of ACT is the 2023 paper from Zhao et al. [7]. ACT was originally designed to enable low-cost and imprecise hardware to perform complex tasks with a robotic manipulator. It is an imitation learning algorithm which characteristic property is to predict actions not one step at a time but in action chunks of multiple consequential actions. In order to allow for smooth trajectories, these chunks are computed at a high frequency. Since the model is queued at a static frequency even during chunk execution, overlapping actions of separate chunks are averaged. This averaging is designated as ensembling. In order to train the model appropriately, action chunks are fed to a transformer type architecture, which characteristic property is to allow for interpretation of the temporal coherency in the data on which it is trained.

The model is trained in a form that resembles a conditional variational auto encoder (CVAE). The model consists of a transformer encoder to function as the CVAE encoder which determines the latent variable z during training time. Besides the action sequence and the joint positions, this "CVAE encoder" is also fed with a "CLS-token". This token is used to allow for prediction of the mean and variance of the z-variable, which is used as an input to the CVAE decoder component later on. From a mathematical perspective, this encoder computes the parameters of the variational distribution $q_\theta(z \mid w, x, y)$ (formula 3.1).

$$[q_\theta(z \mid w, x, y) = \mathcal{N}(z; \mu_\theta(w, x, y), \mathrm{diag}(\sigma^2_\theta(w, x, y))) \tag{3.1}$$

In this expression, w represents the CLS-token, x represents the joint positions, y represents the corresponding action sequence, z represents the latent variable and $\mu_\theta(w, x, y)$ and $\sigma^2_\theta(w, x, y)$ are the mean and variance of the normal Gaus-

sian distribution $\mathcal{N}$ respectively. The encoder's parameters are designated by θ. The encoder subsequently generates the mean $\mu_\theta \mu_\theta$ and standard deviation $\sigma_\theta \sigma_\theta$ of the Gaussian distribution and uses these to compute the z-value by means of the following expression (formula 3.2).

$$z = \mu_\theta(w, x, y) + \sigma_\theta(w, x, y) \odot \epsilon \tag{3.2}$$

This expression showcases how the model samples a value from the corresponding distribution in a way that allows backpropagation due to its differentiability. In this formula ϵ represents a random sample drawn from a normal distribution with a mean of 0 and a standard deviation of 1. After determining the z-latent, the z-latent, the camera observations and the robot states are then fed to a structure that resembles the CVAE decoder. This decoder then learns the parameters of the variational distribution in formula 3.3.

$$p_\phi(y \mid z, x, v) = \mathcal{N}(x; \mu_\phi(z, x, v), \sigma^2_\phi(z, x, v)) \tag{3.3}$$

In this expression v designates the camera images captured during demonstrations. By learning these parameters the model is able to generate outputs corresponding to the action sequence that one fed to the CVAE encoder.

Training the model can be achieved by minimizing the following loss function (formula 3.4) through the process of backpropagation.

$$\mathcal{L}_{\text{CVAE}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}} \tag{3.4}$$

In which the reconstruction loss is defined as in formula 3.5.

$$\mathcal{L}_{\text{recon}} = - \sum_{s_t, a_{t:t+k} \in D} \log \pi_\theta(a_{t:t+k} \mid s_t) \tag{3.5}$$

In this reconstruction loss, k designates the chunk size of the action sequence and $\pi_\theta(a_{t:t+k} \mid s_t)$ designates the result of the current iteration's policy probability function when putting in the action chunk $a_{t:t+k}$ and corresponding observation sources here designated as states s_t. For all state action chunk sequence pairs in the demonstration dataset, these log values are summed, which generates the reconstruction loss. To this reconstruction loss a regularization term is added in the form of KL-divergence loss (formula 3.6) which is common practice for VAE loss functions.

$$\mathcal{L}_{\text{KL}} = KL(q(z|x)||p(z)) = -\frac{1}{2} \sum \left(1 + \log \sigma^2 - \mu^2 - \sigma^2\right) \tag{3.6}$$

In this formula μ^2 designates the square of the mean of the distribution of $q(z|x)$. The corresponding value basically shows how far the distribution is shifted from the origin. σ^2 designates the variance of the distribution and hence shows the spread or the uncertainty of the distribution.

Adding KL-divergence to the overall loss function ensures a well structured, continuous and smooth latent space. It is added to increase generalization capabilities of the model [25]. The architecture of the ACT algorithm has been visualized schematically in Figure 1.

3.2. Diffusion policy

In order to explain the concept of diffusion policy, it is helpful to first explain what general diffusion networks are, since diffusion policy is a subtype of these types of networks.

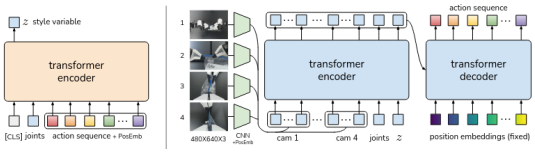


Figure 1: ACT architecture schematics [7]

Denoising Diffusion Probabilistic Models (DDPM's)

Diffusion networks (Denoising Diffusion Probabilistic Models or DDPM's) are algorithms that learn by gradually introducing noise to the data it is fed to and that try estimate the amount of noise added at each of the corresponding steps. This process is repeated until the model's parameters are tuned towards convergence. During inference, the model is then able to denoise an artificially created distribution of Gaussian noise back into an output residing in the data distribution on which it was originally trained. [26]

In order to be able to guide the model towards convergence during training, forward diffusion has to be performed. During this process, k amount of noise additions ϵ_k is added to a clean data sample. The amount of noise added at each step is determined by a predefined noise schedule. The model will then learn to predict the noise ϵ_k added to the sample at each step. During training, at each step k , the actual noise ϵ_k and the predicted noise $\epsilon_\theta(x_0 + \epsilon_k, k)$ are compared in a loss function. The discrepancy between the actual added noise ϵ_k and the one predicted by the current model $\epsilon_\theta(x_0 + \epsilon_k, k)$ guides the model towards convergence. In terms of loss function, the original DDPM paper of Ho et al, uses a mean squared error loss function of the form in formula 3.7 [27].

$$L = \text{MSE}(\epsilon_k, \epsilon_\theta(x_0 + \epsilon_k, k)) \tag{3.7}$$

Mathematically speaking, during inference the network takes a sample x from a Gaussian noise distribution and performs a selected amount of k denoising steps $x^k, x^{k-1}, \ldots, x^0$, to which for each iteration a certain amount of noise ϵ_k is removed until the sample is converged to the correct noise free output x^0. The actual denoising process, also called reverse diffusion, can be mathematically expressed as follows [27].

$$x^{k-1} = \alpha \left(x^k - \gamma \epsilon_\theta(x^k, k) + N(0, \sigma^2 I)\right) \tag{3.8}$$

In this expression ϵ_θ is the noise prediction network with parameters θ that will be optimized through learning, and $N(0, \sigma^2 I)$ is Gaussian noise added at each iteration. This expression can be regarded as mathematically analogous to the standard expression of a single noisy gradient descent step (formula 3.9) [27].

$$x' = x - \gamma \nabla E(x) \tag{3.9}$$

In terms of this analogy, ϵ_θ has a similar function as the parameter gradient field ∇E and the terms of α and σ are functionally similar to γ in that they define the learning rate schedule of the training procedure.

The main strengths of diffusion networks lie in the characteristic of being able to train a model that is able of

creating high-quality and diverse outputs [28]. The main limitation of this technique is that it requires a relatively long time to create an output [29]. This may limit its use for applications that require a relatively fast inference.

Diffusion policy theory

As stated in the previous paragraph, diffusion policy is a specific diffusion network algorithm and is designed to learn policies for a variety of robotic applications. During the training phase it will try to denoise a sequence of actions A_t conditioned on the current observations O_t. In order to achieve this it makes use of the gradient field of the action distribution score for approximating the corresponding conditional distribution $P(A_t \mid O_t)$. The observations that are used for each action sequence prediction are fed to separate encoders each. By using these observations, the model will generate action outputs by taking into account T_o observation steps. The model will then generate an action sequence of length T_p which represents the action prediction horizon, of which only the first T_a actions of this are to be executed, which represents the action execution horizon. This way, the model is supposed to allow for temporally consistent actions, a smooth follow up of successive actions and receding horizon control of the model. The corresponding architecture of the diffusion policy model has been visualized schematically in Figure 2.

Two main architectural adaptations have to be made to the standard DDPM architecture to create a diffusion policy model. Firstly, the output x must now designate agent actions and not images anymore and secondly the denoising process must be conditioned on the observations O_t corresponding to the actions. Hence, mathematically speaking the denoising process of equation 3.8 is changed to the following expression (formula 3.10) in which the denoising process is conditioned on the observations.

$$A_{k-1}^t = \alpha \left(A_k^t - \gamma \varepsilon_\theta \left(O_t, A_k^t, k \right) + N\left(0, \sigma^2 I\right) \right) \quad (3.10)$$

These adaptations result in the following adapted loss function (formula 3.11).

$$L = \text{MSE}\left(\varepsilon_k, \varepsilon_\theta \left(O_t, A_0^t + \varepsilon_k, k \right)\right) \quad (3.11)$$

It should be noted diffusion policy is notorious for having a limited generalization capability. However, multiple algorithms have successfully addressed and mitigated these issues by adding additional rich data sources [9], [22] or by querying an active expert during training [30].

4. Problem formulation

Consider the case in which an ASV has to travel from point A to point B by means of either an ACT or diffusion policy path planner (Figure 3). Additionally, for achieving this goal the ASV is tasked with a safe and time and energy efficient way of traversing the environment. On a global level, this safety of environment traversal can be expressed by means of the success rate, which expresses the rate at which the ASV was able to find its path from A to B regardless of time and energy efficiency. The time and energy efficiency of the traversed path can be determined by looking at the time it took and the amount of traversed meters the ASV required in order to achieve its goal regarding some pre-defined start- and goal position and comparing this with the optimal solution. This being following the shortest possible path with optimal velocity.

Now consider that in our concerning case its corresponding range of possible marine environment scenarios, multiple static and/or dynamic obstacles can be present. In these scenarios, the number of static and dynamic obstacles can vary and each dynamic traffic vessel can be encountered from a multitude of directions, with a variety of different properties such as object size, velocity, etc. These properties make this environment relatively complex compared to other AV environments such as driving on the road, where road cues such as traffic lights, -lanes and -signs limit the number of possible traffic scenario's. Additionally, the vessel encounter behavior in these scenarios can be considered more complex than driving on the road as well. In terms of movement organization in marine environments, one-on-one vessel encounter behavior guidelines have been constructed in the convention on the internal regulations of preventing collissions at sea (COLREG), which allows for somewhat predictable behavior. However, for multiple vessel encounters these guidelines have not been defined and general expert behavior is more loosely based in these types of situations [31]. Hence, because of this traffic vessel encounter behavioral complexity in certain situations, and the high variety of possible environment configurations, strictly defining the appropriate behavior for ASV navigation in complex environments can be challenging. It is because of these aspects, the property of IL-algorithms, ACT and diffusion policy in particular, of capturing behavior inherently present in task-related datasets is supposed to offer a good solution for this type of AV navigation.

As we stated in section 2, ACT has been successfully applied for robotic manipulator planning. However, as to the knowledge of the authors of this research, no research has been performed in which the algorithm was applied for pure AV navigation. Though, the model has been applied for training tasks to a manipulator with a mobile base called mobile ALOHA. Mobile ALOHA can be seen as a somewhat in-between application of a manipulator and an AV, as ACT is also applied in order to allow the manipulator to navigate between tasks in the same room [12]. Hence, this research suggests the notion that the algorithm might be applied with adequate performance for pure navigation purposes as well. Some characteristic strengths of ACT emphasize this notion even further, such as its ability to have a good generalizability [32], an accurate long horizon prediction [33], having a low inference time [32] and having a relatively stable training procedure [33]. Additionally, the action chunking mechanism of ACT allows a correspondingly trained algorithm to be agile in terms of evading environmental objects, since the complete trajectory is not computed at once [7]. This allows the ego vehicle to quickly adapt its trajectory as it goes. Hence by considering these strengths, investigating the performance of ACT for an ASV navigation algorithm proves to be a promising research gap.

As we stated in section 2, diffusion policy has been applied successfully for not only robotic manipulator path

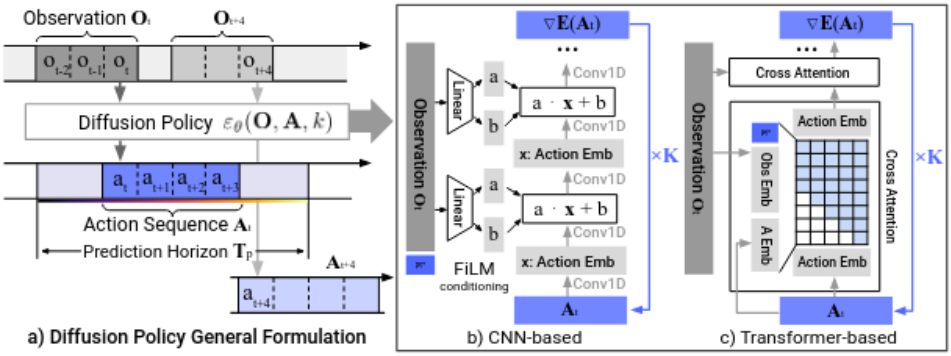


Figure 2: Diffusion policy architecture schematics [27]

Figure 3: Scenario of an ASV attempting to cross Amsterdam's IJ channel. Ubiquity of multiple traffic vessels on the channel make this a complex marine environment.

planning, but also AV navigation [11]. The corresponding performance results emphasize the potential of this technique for closely resembling tasks, such as ASV navigation. Furthermore, the characteristic properties of diffusion policy such as having a stable training procedure [34], being able to accurately capture long range dependencies [27] and the ability to capture different action multi-modalities [27] support this notion. Due to the stable training procedure, the model is likely to converge to a functioning and optimal solution after training. Due to its ability to capture long range dependency, the model might be able to recognize long temporal trends in the fed data sources such as anticipation of the planned trajectory with respect to obstacles in the environment. Furthermore, due to characteristic of being able to capture the action multi-modality, the model might also be able to learn underlying COLREG-conventions inherent in the dataset. It should be noted, diffusion policy is also known to have some weaknesses in terms of navigation as we stated in section 2, which are the generalizability and the inference time of the algorithm. However, Yang et al accomplished inference times of 5.85 seconds for a non-optimized diffusion policy model and even a 0.50 second inference time when the model is optimized for this property [11]. This shows the potential of increasing inference time of these types of models. Furthermore, the inference time for ASV navigation is generally not required to be as fast as for highway driving such as in the research from Yang et al, due to traffic vessels typically traveling less close to one another and the typically less high vessel accelerations and velocities due to higher traffic vessel inertia. In terms of generalizability, the studies from Ze et al. [9], [22] have shown this property can be increased to an extent where even multiple different tasks can be recognized by the model, by feeding more rich data-sources to the model. In the case of this research, LIDAR-data was added in order to feed the model with 3D point cloud clusters of objects in the environment. Altogether, by looking at these strengths and weaknesses, investigating the performance of diffusion policy applied as an ASV navigation algorithm proves to be a promising research gap.

Because of the promises of applying either ACT or diffusion policy as an ASV path planner, we chose to investigate these two models for this particular application in this research. In conclusion, the main goal of the thesis project is to answer the following research question: "To what extent can an ASV navigate in complex marine environments by means of an ACT or diffusion policy algorithm?" For answering this question, the metric of success rate in terms of traveling from point A to point B will be the main guideline.

5. Framework

In order to be able to give a valid answer to the posed research question in section 4, we leveraged two simulator environments for this research.

5.1. Simulator environments

For this research, we designed a basic 2D-grid simulator from scratch. Additionally we used a more complex GAZEBO simulator of Roboat. Due to leveraging two simulators of different complexities, the limitations of the models in terms of capturing the task of ASV navigation could be established in more detail.

2D-grid simulator

Firstly, we designed a very basic 2D-grid ASV simulator. By establishing performance of the models in terms of navigation in the context of a basic simulator, the extent to which the models can capture very basic navigation accurately could be investigated. In Figure 4 the lay-out of this simulator its environment and the corresponding bird's eye view (BEV) image can be seen.

GAZEBO simulator

Additionally, we used the GAZEBO simulator of Roboat. This simulator contains realistic ASV dynamics, a path plan-

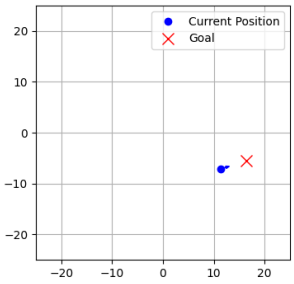


Figure 4: BEV-image of the 2D-grid simulator setup

ner for navigation towards a goal position, ASV sensor plug-ins and an environment resembling the Marineterrein in Amsterdam, all contributing to a simulator environment closely resembling a potential real life ASV scenario (figure 8 and figure 9). The differences between the 2D-simulator and the GAZEBO simulator in terms of complexity have been summarized in Table 1.

Simulator	Environment	Dynamics	Planner	Sensors
2D-grid simulator	2D-grid (20x20)	Basic dynamics	A* (+add. heuristics)	BEV-image
GAZEBO simulator	3D 'Marineterrein' representation	Realistic ASV dynamics	2D-LQR	Three cameras (front, left-back, right-back) BEV-image LIDAR-data

Table 1: Overview of both simulators in terms of complexity

5.2. Framework design pipeline

In order to establish the performance for ASV navigation in complex marine environments for the two models for the two simulator variants, we undertook a number of methodological steps. First of all, we had to collect adequate datasets from the simulators on which a path planning algorithm of ACT and diffusion policy can be trained. These datasets had to represent the behavior of ASV navigation in complex marine environments inherently. Subsequently, we had to adapt the original model architectures of ACT and diffusion policy (as they were designed by Zhao et al. and Chi. et al. respectively) so that they were able to take in the appropriate data sources of these ASV navigation datasets and so that they generate the output of an ASV navigational path. As a last step, we had to incorporate an inference component of the models into the two simulator variants so that the performance of these models as ASV navigation components could be evaluated in a series of experiments. In the sections below, each of the three described methodological steps (Figure 5) has been described in more detail.

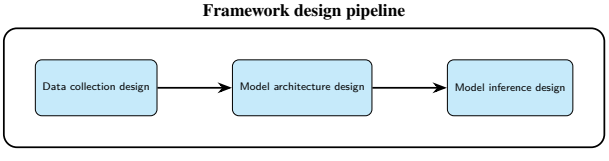


Figure 5: The framework design pipeline showing the sequential steps from data collection to model inference.

Data collection design

In both simulation environments, we ran automatic demonstration trials in which the ASV travels from a starting location to a goal location during which informative ASV training data was collected. For gathering this data from the simulator, we assumed complete and direct knowledge of the environment by the ASV. This entails that every property thinkable regarding environmental objects could be directly extracted from the simulator without having to capture and process sensor data, which would conventionally be necessary for collecting this data.

For performing the automatic demonstration conduction in the 2D-grid simulator, firstly we sampled the starting position of the ASV and a goal location randomly across the 20x20 grid. This ensured each region across the grid would be represented with a relatively large density in terms of starting- and goal locations for a complete training dataset (Figure 6). Whenever these positions where determined

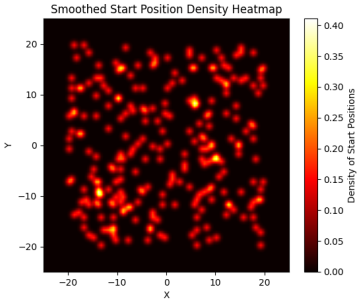


Figure 6: Distribution of starting position over 250 demonstrations

the ASV would make its way towards the goal position. This was achieved by leveraging a 2D-dynamics model for computing the next state at each iteration (equation 3.12 and equation 3.13) and by leveraging a 2D-LQR-controller (equation 3.15) for computing the control input given the current iteration's state error (equation 3.20). The values used for the A- and, B matrices for defining the dynamics of the state space system and the values used for the Q- and R matrices for tuning the behaviour of the control inputs can be found in 3.14 and 3.16 respectively.

$$x_{k+1} = Ax_k + Bu_k \tag{3.12}$$

$$x_k = \begin{bmatrix} x_k \\ y_k f \end{bmatrix}, \quad u_k = \begin{bmatrix} v_{x,k} \\ v_{y,k} \end{bmatrix} \tag{3.13}$$

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \tag{3.14}$$

$$J = \sum_{k=0}^{\infty} \left(x_k^\top Q x_k + u_k^\top R u_k \right) \tag{3.15}$$

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \tag{3.16}$$

In terms of tuning the state and input matrices A and B and the cost matrices Q and R, we attempted to design a system that represents the most basic form of navigation with respect to dynamics and control inputs.

We set the A-marix to be an identity matrix in order

to enforce very basic system dynamics in which the ASV does not change its state unless acted on by control input. Setting the values to 1 ensures growth or decay of the ASV's state parameters is not encouraged when input is absent. By setting the diagonal of the B-matrix as the identity matrix multiplied with Δt, the transition to the next state is merely governed by the computed velocity input for the ASV at each iteration step.

We set the Q-matrix to have the same values across the matrix's diagonal so that the controller would not penalize presence of the ASV at any specific location more or less across the 2D-grid. Furthermore, we set the values in the matrix to the arguable value of 1 so that the model would still allow for relatively tight reaction to the state-space error, but would not allow the ASV to be too jerky or overshoot easily. Additionally, we set the values of the R-matrix at an arguable value as well. This was due to adding an additional operation of clipping the computed control values so that the velocity in neither direction (x,y) could be higher than 2.2. By designing the controller this way, it was intended to gain more control over the control inputs and thus the displayed navigation behaviour. By doing so the ability of the models to capture the aspect of a maximum velocity in its learned dynamics could be interpreted later on during evaluation of the experiments as well. Additionally, setting the maximum velocity relatively low ensured changes in the ASV's state between iterations stayed relatively small, giving the models larger datasets and more data to interpret the corresponding behaviour for any demonstration trial. The sole aspect of the R-matrix that was tuned with a strong rationale in mind was that the control input costs of x- and y- were set to equal values. This ensured no input (x- or y related) was favored over the other and thus that the most basic form of navigation behaviour is employed. The behaviour displayed by this navigational system in the 2D-grid simulator over 250 demonstration trials has been displayed by means of a streamline plot in Figure 7.

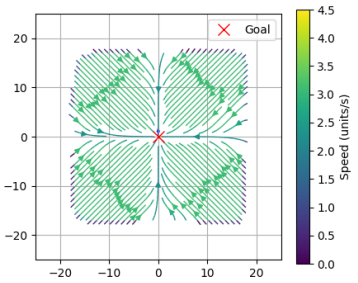


Figure 7: 2D-simulator dynamics visualized as streamline plot

We computed control input values by solving the Riccatti equation (equation 3.17 and equation 3.18) to establish the control gain and multiply this value with the state error (equation 3.19 and equation 3.20) for each iteration in this simulator. One iteration in the simulation represented a Δt of 0.9 s. Whenever the ASV would reach the goal location, the goal location and the ASV would be spawned at a new random location again and the next demonstration trial would start.

$$P_{\text{new}} = A^\top PA - A^\top PB(R + B^\top PB)^{-1}B^\top PA + Q \quad (3.17)$$

$$K = \left(R + B^\top PB\right)^{-1} B^\top PA \quad (3.18)$$

$$u = -K \cdot \text{state error} \quad (3.19)$$

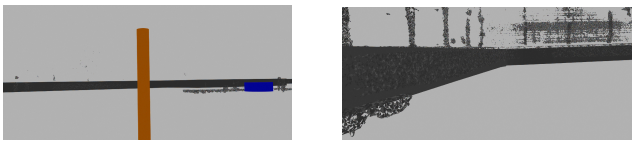
$$\text{state error} = x_{\text{current}} - x_{\text{goal}} \quad (3.20)$$

For performing the automatic demonstration conduction in the GAZEBO simulator, we wrote an algorithm from scratch in which the ASV would be assigned a goal position just beside an orange pole spawned at a random location in the environment. Whenever the ASV would reach this goal position near the pole, a new goal position and corresponding orange pole would be placed. Additionally, besides changing the position of the goal position and orange pole, reaching the goal position also respawned the location of three static obstacles available in the environment. These were then respawned at randomly assigned locations in the environment. These static obstacles were present in the environment in the form of three blue disks which base started on the level of the water surface in the simulator.

During the automatic demonstration trials, we logged data sources from both simulators into hdf5-files, which is a common file format used for training AI-models. We created these hdf5-dataset loggers from scratch for this project. As stated before in section 5.2, we assumed complete and direct knowledge of the ASV environment for this project. Data sources that we logged during demonstrations were: goal-position with respect to the ego vehicle, ASV orientation, ASV velocity, the pose data of all environmental objects relative to the ego vehicle (GAZEBO sim. only), the relative pose of the ASV with respect to the pose of the previous logged pose, raw bird's eye view (BEV) image data of the scene and raw camera data of a camera sensor at the front, the right back and the left back of the ASV (GAZEBO sim. only). The relative pose of the ASV was the action the models were intended to learn. An impression of the logged camera images and BEV-image from the GAZEBO simulator can be seen in Figures 8 and 9.

For the 2D-simulator, we extracted the data sources straight from the simulator. However, for the GAZEBO simulator this was not possible due to the fact that the camera image data and BEV-image data were not directly obtainable from the simulator environment. Hence, we achieved data logging for the GAZEBO simulator by making use of a function called ApproximateTimeSynchronizer from the message filters library with a slop value of 0.1, which allowed logged data of the same timestep to be within a 0.1 second range. The data sources were transmitted at a frequency of 2 Hz by means of robot operating system (ROS) topics and the data was captured by the logger at a frequency of 2 Hz. By setting the slop parameter to a value of 0.1 in combination with transmitting and capturing data every 0.5 seconds (2 Hz), we ensured all data that was transmitted corresponding to each 0.5 second temporal bin was stored together in the datasets. Additionally, choosing this sampling frequency ensured collected hdf5-files were still manageable in terms of size for both storing on laptop SSD as well as storing them on the high performance

computing (HPC) server and loading the data to the models before training and performing the actual training steps. Due to RAM-limitations and GPU limitations of the leveraged HPC-server, we set this sample frequency at this relatively low value to balance an accurate representation of the task being performed with minimizing the dataset size.



(a) front camera image **(b)** right back camera image

Figure 8: Two camera images captured from the GAZEBO simulator

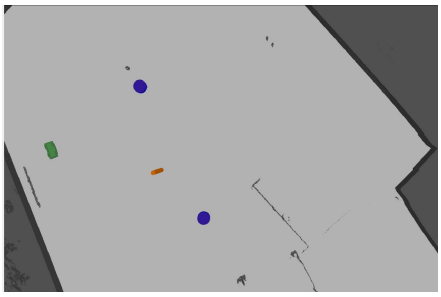


Figure 9: BEV-image captured from the GAZEBO simulator

In order to allow stacking of data in the downstream task of training the ACT models, for the ACT demonstrations we padded the non-camera data sources with zeroes when the goal was reached for each demonstration. We padded the camera image data and BEV-image data with black images. For the 2D-grid simulator we performed padding until each source had a length of 20 data points in total. For the GAZEBO simulator we performed padding until each data source would have a length of 200 data points in total. We set the padding length so that a demonstration trial of the maximum length would still fit in these sizes of 20 and 200 for each simulator respectively. The structure of the hdf5-files for the GAZEBO simulator can be found in the appendix (list 1 for ACT and list 2 for diffusion policy).

For both ACT and diffusion policy, we collected a total of 250 demonstrations for the 2D simulator and a total of 500 demonstrations for the GAZEBO simulator. For both simulators the first 250 demonstrations consisted of the task of navigation performed in an environment without a dynamic obstacle. For the remaining 250 demonstrations for the GAZEBO simulator, we performed the demonstrations in an environment in which a dynamic traffic vessel was present. By choosing these amounts of demonstrations we aimed to balance obtaining an accurate representation of the task being performed and minimizing dataset size. Rationale for trying to limit dataset size was again the limit in computational resources on the HPC-cluster node that was to be used for training the models. As can be seen in Figure 6, using 250 demonstrations with randomly sampled locations allowed for the entire environment to be sampled relatively densely.

For incorporating the traffic vessel component in the GAZEBO simulator, we used a readily available urdf-file from Roboat for the physical implementation of the vessel (Figure 10). A basic movement algorithm that we designed from scratch was applied to this vessel model. In this movement algorithm, we programmed the traffic vessel to keep going back and forth along one specific straight line trajectory in the simulator. Whenever the ASV would approach the traffic vessel close enough, the ASV would wait for the traffic vessel to be traveled out of range before resuming its trajectory. Furthermore, whenever the ASV would be close to this traffic vessel and in the line of travel of this traffic vessel, the traffic vessel would come to a stop and the ASV was ordered to keep moving even if it would be within the designated stopping range (algorithm 1).

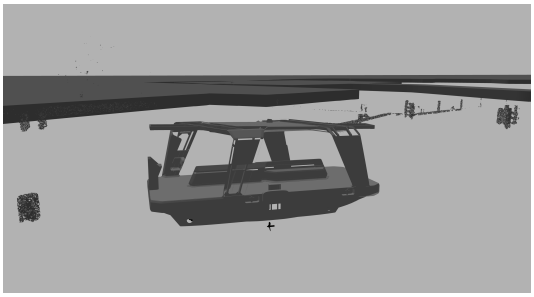


Figure 10: Physical appearance of urdf traffic vessel

Algorithm 1 Traffic Vessel Algorithm

$TrafficVesselMoving$ = true **and** $ASVMoving$ = true
if $ASV \subset$ TrafficVesselTrajectory **and**
    $ASVTrafficDist <$ MinDist **then**
    $TrafficVesselMoving$ = false
    $ASVMoving$ = true
else if $ASV \subset$ TrafficVesselTrajectory **then**
    $TrafficVesselMoving$ = false
else if $ASVTrafficDist <$MinDist **then**
    $ASVMoving$ = false
end if

Model Architectural Design

In terms of developing the model architectures, we took both the standard ACT and the standard diffusion policy as a starting point. This entailed taking the original algorithms of Zhao et al. [7] and Chi et al. [27] mentioned in section 3 and adapting these algorithms to enable training and inference with the collected hdf5-datasets from both simulators. These adaptations mainly meant adapting the dataloader components and adapting the input sources of the models. Both the standard ACT- and diffusion policy algorithm were open source available on GitLab [35], [36].

Implementing the necessary changes to diffusion policy was rather straightforward for us, since the model available on GitLab is built in a flexible manner in terms of defining the structure of the dataset on which the model is trained on. We defined the necessary data sources for diffusion policy by adapting the task related .yaml files, from which the data sources were read by the algorithm

before constructing the model. Additionally, we adapted the dataloader of the diffusion policy algorithm to enable data normalization for each of the specific data sources to allow for maintaining proper model training convergence.

In terms of ACT, adapting the model appropriately was slightly more elaborate for us. First of all, as with diffusion policy we had to adjust the dataloader to allow interpretation of the hdf5-file that the model was trained on. For ACT this meant we had to normalize each of the data sources by means of their respective mean and standard deviation and then passing the correct data sources by name for the training procedure. Furthermore, we had to adjust the encoder associated with the model its latent variable z to take in the appropriate data sources. By doing so we ensured that the algorithm did not use joint angles as an input as was used in the original algorithm, but used the sources corresponding to the new dataset. Additionally, we had to adapt the CVAE decoder to take in all the same proprioceptive data sources as the CVAE encoder.

Model Inference Design

In order to test model performance, we had to link the trained models to both the simulator variants. As a first step, this meant we had to implement the model libraries in the associated (robotic operating system) ROS-environment of Roboat in the form of python modules. By doing so the inference components could load the appropriate model architectures. Secondly, we developed the actual inference components of both ACT and diffusion policy for both simulators. In these inference components, we loaded the model architectures and we set the model parameters by loading in the checkpoint files of the trained models. In order to allow the model to make predictions for the 2D-simulator, we would queue the model for an output based on data sources retrieved straight from the simulator. In order to allow the model to make predictions for the GAZEBO simulator, we would queue the model for an output based on data sources retrieved from the simulator by means of the ApproximateTimeSynchronizer function with a slop value of 0.1 seconds. We set this parameter at a value of 0.1, the same as for the data loggers we designed. We ran the inference of ACT on a NVIDIA RTX A2000 GPU. We ran the inference of diffusion policy on a intel Core i9-11950H CPU due to VRAM limitations of the RTX A2000 GPU in the laptop on which the simulations were performed. We added the action output of this model, which was a sequence of relative poses to be added to the ASV pose at each time step, to the current pose of the ASV in the simulator. Hence, this way we constructed a sequence of navigation waypoints for the ASV to navigate towards. For the 2D-simularor these waypoints were navigated to directly. For the GAZEBO simulator we published the output of the models to the roboat local planner with a frequency of 10 Hz to allow proper linking to this planner. This planner was designed for interpreting trajectories optimally at this frequency. The local planner of Roboat was made to ensure the created trajectory would be straightened out so that it would adhere to the vessel dynamics of the concerning ASV if necessary.

6. Experiments

For being able to assess the model performances with respect to the task of ASV navigation in (complex) marine environments, we performed evaluative experiments in both the 2D-simulator environment as well as the the GAZEBO simulator environment. In these environments we identified the performance of the models by determining the success rate for ASV navigation towards a static goal position in the environment over 50 trials. In order to be able to determine the success rates of the models in these experiments, firstly the models were trained on the HPC-cluster node of the TU Delft. We performed our experiments for various combinations of hyper parameters for both algorithms in order to find their respective optimal hyper parameter configuration. Additionally, we used the optimal hyper parameter configurations found for each model to determine the ability of the models to accurately capture ASV dynamics and to determine their generalization capabilities in a series of tests.

In order to contextualize the performance of the models compared to other non-traditional (non-grid search-, non-MPC- or non-RL-based) state of the art ASV planning algorithms, we used a 2024 study by Wiersma et al. as a benchmark. In this study, deep reinforcement learning was applied to perform the task of ASV navigation towards a goal position. For an empty environment containing only a goal location, Wiersma et al. found a success rate of 50% for the model variant which had learned direct control and MPC behaviour.

6.1. 2D-grid simulator experimental setup

For the 2D-grid simulator we created a basic experimental setup in the simulator. We placed the goal location, which we marked as a red cross in the BEV-image, in the middle of the 2D-grid (coordinates 0,0) and we sampled the ASV starting location randomly on the grid. Subsequently, the ASV would attempt to reach the goal location within a margin of 1.5 distance units on the grid by applying the action sequences obtained from model inference. Whenever this margin would be reached, we spawned the ASV again at a random location for the next trial. When a model would require more than 40 inference outputs to reach the goal, we noted the trial as a failure. For these experiments, we ran 50 trials and we logged the corresponding success rate for both models.

6.2. GAZEBO simulator experimental setup

In order to determine the performance of both ACT and diffusion policy for the GAZEBO simulator environment, we applied a similar experimental setup as for the 2D-simulator. In this setup we spawned the ASV at a random location in the simulator while we placed the orange pole and the three blue static obstacles at a static location. For 50 trials we then let the ASV try to approximate the orange pole placed in the middle of the environment within a margin of 2 meters. Whenever the pole would be reached, we would respawn the ASV at a random location in the simulator again until the last trial was completed. Afterwards, we determined the success rate based on these trials.

After we determined the optimal set of hyper parameters for ACT and diffusion policy for the GAZEBO simulator environment, we evaluated the models for 50 trials in terms of the success rate in an additional scenario. This scenario contained two static obstacles in between the starting and the goal position and a dynamic traffic vessel moving back and forth in a straight line creating a very basic form of a complex marine environment.

6.3. Model Training

In order to train the algorithms, we used the HPC server node of the TU Delft. This node allowed us to train the models by means of a NVIDIA A40 GPU in order to allow time efficient training sessions for the ACT and diffusion policy hyper parameter studies. In order to train the algorithms, we had to upload and configure the models, the datasets and the associated conda environments on the HPC server. We accomplished this by making use of apptainer scripts, which allow for an organized set-up of the training procedure on the server. After each training procedure was completed, we retrieved the checkpoint files containing the tuned parameters of the models from the server and used these to build the ACT and diffusion policy inference components to evaluate the corresponding performance.

6.4. Hyper parameter setup

In order to fine tune the algorithms of ACT and diffusion policy, we performed a hyper parameter study for both models in which the performance of the models was evaluated when trained for various combinations of hyper parameter values.

The hyper parameter values of the models that we varied for ACT in this hyper parameter study were: chunk-size, and the sources for data input. We set the chunk size to the varying values of 2, 5, 10 and 20 (Table 2). According to the authors of the original ACT paper, tuning the chunk size towards a size corresponding to a wall clock time of 1 second of robot motion had proven to be optimal in their experiments. Since for this study data loggers were running at a frequency of 2 Hz, this meant having a chunk size of 2 was expected to show the best results. The combinations of input data sources that we trained for ACT can be found in Table 2. We established by looking at the training- and validation loss functions of the ACT training procedure for several hyper parameter combinations, that running 350-700 epochs in combination with a learning rate of 1e-6 and batch size 16 allowed for proper convergence of the model and plateau of the training- and validation loss functions (Figure 11). However, in a tuning tips document provided by the developers of the model on the corresponding Git page, it was stated to let the model train "well after things plateau" to allow for proper behaviour learning. The corresponding graph implied training it 4-5 times as long, since the graph plateaued at 300-700 epochs and ended at 2000 epochs. Hence, for this research we trained all ACT models for 2500 epochs and evaluated these for performance assessments.

For diffusion policy the hyper parameters that we varied were the number of action steps and the number of observation steps. The number of action steps we varied at 4 and 8, and the number of observation steps we varied at

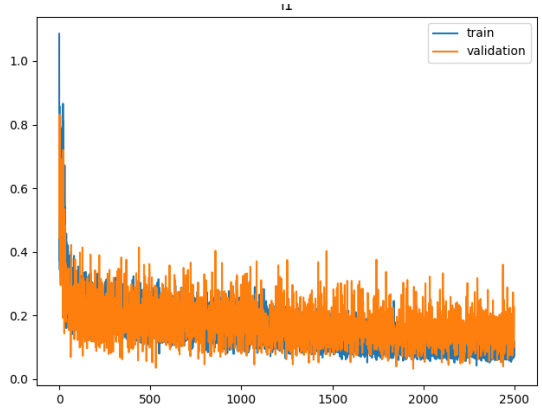


Figure 11: L1 loss function graph ACT

2 and 4. We trained the model for each hyper parameter configuration with a batch size of 32 for a total of 1000 epochs for each model, since multiple training session loss functions showed proper training convergence and plateauing of the loss functions for this amount of epochs. Additionally, the developers of the diffusion policy model stated proper convergence around 1000 epochs of training time [27]. The combinations of hyper parameters that we trained for diffusion policy can be found in Table 2 as well.

Additionally, we established the inference time of both models.

Method	Chunk Size / Action Steps	Obs. Steps	Data Input
ACT	2, 5, 10, 20	–	rel. goal pose + ASV orientation + ASV velocity + rel. ASV pose (action) + no image / front camera / three camera's / BEV-image
Diffusion Policy	4, 8	2, 4	rel. goal pose + ASV orientation + ASV velocity + rel. ASV pose (action) + no image / front camera / three camera's / BEV-image

Table 2: varied hyper parameters for ACT and diffusion policy

6.5. Debugging data logging

For potentially necessary debugging of the GAZEBO simulator data logging components, we performed an additional experiment for the 2D-grid simulator. In this experiment we compared found navigation success rates for the most optimal hyper parameter variation found for the ACT algorithm when we retrieved data either straight from the control loop or by means of the same ApproximatetimeSynchronizer as for the GAZEBO simulator. This way we could later on argue that whenever any of the models was able to capture the correct behavior for the 2D-simulator, but not for the GAZEBO simulator for any of the loggers, the issue probably lies in the lack of high quality data sources being fed to the model from the GAZEBO simulator to represent the more complex behaviour. Whenever, we would find the model does not yield similar success rates for navigation in the 2D-simulator for both the regular logger and the ApproximateTimeSynchronizer, the ApproximateTimeSynchronizer logger probably does not function properly and needs to be debugged.

6.6. Additional performance metrics

In the sections below the applied method for determining the performance of the models in terms of copying dynamics and

displaying generalization capabilities can be found.

copying of dynamics

In order to assess the ability of copying the dynamics by both models, we made two visualization output types for the optimally found hyper parameter configurations based on success rate. These visualization outputs consisted of: a streamline plot of navigation trajectories towards the goal for 250 trials and a heatmap describing the number of steps required for the models to converge towards the goal location for 250 trials. By comparing the visualizations found during inference for these models with the visualizations from the original trained on navigation behaviour, we could gain insight in the ability of the models to capture and understand the correct navigation dynamics. Additionally, a series of trajectory density plots (Figure 23, Figure 24 and Figure 26) and corresponding success rate heatmaps (Figure 25 and Figure 27) were generated for an equidistantly sampled 20x20 grid (Figure 22) in terms of starting positions. These plots were made in order to be able to identify whether specific trajectories were preferred over other ones by both model types. Due to close resemblance to the mentioned streamline plots, these plots have been included in the appendix section.

Generalization tests

In order to assess the generalization capabilities and profound understanding of the navigation task and its corresponding dynamics by both models, we made two main visualizations for each of the model types. Firstly, we visualized the success rates for the model outputs not only in the area that contained the original starting locations of the trained on demonstration dataset, but also for a small outer ring area extending the simulator environment with 15 distance units/ meters (depending on the simulator) in both directions with respect to both the x- and the y-axis. This way we could test the level of understanding of the task of navigation even outside of the original training area by the models. Additionally, we expanded the streamline plot for evaluating the ability to capture dynamics in this outer region by the models. This way we could assess the ability of the model to yield the correct dynamics even for a region on which the model was not originally trained and that resides out of the trained on distribution.

7. Results

In the sections below the results for the 2D-grid simulator and GAZEBO simulator experiments can be found.

7.1. Hyper parameter studies

The 2D-simulator trained ACT algorithm showed varying success rates depending on the configuration of hyper parameters applied. As can be seen we found the highest success rate (0.98) for the variation with the action chunk size of 5 and the BEV-image input. This variation also showed a significantly higher success rate than Wiersma et al. found for applying deep reinforcement learning for a similar experiment (0.50), suggesting superiority of ACT models for learning this type of navigation task. In terms of the other ACT trained algorithms with BEV-image input, the score is only slightly higher than what we found for the version with chunk size 10 (0.94), but significantly higher than what we found for the chunk size 2 (0.66) and chunk size 20 (0.50) variants. Furthermore, we found all non-BEV-image model variants to yield a 0.0 success rate in reaching the goal location, suggesting high importance of this data source to the model for adequate interpretation of the navigation task to be learned. Furthermore, by looking at Figure 13 one can see that the model learns how to navigate correctly towards the goal location for the majority of the grid regions. However, the lower left part of the 20x20 grid in terms of starting position appeared to yield very low success rates in reaching the goal position. This suggests the model is not able to fully grasp the navigation task to be learned given the trained on data sources.

The 2D-simulator trained diffusion policy algorithm also showed significantly different success rates depending on the configuration of the hyper parameters applied. As can be seen we found the highest success rate (0.53) for the variation with 2 observation steps and 4 action steps. This is only a slightly higher success rate than Wiersma et al. found for applying deep reinforcement learning for a similar experiment (0.53), suggesting similar capabilities in learning this task to a diffusion policy model. As for the other trained diffusion policy algorithms without BEV-image input, the score is only slightly higher than what we found for the version with 4 observation steps and 8 action steps (0.50), but significantly higher than what we found for the 2 observation steps and 4 action steps (0.3) and the 4 observation steps and 4 action steps variants (0.18). These results suggest using a higher number of action steps than observation steps for attaining highest possible success rates. Unfortunately, due to RAM limitations on the HPC cluster of the TU Delft, it proved to be impossible to train the BEV-image variants for diffusion policy for us. Hence, no success rates for these hyper parameter variants have been established for diffusion policy. Furthermore, by looking at figure 15 one can see that the model learns how to navigate correctly towards the goal location for only a small region of the grid. Merely, a downward sloped striped region in the middle part of the 20x20 grid in terms of starting position appeared to yield adequate success rates in reaching the goal position. This might suggest that the the essence of the navigation task is not fully grasped by the model.

In Table 5 one can see the success rates we determined for the different hyper parameter configurations for the ACT model for navigation in the GAZEBO simulator. Unfortunately, for none of the trials for any of the hyper parameter configurations the model of ACT was able to navigate the ASV adequately towards the orange pole. Hence, for each configuration we found a 0.0 success rate. These findings suggest the model is unable to capture the more complex task of navigation corresponding to the GAZEBO simulator environment given the trained on data sources.

As can be seen when comparing Table 5 and 6 with Table 3 and 4, the success rates we determined were significantly lower for ACT for the 2D-simulator experiments than for the GAZEBO simulator experiments. This results suggests that the quality an/or quantity of the data sources being fed to the model from the GAZEBO simulator was insufficient to represent the more complex behaviour associated to the GAZEBO simulator navigation task.

Unfortunately we could not establish the success rates for the diffusion policy model for the GAZEBO simulator environment (Table 6). Due to limited GPU resources of the hardware on which experiments were performed, we could not test diffusion policy inference performance adequately. The laptop on which we ran our experiments contained a NVIDIA RTX A2000 GPU, which proved to contain too few VRAM capacity to allow the model to run the diffusion policy inference on this GPU. Because of this, we attempted to run our experiments on CPU resources (Core i9-11950H). However, this would lead to very slow inference times (10-15 minutes depending on the set hyper parameters) and would eventually lead to the ROS memory manager killing off the node, disabling the ability of testing the performance for the GAZEBO simulator.

Due to the low success rates we found for the GAZEBO simulator tests, we only performed the experiments in terms of copying ASV dynamics and showing generalization capabilities for the 2D-simulator environment.

Component	Success Rate
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 2	0.66
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 5	**0.98**
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 10	0.94
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 20	0.50
input: rel. goal pose, ASV velocity, chunk size: 2	0.00
input: rel. goal pose, ASV velocity, chunk size: 5	0.00
input: rel. goal pose, ASV velocity, chunk size: 10	0.00
input: rel. goal pose, ASV velocity, chunk size: 20	0.00

Table 3: hyper parameter study ACT 2D-grid

Component	Success Rate
input: rel. goal pose, ASV velocity, obs steps: 2, action steps: 4	**0.53**
input: rel. goal pose, ASV velocity, obs steps: 2, action steps: 8	0.30
input: rel. goal pose, ASV velocity, obs steps: 4, action steps: 4	0.18
input: rel. goal pose, ASV velocity, obs steps: 4, action steps: 8	0.50
input: rel. goal pose, ASV velocity, BEV-image, obs steps: 2, action steps: 8	-
input: rel. goal pose, ASV velocity, BEV-image, obs steps: 2, action steps: 16	-
input: rel. goal pose, ASV velocity, BEV-image, obs steps: 5, action steps: 8	-
input: rel. goal pose, ASV velocity, BEV-image, obs steps: 5, action steps: 16	-

Table 4: hyper parameter study diffusion policy 2D-grid

Debugging data logging

As can be seen in Table 7, we found that using any of the two logging variants for capturing datasets resulted in similar ACT success rates for the 2D simulator (0.98 for direct data extraction and 0.83 for AproximateTimeSynchronizer extraction). Albeit, that we found a slightly higher success rate for direct extraction than for the time synchronizer, this small difference suggests the method of data logging was not the main limiting factor for learning appropriate navigation behaviour for both model types.

7.2. Additional performance metrics

In order assess the ability of both ACT and diffusion policy to understand the essence of the navigation behaviour to be captured, we also assessed the models on their ability to copy vehicle dynamics and on their ability to perform the task appropriately outside of the trained on distribution to assess the generalization capabilities of the models. The results we

Component	Success Rate
input: rel. goal pose, ASV velocity, front camera, chunk size: 2	0.00
input: rel. goal pose, ASV velocity, front camera, chunk size: 5	0.00
input: rel. goal pose, ASV velocity, front camera, chunk size: 10	0.00
input: rel. goal pose, ASV velocity, front camera, chunk size: 20	0.00
input: rel. goal pose, ASV velocity, three camera's, chunk size: 2	0.00
input: rel. goal pose, ASV velocity, three camera's, chunk size: 5	0.00
input: rel. goal pose, ASV velocity, three camera's, chunk size: 10	0.00
input: rel. goal pose, ASV velocity, three camera's, chunk size: 20	0.00
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 2	0.00
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 5	0.00
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 10	0.00
input: rel. goal pose, ASV velocity, BEV-image, chunk size: 20	0.00
input: rel. goal pose, ASV velocity, chunk size: 2	0.00
input: rel. goal pose, ASV velocity, chunk size: 5	0.00
input: rel. goal pose, ASV velocity, chunk size: 10	0.00
input: rel. goal pose, ASV velocity, chunk size: 20	0.00

Table 5: hyper parameter study ACT GAZEBO

Component	Success Rate (%)
input: rel. goal pose, ASV velocity, front camera, action steps: 2, obs. steps: 2	-
input: rel. goal pose, ASV velocity, front camera, action steps: 4, obs. steps: 2	-
input: rel. goal pose, ASV velocity, front camera, action steps: 2, obs. steps: 4	-
input: rel. goal pose, ASV velocity, front camera, action steps: 4, obs. steps: 4	-
input: rel. goal pose, ASV velocity, three camera's, action steps: 2, obs. steps: 2	-
input: rel. goal pose, ASV velocity, three camera's, action steps: 4, obs. steps: 2	-
input: rel. goal pose, ASV velocity, three camera's, action steps: 2, obs. steps: 4	-
input: rel. goal pose, ASV velocity, three camera's, action steps: 4, obs. steps: 4	-
input: rel. goal pose, ASV velocity, BEV-image, action steps: 2, obs. steps: 2	-
input: rel. goal pose, ASV velocity, BEV-image, action steps: 4, obs. steps: 2	-
input: rel. goal pose, ASV velocity, BEV-image, action steps: 2, obs. steps: 4	-
input: rel. goal pose, ASV velocity, BEV-image, action steps: 4, obs. steps: 4	-
input: rel. goal pose, ASV velocity, action steps: 2, obs. steps: 2	-
input: rel. goal pose, ASV velocity, action steps: 5, obs. steps: 2	-
input: rel. goal pose, ASV velocity, action steps: 2, obs. steps: 4	-
input: rel. goal pose, ASV velocity, action steps: 2, obs. steps: 4	-

Table 6: hyper parameter study diffusion policy GAZEBO

Model	Success Rate
ACT (2D-grid simulator)	**0.98**
ACT (GAZEBO simulator)	0.00
Diffusion policy (2D-grid simulator)	0.53
Diffusion policy (GAZEBO simulator)	-
Deep Reinforcement Learning (Wiersma et al.)	0.50
ACT (2D-grid simulator + ApproxTimeSynchronizer)	0.83

Table 7: Model success rates compared to benchmark

Model	Inference time per model type (s)
ACT	0.5
Diffusion policy	4.4

Table 8: Inference times 2D-grid simulator

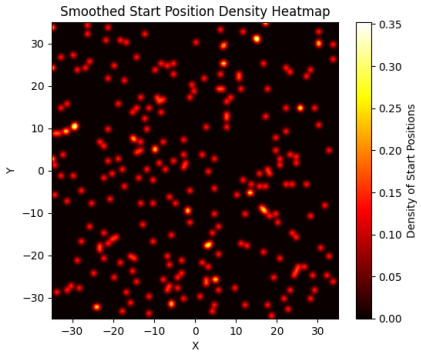


Figure 12: Starting state for 250 trials using ACT inference

found for these tests can be found in the sections below.

copying of dynamics

We found that the ACT model is able to capture the appropriate navigation task and corresponding LQR-dynamics quite accurately for the 2D-simulator as can be seen in the concerning streamline plot (Figure 17). For most of the regions on the grid the model is able to converge the ASV towards the goal position with a similar trajectory as the LQR-controller. The ACT streamline plot displays a straight line navigation trajectory for starting positions removed significantly far from either the x- or y-axis. Furthermore, the model displays a priority in navigating towards any of the closeby axes for starting locations close to the x-or y-axes. Additionally, the the model appears to be able to copy the behaviour of maximum velocity for the 20x20 region adequately as well, displaying similar velocities for each region with respect to the original 2D-LQR dynamics streamline plot. The most remarkable feature of this plot are the trajectories generated in the bottom part of the 2D-grid. The trajectories in this area contain a wide downward sweep before returning on their path towards the goal. This behaviour is not in line with the trained on 2D-LQR dynamics and indicates the model making wrong assumptions regarding the dynamics for these locations and not fully grasping the dynamics to be learned. Additionally, as can be seen in Figure 13 the success rates for these corresponding locations are found to be much lower as well, suggesting the same notion.

Furthermore we found that the model is able to capture the step size corresponding to the original LQR-controller relatively accurately as can be seen in Figure 20. This heatmap plot displays the expected pattern for most of the 20x20 region of requiring more steps to converge towards the goal when starting further from the goal as the original LQR controller. However, some areas in the environment showed a slight deviation from this pattern, particularly the lower left part. For these regions more steps were necessary to reach then goal position than for the LQR-controller.

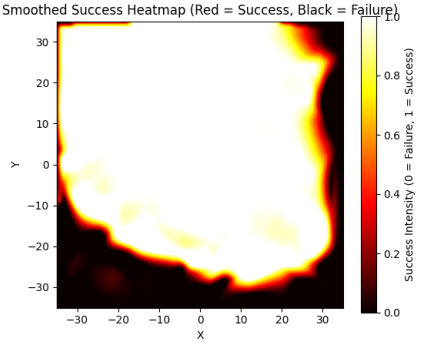


Figure 13: Success rate heatmap for 250 trials using ACT inference

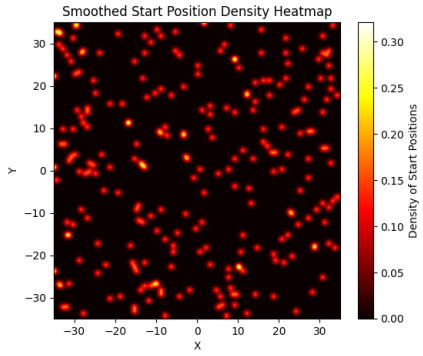


Figure 14: Starting state for 250 trials using diffusion policy inference

We remark this region overlaps slightly with the regions of lower success rates and deviated dynamics mentioned before. These findings suggest limitation of the ACT model to profoundly grasp the dynamics to be learned.

As can be seen in the diffusion policy streamline plot, we found that the diffusion policy model is not as good in capturing the LQR-dynamics as the ACT model. The trajectories traversed for navigation towards the goal location appear smeared out compared to the 2D-LQR dynamics trajectory pattern. The phenomenon of straight line trajectories towards the goal at any of the quadrants is somewhat recognizable, however the trajectories seem to be twisted as to form a leftward inward spiral as a general trajectory pattern. Furthermore, for starting positions close to the y-axis, the ASV does not prioritize converging to the axes first as is the case for the LQR-controller. However, this converging behaviour can be recognized for starting locations close to the x-axis. Additionally, the model does not appear to grasp the velocities corresponding to the designed controller as well, and either yields actions that allows the model to surpass these velocities or stay below them. Lastly, another undesirable result that one can recognize in the streamline plot upon closer inspection is that the streamline trajectories in the upper half of the plot and in a region of the lower right quadrant of the grid do not converge towards the intended goal location of 0,0. Instead these streamlines end at a location slightly to the upper right with respect to the origin. This behaviour might explain the low success rates found for diffusion policy for the 2D simulator. This notion is supported by remarking that these regions overlap with the low success rate regions

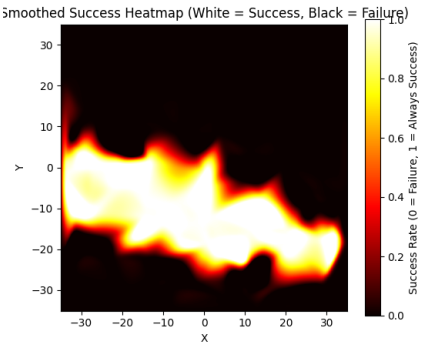


Figure 15: Success rate heatmap for 250 trials using diffusion policy inference

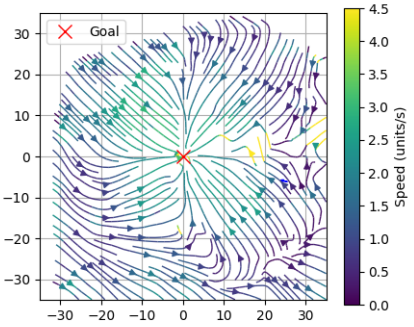


Figure 17: streamline plot for 250 trials for ACT inference

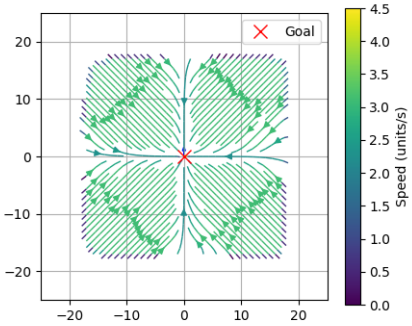


Figure 16: streamline plot for 250 trials for 2D-LQR dynamics

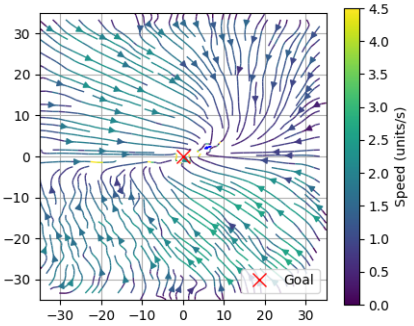


Figure 18: streamline plot for 250 trials for diffusion policy inference

found in the success rate heatmap of diffusion policy for the 2D simulator (Figure 15). These findings suggest limitation of the diffusion policy model to accurately understand the dynamics to be learned. However, it can be recognized from these plots, the model is able to grasp the core concept of the navigation task towards the middle region of the grid well, since the majority of the stream lines end at roughly the goal location.

Furthermore we found that the model does not capture the step size corresponding to the original LQR-controller accurately as can be seen in Figure 21. This heatmap plot displays the same downward slope stripe pattern of requiring more steps to converge towards the goal as the success rate heatmap for diffusion policy showed in terms of successful navigation in Figure 15. This again suggests the model is not able to grasp the essence of the dynamics to be learned given the trained on data sources. Manual inspection of the ASV state as it converged towards the goal showed adequate navigation of the ASV until it came within about 7.5 units distance with respect to the goal location. Whenever the ASV would approach the goal location closely, the size of the generated action steps would decrease significantly. This lead to the model requiring much more than the limit of 40 steps to achieve the goal location within the 1.5 unit margin, explaining the low success rates found across the grid.

Generalization tests

In terms of generalization performance, we found the ACT model is able to extend the high success rate to locations out of the trained on 20x20 grid as can be seen in Figure 3. However only the upper left quadrant and slightly upper right quadrant display this generalization capability. In the other outside regions success rates quickly taper off to 0.0, suggesting limitation in capturing the essence of the navigation task.

In terms of capturing the dynamics outside of the trained on 20x20 region of grid, we found the ACT model showcases some generalization capability. The model displays very similar streamline behaviour for the outer region of the left upper quadrant of the grid as for the 20x20 region (Figure 18), showcasing some abilities in understanding the essence of the navigation dynamics. However, for the other regions multiple disturbances can be recognized in each quadrant of the outer region of the grid. The lower quadrants appear to generate an ever wider downward sweep of the trajectories as was found for the inner region of the grid. While the upper right quadrant appears to show correct dynamics, prioritizing convergence towards the x-axis was not understood correctly by the model. In terms of extending the maximum velocity property of the controller towards the outward region, the model shows poor performance. For most of the grid, velocities exceed expected velocities, especially in the right upper quadrant in which velocities attain values of 4.5 units per second. These findings suggest limitations in terms of generalizing and capturing the essence of the dynamics by the ACT model. In terms of steps required, the model appears to follow the same trend as we found for the extended success rate region (Figure 13). The model appears to be able to extend the dynamics in terms of step size for the regions it did yield adequate success rates for to the outward regions suggesting some generalization

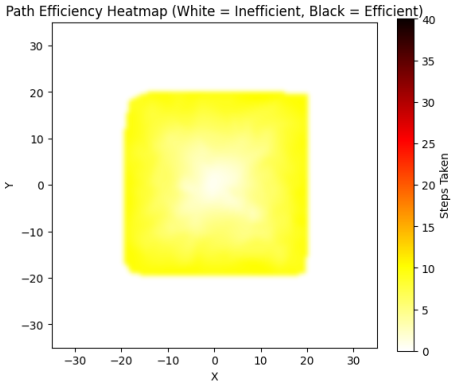


Figure 19: steps required for 250 trials for 2D-LQR dynamics

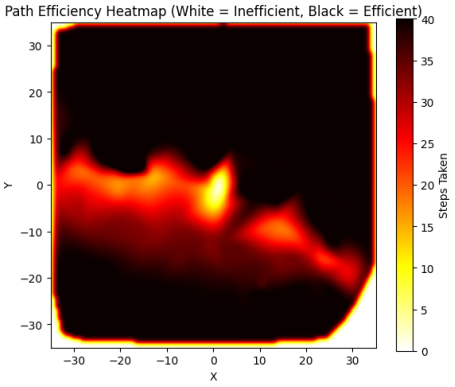


Figure 21: steps required 250 trials inference diffusion policy

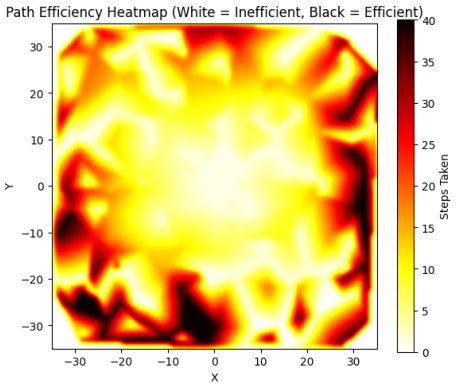


Figure 20: steps required 250 trials inference ACT

capabilities. Each outside of training distribution region, besides the upper right one, shows an unexpectedly large increase in steps required which form the dark areas in the plot.

In terms of generalization performance, we found the diffusion policy model is also able to extend success rates only with respect to the downward slope middle region in the 2D-grid identified earlier (Figure 15). It can be seen the out of grid regions that yield success are joined to this main success rate region in the grid found for diffusion policy. At the other out of grid regions, success rates of 0.0 were determined. This suggests some generalization capabilities and profound understanding of the navigation task interpreted by the model with regard to ASV navigation, however only for regions where the task has been captured significantly adequate.

In terms of capturing the dynamics outside of the trained on 20x20 region of grid, we found the diffusion policy model showcases good generalization capability. The model displays similar streamline dynamics and velocities for the majority of the outer region as for the 20x20 region (Figure 18), showcasing abilities in understanding the essence of the navigation dynamics as it was interpreted by the model. However, in the outer region of the lower left quadrant, a slight distortion can be seen for the associated stream line trajectories, suggesting some limitation in terms of this generalization capability. In terms of steps required, the model appears to follow the same trend as we found for the extended success rate region (Figure 15). The model appears to be able to extend the dynamics in terms of step size for the regions it did yield adequate success rates for to the adjacent outward regions suggesting some generalization capabilities by the model.

8. Discussion

In the following sections the interpretation of the results and the limitations and the recommendations regarding the results can be found.

8.1. Interpretation of results

In the sections below we elaborate on the results found for both the 2D-grid simulator and GAZEBO simulator trained models.

2D-grid simulator

In the sections below we elaborate on the results found for the 2D-grid simulator.

ACT

By looking at the hyper parameter study results for ACT for the 2D-simulator (Table 3), one can recognize the importance of including the BEV-image, or perhaps image data in general, in the data sources for yielding an appropriate action output sequence by the model. Apparently leaving out this data and merely conditioning the action output on the relative goal position and ASV velocity is simply not enough to get adequate action outputs. These findings suggest the BEV-image contained information inherently that was necessary for the model to figure out the nature of the task that was supposed to be learned. It might have been possible this inherent information was the relative location of the ASV with respect to the frame of the 2D-grid. For the currently trained algorithm, the model was trained on merely information regarding its relative position with respect to the goal and its velocity. However, it may have been unclear to the model in which direction the ASV was supposed to navigate in order to approach the goal location. Hence, these tests showcase the importance of adding BEV-image data, or perhaps image data or ASV-orientation data in general, to train an ACT model to achieve significant success rates for learning the task of ASV navigation.

As can be seen in Table 3, the optimal hyper parameter configuration concerns an action chunk size of 5. This is a slightly different result as to what was expected. Based on the tuning tips document of the authors of the ACT paper,

applying a chunk size corresponding to a wall-clock time of 1.0 seconds would show optimal performance. However, given that the timestep difference between action steps was set to 0.9 seconds, these results suggest using a chunk size corresponding to a 4.5-9.0 seconds wall clock time in order to learn the task of ASV path planning to an ACT model optimally.

The reason as to why the model was unable to capture the correct dynamics for the lower (left) part of the environment remains hard to identify. As can be seen in Figure 6 the sampling of starting point locations for training is most likely not the cause of this, since each region of the grid appears to be sampled with similar densities. Although it remains conjecture, it might have been that the goal locations randomly assigned to these starting locations in the demonstration dataset might all have been not close to the origin point, meaning that this specific task has been underrepresented for some regions of the grid in the dataset causing this phenomenon. Hence, adding more than 250 demonstrations of the task to be performed, thus covering a wider distribution of the task to be learned in terms of starting- and goal location, might increase the success rates of ASV navigation performed by an ACT model. If this proves to be the limiting factor for achieving adequate success rates in a potential future work study, this would be another indicator for limited generalization capabilities of an ACT model in terms of capturing the task of ASV navigation.

As for the determined generalization capabilities for the ACT model, it can be seen in Figure 13 and Figure 17 that the ACT algorithm performs well for the upper left part of the environment, however not so much for the rest of the environment. The reason as to why this is remains hard to identify. As for copying the dynamics it might have been that the goal locations randomly assigned to the starting locations corresponding to the low success rate regions outside of the trained on distribution might all have been not close to the origin point. This might have led to the fact that this specific origin navigation task for these locations has been underrepresented in the dataset causing this phenomenon. Additionally, it might be possible that providing merely 250 demonstrations of the task for training was simply not enough to figure out the essence of the ASV navigation and its corresponding dynamics. Increasing the number of demonstrations for training the model might solve these discrepancies across the grid environment. If this proves to be the limiting factor for achieving adequate dynamics and success rates outside of the trained on distribution in future work studies, this would be another indicator for limited generalization capabilities of an ACT model in terms of capturing the task of ASV navigation.

Diffusion policy

In terms of the success rates found for the 2D-simulator diffusion policy trained model (Table 4), the model performance can be assessed to be quite low in terms of ASV navigation. As we stated before, we expect the reason for this to mainly lie in the identification of a shifted goal location for a large portion of the 2D-grid by the model. We expect this wrongly interpreted behaviour by the model explains the established success rate heatmap for the 2D simulator for diffusion policy (Figure 15). Additionally, the reason partially lies in the large amount of action steps required to converge towards the goal location for the last part of the trajectory. By inspecting the traversed trajectories per demonstration manually one could see a trend in which the action step size or speed was appropriate for relatively long distances with respect to the goal location. However, as the ASV would approach the goal more closely, the model would yield increasingly smaller action steps so that the ASV would never come close enough to the goal location for the trial to be considered successful (<1.5m distance). Pinpointing the exact reason as to why this decrease in velocity during convergence towards the goal arises remains conjecture. However, as a matter of fact, control input of the orginal LQR-controller is computed as $-K$ (computed by the Ricatti equation), multiplied by the state error. This ensures that also in the original 2D-LQR dynamics behaviour the iteration step size (or action) lowers gradually as the ASV approaches the goal. However, since the time step taken per iteration ΔT is relatively large in the automatic demonstration, it is ensured the final step towards the goal of each demonstration is still relatively large. Subsequently during training, it might be possible the model learns to interpolate this decrease in step size over the grid-map, ensuring the ASV simply does not converge with an adequate amount of steps to the goal position during model inference. Additionally, the limiting factor here might have been the fact that the model's forward diffusion steps and denoising steps were changed from 100, (as it was in the original algorithm from Chi et al.) to the value of 50. As stated before this was done, to reduce burden on the training resources so that the diffusion policy models could be trained on the HPC-cluster. However, it might have been possible this number of steps for both forward- and backward diffusion might not have been enough to yield an accurate action sequence by the model.

Additionally, pinpointing as to why the found dynamics represents more a leftward spiral pattern as opposed to the 2D- LQR dynamics pattern expected remains hard as well. The model simply does not capture the dynamics perfectly. As for the ACT model it might have been possible that the goal locations randomly assigned to the starting locations corresponding to the low success rate regions outside of the trained on distribution might all have been not close to the origin point, resulting in an incomplete distribution of starting- and corresponding goal states in the training dataset. Additionally, again the limited amount of diffusion steps may have been the cause of this.

In terms of generalization capabilities the model of diffusion policy performed quite well. It showed ability to apply more or less the same dynamics behaviour in the outer ring region of the grid as for the trained on inner region. However, still for most of these cases the success rates remained close to 0.0 for these outside regions for the same reasons of a second wrongly identified goal location and decreasing action size over distance with respect to the goal.

Comparing the results of the 2D-grid simulator of both ACT (Table 3) and diffusion policy (Table 4) yields another interesting insight. ACT did require the BEV-image to yield

any success rates for the navigation tasks, however diffusion policy did not require these images to learn the navigation behaviour relatively adequately. The reason for this remains conjecture. However, it could be possible that the incorporated observation steps in the diffusion policy model might have been the reason for this. Since diffusion policy gets multiple steps of history as an input for its inference output, it might be able to figure out the orientation with respect to the goal position due to this history and thus might figure out which way its actions have to be directed to converge to the goal location. Since the ACT models lacked both orientation data and multiple observation steps, there might have been no way for the model to figure out which way to move to to approach the goal position. This might also be a motivation to adapt the ACT model in future studies to incorporate multiple observation steps and evaluate the performance.

GAZEBO simulator

As can be seen in Table 5, the established success rates for ACT for the GAZEBO simulator were very poor. For each hyper parameter variation only a 0.0 success rate was found. It might have been possible that the navigation task to be captured was simply too complex for the model to understand given the data that was used to train it. As could be seen in Table 7 the data logging system most likely was not the factor causing the low success rate values, given the fact that quite similar success rates were found for direct logging (0.98) as for leveraging the ApproximateTimeSynchronizer for the task of logging (0.83). The reason for yielding low success rates is expected to be the quality or the amount of conditioning data sources, since the model architecture was left unchanged with respect to the original ACT algorithm [7], combined with the higher complexity of the navigation to be learned. However, the only data source that was not comparable to the ones leveraged for training the ACT algorithms from Zhao et al. was the camera data. For achieving the success rates in the original ACT paper, four photorealistic camera images were leveraged for training the model. However for this research, collected camera data was of a lot poorer quality. This camera data contained vision input of generally low texture from a greyscale environment of the GAZEBO simulator (Figure 8a and Figure 8b). This might have been a big limitation for the models to interpret and understand the environment and hence yield proper action outputs.

As the results for diffusion policy, since success rates could not be established in this research for the GAZEBO simulator, no elaborative comments and/or statements can be made about its performance in this more complex environment.

Assessment regarding ASV path planning

Based on our experiments, it appears that the ACT model is not able to perform the task of realistic ASV navigation with desirable performance given the trained on data sources. However, for a less realistic and basic navigation task, the model can achieve close to perfect success rates (0.98). Additionally, results showed that when trained for a less complex navigation task, the model is able to accurately capture the corresponding navigation dynamics. It should be noted that limited generalization capabilities of the model ensures these adequate navigation performances are merely present when the ASV is released in an environment inside the trained on distribution. As we stated in the results, the generalization capabilities we found for ACT regarding ASV navigation are limited.

Diffusion policy showed inferior success rates with respect to ACT and was only evaluated for the more basic navigation task of the 2D-simulator. However, we assess its promise with regard to ASV navigation (in complex marine environments) to be high. The model was already able to capture the essence of the navigation task with a significant data source less than the ACT model (BEV-image), showing converging behaviour towards the goal location region in pretty much every tested region outside of the trained on distribution. We expect the main limitation of achieving success rates in the 2D-simulator for diffusion policy mainly lies in the identification of a second slightly shifted goal state for a portion of the ASV starting locations and in the design of the LQR-controller for performing the automatic demonstrations. We remark that the second goal state limitation and the fact that the model displayed a slight but significant deviation of the dynamics to be learned, suggests fine tuning of the models its design might be necessary in order to be able to apply it for ASV navigation in real life.

Due to the failed GAZEBO simulator tests in terms of success rates (ACT) and training the model due to computational restrictions (diffusion policy) merely performance results for simple environments and simple navigation behaviour without dynamic obstacles were collected and evaluated. The more complex environments, including dynamic obstacles such as traffic vessels, have not been investigated in depth in this study. Hence in terms of being able to perform ASV navigation in complex marine environments no elaborate comments and/or statements can be made. In future work studies, the performances of the models of ACT and diffusion policy in these environment types could be evaluated.

8.2. Limitations

In this section, we analyzed the limitations of this study with respect to the four major parts regarding the applied methodology. This means that for the simulator and corresponding constructed demonstration loop, the method of data logging, the model training methodology and the construction of the inference components the limitations and recommendations where established and elaborated on. Additionally, we elaborate on the limitations of a potential ACT or diffusion policy ASV path planner for complex marine environments.

Simulator and demonstration loop

In terms of how the automatic demonstrations where performed in the simulator, there are some notable limitations. Firstly, it should be noted that the ASV's trajectory planner applied for navigation during the automatic demonstration loop of the GAZEBO simulator was not 100% successful in avoiding collisions with the static obstacles. During some trials, the planner still collided with these obstacles, bumping them out of the way. This did not happen often, estimated around once in 75-100 demonstrations. However,

it ensured that the demonstration datasets might not have been 100% clean of correct demonstrations with regard to obstacle avoidance and hence the correct obstacle avoidance might not be perfectly represented.

Additionally, it should be remarked the currently performed automatic demonstrations for the GAZEBO simulator were representative of a relatively complex ASV navigation task. The demonstration loop was programmed so that the vessel would navigate towards a static orange pole goal location with static obstacles present in the environment. Hence, not the most basic task of navigation possible has been learned to the models. It might be the case that when the model is trained on demonstration data representing the more simple task of strictly navigation without obstacle avoidance, any of the two models might be able to capture the correct behaviour correctly. Furthermore, the behaviour to be learned is also extra complex due to the incorporated vessel dynamics to the path planner currently used for navigation in the GAZEBO demonstration trials.

Logging of data sources

There are also some noteworthy limitations in terms of the nature of and the way in which the data was captured for the training datasets from the simulator. In particular the camera data that was retrieved from the GAZEBO simulator might have been a limiting factor for both model performances. In order to achieve the success rate of the original ACT paper [7], Zhao et al. retrieved camera data from four photorealistic camera's in their corresponding simulator environment. The original diffusion policy paper used only one camera data source (BEV-image) [27], but still of high photorealistic quality. However, for this research collected camera data was of a lot poorer quality. This camera data collected vision input from the generally low texture and grey environment from the GAZEBO simulator (Figure 8a and Figure 8b). This might have been a big limitation for the models to interpret the environment on each time steps and hence yield proper action outputs.

Additionally for this research, data was captured from the simulator at a frequency of 2 Hz. However, the influence of the sampling frequency was not tested in terms of performance of the models. It might be that the chosen sampling frequency yields suboptimal action values for the models to learn and interpret the correct behaviour to be learned.

Model training

For training the models, merely 250 demonstrations were collected per task. This amount was chosen with the HPC node restrictions in terms of RAM-size and GPU in mind. Additionally, the original ACT paper algorithms were trained on datasets of merely 50 trials, while the original diffusion policy paper algorithm were trained on 200 demonstrations, suggesting the notion this amount of demonstrations would be appropriate. However, it might be the case that the complexity of the task of ASV navigation requires a higher amount of samples to accurately represent the task for the models. Hence, it might be the case that the amount of 250 demonstrations chosen for the model training was a suboptimal parameter choice.

Additionally, during the hyper parameter study not all possible hyper parameter configurations were investigated. Merely the ones that focused on model in- and outputs have been varied for finding the optimal configuration. The task of ASV navigation was deemed a task not more complex than the ones achieved with high success rates in the original papers (tying shoe laces and putting on a small lid for ACT and performing accurate hanging of a tool for diffusion policy). Because of this it was assumed the that original model architecture and training procedure parameters would suffice for learning the task of ASV navigation.

Additionally, it was hard to establish precisely when the proper amount of epochs was achieved for ACT. Due to the way the validation- and training loss functions were constructed the train loss and validation loss were not trustworthy to directly assess model parameter convergence. Because of this, the models were trained for 4-5 times the amount necessary for convergence as was advised by the authors of the original ACT paper of Zhao et al.. Generally, the ACT models would convergence around the amount of 350-700 epochs for the task of ASV navigation. Because of this, the ACT models were trained for a total 2500 epochs for each hyper parameter configuration.

Furthermore, diffusion policy with image data input could not be trained properly due to RAM and GPU limitations on the HPC-server. Merely running the training steps with a batch size of 1 and cutting the camera images to much smaller values of 50x50 pixels instead of 240x240 pixels, allowed the model to be trained given the available resources for a maximum of 200 epochs, which was not enough to allow the loss function of the model to plateau adequately.

Inference components

In the currently designed GAZEBO inference components, new actions are queued from the models even during action sequence output execution. However for the original results of Zhao et al. for ACT and of Chi et al. for diffusion policy, a smooth continuous trajectory is generated by averaging overlapping action sequences during execution (designated as ensembling for ACT). However, for the inference components of the GAZEBO simulator, overlapping actions are not averaged, which can perhaps result in inadequate state transitions for the ASV regarding its dynamics. This was however not an issue for the current experiments due to a component of the GAZEBO simulator straightening out the trajectory conditioned on the vessel dynamics. However, for more widespread adaptation of the component, the action outputs may lead to impractical state transitions.

8.3. Generalizability and reproducibility

Regarding the generalizability of the application of ACT and diffusion policy as ASV path planners in complex marine environments, we can make only limited statements of the tests performed and the corresponding results.

As stated before, for ACT it appears that the model is not able to perform well in a more realistically complex navigation task given the current model architecture and

data sources. Hence, it is to be expected that releasing the ASV in a different semi-realistic simulator environment scenario (with or without inclusion of dynamic obstacles) when being trained on the same single GAZEBO scenario will result in success rates of 0.0 once again. Additionally, since the generalization tests of ACT showed limited results for performing the task of ASV navigation in the 2D-simulator outside of the trained on distribution, these success rates are expected to be low. Thus, generalizability of the currently designed solution of ACT for an ASV path planner is estimated to be limited. In order to further analyze the generalizability of the ACT model for ASV navigation, in future studies one could try to apply a model trained on the less complex navigation of the 2D-simulator and apply it to the GAZEBO simulator inference. This could test the ability of the model to interpret similar type BEV-images and yield adequate action sequence outputs.

As stated before, diffusion policy appears to learn the essence of the navigation behaviour and corresponding dynamics to a significant extent for diffusion policy either without or with BEV-image for the 2D-simulator, showcasing the model its promise to be used for the more complex environment of the GAZEBO environment as well. Since, BEV-images are not required for achieving significant success rates in the 2D-simulator, this is also expected for performing the less complex navigation task in the more complex GAZEBO environment. It should be noted that since the success rates we found were limited (0.53), merely limited success rates are expected to be found for the simple navigation behaviour in the GAZEBO simulator as well. Additionally, since the generalization tests of the diffusion policy showcased significant, albeit limited, generalization capabilities, it might be possible that testing the task of ASV navigation in a slightly different environment (out of training distribution) might still yield adequate success rates in the GAZEBO simulator environment.

In terms of reproducibility, this study is expected to be able to be reproduced with relatively few complications for the 2D simulator. The setup and conduction of the experiments was straightforward and simple and should not limit any future studies towards the performance of either the ACT or diffusion policy model. However, since this study leveraged a company specific GAZEBO simulator with associated NDA restrictions for sharing its specifics, it will not be possible to re-evaluate these GAZEBO experiments for outsiders of the company of Roboat.

8.4. Recommendations

In this section we will address possibilities for addressing the current limitations of this research. Furthermore, we will elaborate on additional recommendations that were not implemented due to time constraints of the thesis.

Simulator and demonstration loop

Since the model performances we establised were relatively poor for the GAZEBO simulator, but relatively high for the 2D-grid version, an interesting opportunity lies in incrementally building up the 2D-grid simulator complexity until the realism of the GAZEBO simulator. This way it might be possible to identify at which level of complexity the models can not capture and hence perform the navigation task correctly anymore. As a first step for this, the 2D-grid simulator can be expanded by applying the same state space equation as the GAZEBO simulator with regard to vessel dynamics instead of the simple dynamics currently applied. This might yield interesting insights, as to whether the GAZEBO simulator vessel dynamics might have been too complex to capture for the model and might have been the reason the model fails in achieving a success rate higher than 0.0. Additionally, the 2D simulator could be expanded with an A* planner to plan its trajectory. By applying this aspect, obstacle avoidance could be tested as well in a rather simple setup by adding static obstacles in the 2D-grid environment. As for the incorporating the more complex dynamics, these experiments might give insights as to whether the obstacle avoidance or more complex path planner applied in the GAZEBO simulator might have been the bottleneck for the model performance due to complexity of the task.

Furthermore, as diffusion policy could not be tested for the GAZEBO simulator with the current hardware setup, in a future work study one could try to gather more powerful hardware in order to also test diffusion policy in terms of navigation in the GAZEBO simulator.

Logging of data sources

In terms of data logging, there is a multitude of design choices one might investigate in order to try and make the models achieve high success rates for the GAZEBO simulator still. Firstly, one could try to tune the sampling frequency of the data loggers to a higher or lower value. When increasing this frequency, smaller ASV action values will be logged, since the pose difference between logging moments will be smaller. On the other hand, decreasing this frequency will result in logging higher action values given the same rationale. Investigating the model performance for different sampling rates might be interesting, since the sampling rate has a direct influence on the temporal character of the logged data. Regarding one demonstration, increasing the sampling frequency will result in a higher amount of logged data points, which might allow the model to leverage more temporal information with respect to the task to be learned. However, reducing the sampling frequency reduces the number of logged poses per demonstration. This results in a less detailed logged pose trajectory. It might be possible that the less detailed pose transition information ensures the task is more easily interpretable for the model. However, it should be noted that increasing the sampling frequency requires adequate adaptation of the approximatetymesynchronizer slop value, so that corresponding data of each action is still logged in tandem.

Additionally, one could investigate the effect on the model performances when learning the actions not relative to the previously logged pose as was done in this research, but by taking the starting point of each demonstration trial as the origin point (0,0,0) with respect to the demonstration trajectory and log each new ASV pose with respect to this coordinate frame as the action of that timestep. This way perhaps it might be more easy for the model to detect a more temporal connection between the logged action poses, since

these action poses would increase until the relative goal position approaches 0. Perhaps this temporal relation might help the model to figure out what is the desirable navigation behaviour to be learned.

As has been stated before, the leveraged camera data used during training was of relatively poor quality. It might be possible that a more photorealistic camera sensor data might help the model extract more information from the camera data which might help the model significantly in interpreting its environment and hence yield adequate action values.

The most rigorous option in attempting to allow the models to capture correct behaviour correctly for the GAZEBO simulator would be to simply add more (types of) data sources to the model. One prominent data source that might help is the inclusion of LIDAR point cloud data. Before starting this research it was already intended to use this data source for training both ACT and diffusion policy. However, due to time constraints this was not brought into practice. Ze et al. already showed that the inclusion of LIDAR point cloud data might improve generalization capability of diffusion policy considerably [22]. Hence, it might be interesting to see what the effects are in terms of performance when adding this data source for diffusion policy, but also for ACT.

Lastly, in order to allow the model to model to capture the correct navigation behaviour more accurately one could apply feature augmentation to the current datasets to try and obtain more generalized behaviour from either models. Feature augmentation is a standard method in AI-model training that allows a model to have less tendency to get stuck in local minima during training due to the higher constrained nature of the datasources that represent the behaviour to be learned. It might be possible that applying feature augmenation yields an increase in success rates found for the performed experiments.

Model training

In future work studies a number of non investigated hyper parameters could be varied and perhaps finetuned in a more elaborate hyper parameter study. For ACT one could still investigate the influence of varying: batch size, learning rate and number of hidden layers. For diffusion policy one could still investigate the influence of varying: batch size, learning rate schedule parameters, noise schedule parameters, the action horizon and model architecture parameters.

Inference components

In order to make the design of the GAZEBO inference components more flexible in its application, temporal ensembling of overlapping action chunks can be implemented to make the inference component more robust and flexible for its application.

9. Conclusion

This research aimed to find out to what extent ACT and diffusion policy trained algorithms allow for ASV navigation in complex marine environments. Based on the results it was found that ACT might show poor performance for a task as complex as this, however multiple limitations of the applied simulator and logged data sources might have been the cause for finding these poor results. For diffusion policy we found results that show promise with respect to this application, however due to hardware constraints for performing experiments, no solid statements can be made regarding the actual task of ASV navigation. Future work should focus on trying to perform similar experiments in a simulator environment containing more photorealistic camera images for training with less complex dynamics as it is possible these were the main reasons for failure. As for diffusion policy, in future studies diffusion policy could be evaluated in more detail when hardware resources in terms of GPU and RAM are available.

10. References

[1] Yubing Wu, Tao Wang, and Shuo Liu. "A Review of Path Planning Methods for Marine Autonomous Surface Vehicles". In: *Journal of Marine Science and Engineering* 12.5 (2024). Academic Editor: Sergei Chernyi, Received: 24 April 2024, Revised: 14 May 2024, Accepted: 15 May 2024, Published: 16 May 2024, p. 833. DOI: 10.3390/jmse12050833. URL: https://doi.org/10.3390/jmse12050833.

[2] Xinyu Zhang et al. "Collision-avoidance navigation systems for Maritime Autonomous Surface Ships: A state of the art survey". In: *Ocean Engineering* 235 (2021), p. 109380. DOI: 10.1016/j.oceaneng.2021.109380. URL: https://doi.org/10.1016/j.oceaneng.2021.109380.

[3] Arman Asgharpoor Golroudbari and Mohammad Hossein Sabour. *Recent Advancements in Deep Learning Applications and Methods for Autonomous Navigation: A Comprehensive Review*. 2023. arXiv: 2302.11089 [cs.RO]. URL: https://arxiv.org/abs/2302.11089.

[4] Jonghwi Kim et al. "Navigable Area Detection and Perception-Guided Model Predictive Control for Autonomous Navigation in Narrow Waterways". In: *IEEE Robotics and Automation Letters* 8.9 (Sept. 2023), pp. 5456–5463. DOI: 10.1109/LRA.2023.3273512.

[5] Maryam Zare et al. *A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges*. 2023. arXiv: 2309.02473 [cs.LG]. URL: https://arxiv.org/abs/2309.02473.

[6] Jiang Hua et al. "Learning for a Robot: Deep Reinforcement Learning, Imitation Learning, Transfer Learning". In: *Sensors* 21.4 (2021), p. 1278. DOI: 10.3390/s21041278.

[7] Tony Z. Zhao et al. *Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware*. 2023. arXiv: 2304.13705 [cs.RO]. URL: https://arxiv.org/abs/2304.13705.

[8] J Hyeon Park et al. "Hierarchical Action Chunking Transformer: Learning Temporal Multimodality from Demonstrations with Fast Imitation Behavior". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Abu Dhabi, United Arab Emirates: IEEE, Oct. 2024, pp. 12648–12654.

[9] Yanjie Ze et al. *3D Diffusion Policy: Generalizable Visuomotor Policy Learning via Simple 3D Representations*. 2024. arXiv: 2403.03954 [cs.RO]. URL: https://arxiv.org/abs/2403.03954.

[10] Yulai Zhao et al. *Adding Conditional Control to Diffusion Models with Reinforcement Learning*. 2024. arXiv: 2406.12120 [cs.LG]. URL: https://arxiv.org/abs/2406.12120.

[11] Brian Yang et al. *Diffusion-ES: Gradient-free Planning with Diffusion for Autonomous Driving and Zero-Shot Instruction Following*. 2024. arXiv: 2402.06559 [cs.LG]. URL: https://arxiv.org/abs/2402.06559.

[12] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. *Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation*. 2024. arXiv: 2401.02117 [cs.RO]. URL: https://arxiv.org/abs/2401.02117.

[13] Saurabh Arora and Prashant Doshi. "A survey of inverse reinforcement learning: Challenges, methods and progress". In: *Artificial Intelligence* 294 (2021). Available online 24 March 2021, p. 103457. DOI: 10.1016/j.artint.2021.103457. URL: https://doi.org/10.1016/j.artint.2021.103457.

[14] Tomoya Yokoyama et al. "Intervention Force-based Imitation Learning for Autonomous Navigation in Dynamic Environments". In: *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. Auckland, New Zealand: IEEE, Dec. 2020. URL: https://ieeexplore.ieee.org/document/XXXXXXX.

[15] Joonwoo Ahn, Minsoo Kim, and Jaeheung Park. *Vision-based Autonomous Driving for Unstructured Environments Using Imitation Learning*. 2022. arXiv: 2202.10002 [cs.RO]. URL: https://arxiv.org/abs/2202.10002.

[16] Tung Phan-Minh et al. *Driving in Real Life with Inverse Reinforcement Learning*. 2022. arXiv: 2206.03004 [cs.RO]. URL: https://arxiv.org/abs/2206.03004.

[17] Zhiyu Huang, Jingda Wu, and Chen Lv. *Driving Behavior Modeling using Naturalistic Human Driving Data with Inverse Reinforcement Learning*. 2021. arXiv: 2010.03118 [cs.RO]. URL: https://arxiv.org/abs/2010.03118.

[18] Zhiyu Huang et al. "Conditional Predictive Behavior Planning with Inverse Reinforcement Learning for Human-like Autonomous Driving". In: *arXiv preprint arXiv:2212.08787* (2022). Submitted on 17 Dec 2022, last revised 7 Mar 2023. URL: https://doi.org/10.48550/arXiv.2212.08787.

[19] David Sierra González et al. *Modeling Driver Behavior from Demonstrations in Dynamic Environments Using Spatiotemporal Lattices*. Brisbane, QLD, Australia, 2018. DOI: 10.1109/ICRA.2018.8460208. URL: https://ieeexplore.ieee.org/document/8460208.

[20] Gustavo Claudio Karl Couto and Eric Aislan Antonelo. "Hierarchical Generative Adversarial Imitation Learning With Mid-Level Input Generation for Autonomous Driving on Urban Environments". In: *IEEE Transactions on Intelligent Vehicles* (2024), pp. 1–14. ISSN: 2379-8858. DOI: 10.1109/tiv.2024.3436587. URL: http://dx.doi.org/10.1109/TIV.2024.3436587.

[21] Pin Wang et al. *Decision Making for Autonomous Driving via Augmented Adversarial Inverse Reinforcement Learning*. 2021. arXiv: 1911.08044 [cs.AI]. URL: https://arxiv.org/abs/1911.08044.

[22] Yanjie Ze et al. *Generalizable Humanoid Manipulation with Improved 3D Diffusion Policies*. 2024. arXiv: 2410.10803 [cs.RO]. URL: https://arxiv.org/abs/2410.10803.

[23] Piyabhum Chaysri et al. "Unmanned Surface Vehicle Navigation Through Generative Adversarial Imitation Learning". In: *Ocean Engineering* 282 (2023), p. 114989. DOI: 10.1016/j.oceaneng.2023.114989. URL: https://doi.org/10.1016/j.oceaneng.2023.114989.

[24] Takefumi Higaki and Hirotada Hashimoto. "Human-like Route Planning for Automatic Collision Avoidance Using Generative Adversarial Imitation Learning". In: *Applied Ocean Research* 138 (2023). Available online 7 June 2023, p. 103620. DOI: 10.1016/j.apor.2023.103620. URL: https://www.sciencedirect.com/science/article/pii/S014111872300161X.

[25] Andrea Asperti and Matteo Trentin. *Balancing reconstruction error and Kullback-Leibler divergence in Variational Autoencoders*. 2020. arXiv: 2002.07514 [cs.NE]. URL: https://arxiv.org/abs/2002.07514.

[26] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG]. URL: https://arxiv.org/abs/2006.11239.

[27] Cheng Chi et al. *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion*. 2024. arXiv: 2303.04137 [cs.RO]. URL: https://arxiv.org/abs/2303.04137.

[28] Toshihide Ubukata, Jialong Li, and Kenji Tei. *Diffusion Model for Planning: A Systematic Literature Review*. 2024. arXiv: 2408.10266 [cs.LG]. URL: https://arxiv.org/abs/2408.10266.

[29] Florinel-Alin Croitoru et al. "Diffusion Models in Vision: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.9 (Sept. 2023), pp. 10850–10869. ISSN: 1939-3539. DOI: 10.1109/tpami.2023.3261988. URL: http://dx.doi.org/10.1109/TPAMI.2023.3261988.

[30] Sung-Wook Lee and Yen-Ling Kuo. *Diff-DAgger: Uncertainty Estimation with Diffusion Policy for Robotic Manipulation*. 2024. arXiv: 2410.14868 [cs.RO]. URL: https://arxiv.org/abs/2410.14868.

[31] Chia-Hsun Chang et al. “COLREG and MASS: Analytical review to identify research trends and gaps in the Development of Autonomous Collision Avoidance”. In: *Ocean Engineering* 302 (June 2024), p. 117652. DOI: 10.1016/j.oceaneng.2024.117652. URL: https://doi.org/10.1016/j.oceaneng.2024.117652.

[32] Abraham George and Amir Barati Farimani. *One ACT Play: Single Demonstration Behavior Cloning with Action Chunking Transformers*. 2023. arXiv: 2309.10175 [cs.RO]. URL: https://arxiv.org/abs/2309.10175.

[33] Thanpimon Buamanee et al. *Bi-ACT: Bilateral Control-Based Imitation Learning via Action Chunking with Transformer*. 2024. arXiv: 2401.17698 [cs.RO]. URL: https://arxiv.org/abs/2401.17698.

[34] Xinyin Ma, Gongfan Fang, and Xinchao Wang. *DeepCache: Accelerating Diffusion Models for Free*. 2023. arXiv: 2312.00858 [cs.CV]. URL: https://arxiv.org/abs/2312.00858.

[35] Tony Z. Zhao and Cheng Chi. *ACT: Action Chunking with Transformers*. https://github.com/tonyzhaozh/act. Accessed: 2025-05-01. 2024.

[36] Cheng Chi et al. *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion*. https://gitlab.com/real-stanford/diffusion_policy. Accessed: 2025-05-01. 2024.

11. Appendix

Listing 1: ACT HDF5 File Structure

```
action: <HDF5 dataset "action": shape (200, 4), type "<f4">
observations: <HDF5 group "/observations" (6 members)>
    goal_pose: <HDF5 dataset "goal_pose": shape (200, 4), type "<f4">
    images: <HDF5 group "/observations/images" (1 members)>
        front: <HDF5 dataset "front": shape (200, 400, 480, 3), type "|u1">
    lucy_orientation: <HDF5 dataset "lucy_orientation": shape (200, 1), type "<f4">
    lucy_velocity: <HDF5 dataset "lucy_velocity": shape (200, 6), type "<f4">
    poses: <HDF5 dataset "poses": shape (200, 24), type "<f4">
    velocities: <HDF5 dataset "velocities": shape (200, 36), type "<f4">
```

Listing 2: Diffusion policy HDF5 File Structure (Reformatted with demo data)

```
data: <HDF5 group "/data" (2 members)>
    demo_0: <HDF5 group "/data/demo_0" (3 members)>
        actions: <HDF5 dataset "actions": shape (84, 4), type "<f4">
        obs: <HDF5 group "/data/demo_0/obs" (6 members)>
            front: <HDF5 dataset "front": shape (84, 240, 240, 3), type "|u1">
            goal_pose: <HDF5 dataset "goal_pose": shape (84, 4), type "<f4">
            lucy_orientation: <HDF5 dataset "lucy_orientation": shape (84, 1), type "<f4">
            lucy_velocity: <HDF5 dataset "lucy_velocity": shape (84, 6), type "<f4">
            poses: <HDF5 dataset "poses": shape (84, 24), type "<f4">
            velocities: <HDF5 dataset "velocities": shape (84, 36), type "<f4">
        states: <HDF5 dataset "states": shape (20, 3), type "<f4">
    demo_1: <HDF5 group "/data/demo_1" (3 members)>
        actions: <HDF5 dataset "actions": shape (20, 4), type "<f4">
        obs: <HDF5 group "/data/demo_1/obs" (6 members)>
            front: <HDF5 dataset "front": shape (20, 240, 240, 3), type "|u1">
            goal_pose: <HDF5 dataset "goal_pose": shape (20, 4), type "<f4">
            lucy_orientation: <HDF5 dataset "lucy_orientation": shape (20, 1), type "<f4">
            lucy_velocity: <HDF5 dataset "lucy_velocity": shape (20, 6), type "<f4">
            poses: <HDF5 dataset "poses": shape (20, 24), type "<f4">
            velocities: <HDF5 dataset "velocities": shape (20, 36), type "<f4">
        states: <HDF5 dataset "states": shape (20, 3), type "<f4">
```

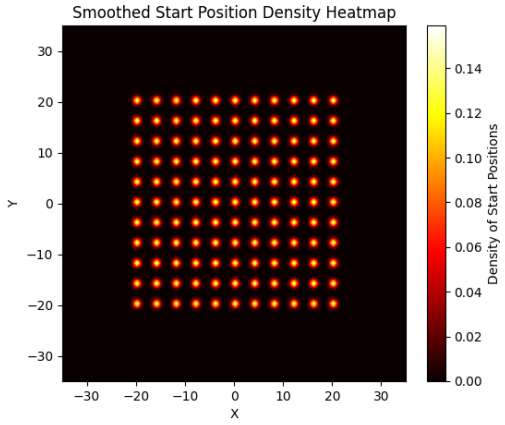


Figure 22: equidistant sampling across the grid for trajectory density plots

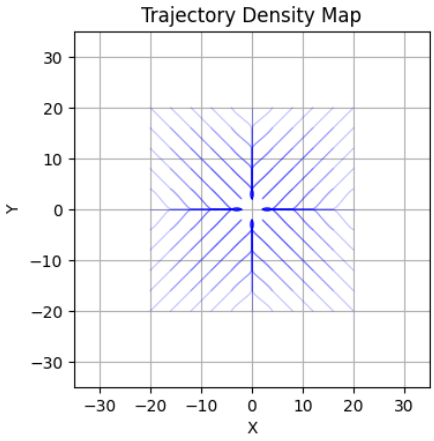


Figure 23: trajectory density plot for 2D-LQR dynamics

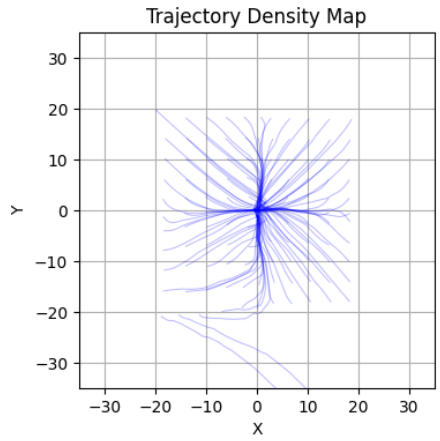


Figure 24: trajectory density plot for ACT inference

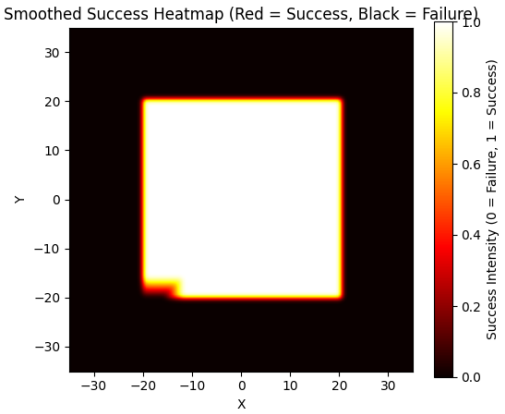


Figure 25: success rate heatmap for ACT trajectory density plot

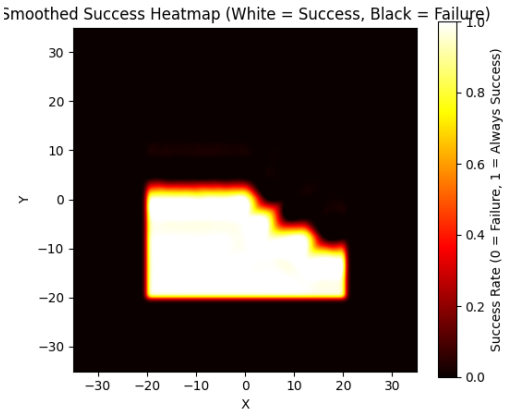


Figure 27: success rate heatmap for diffusion policy trajectory density plot

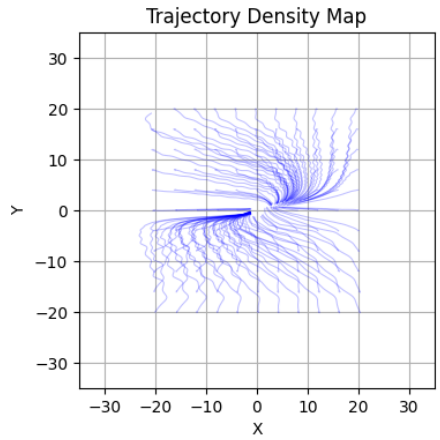


Figure 26: trajectory density plot for diffusion policy inference