Journal of Open Aviation Science (2025), Vol.**3**
doi:10.59490/joas.2025.7875

JOAS

ARTICLE

Sampling-Based Aircraft Path Planning with Soft Actor-Critic

Dieudonne Groot, * Joost Ellerbroek, and Jacco Hoekstra

Delft University of Technology, Kluyverweg 1, 2629 HS, Delft, Zuid-Holland, The Netherlands
*Corresponding author: d.j.groot@tudelft.nl

(Received: 21 Oct 2024; Revised: 19 Feb 2025; Accepted: 31 Mar 2025; Published: 24 Apr 2025)
(Editor: Tatiana Polishchuk; Reviewers: Sameer Alam, Arsyi Aziz, Filipo Perotto)

Abstract

This paper investigates the usage of reinforcement learning for a global path planning task in terminal airspace, specifically for training a policy that can generate paths from any given position in the airspace to the runway. To do this, the Soft Actor-Critic (SAC) algorithm is trained on a simplified version of the Dutch airspace and compared to the solutions generated by the Dijkstra algorithm for varying discretization resolutions. SAC, which uses a Gaussian distribution for the action policy, has previously been shown to be successful in other global planning tasks in continuous environments. However, evaluating the policy by following the mean of the learned distribution, which is the standard evaluation method, may yield suboptimal performance when dealing with complex cost functions that deviate from a normal distribution. To address this, the paper proposes and evaluates a sampling-based strategy, which generates an ensemble of paths by sampling from the learned policy distribution. These three methods: mean-based SAC, Dijkstra and sampling-based SAC, are then tested on a bi-criterion cost function which includes both fuel and noise emissions in varying ratios. It was found that Dijkstra outperforms mean-based SAC for all cost ratios at the best discretization resolution, regardless of the neural network architectures used. However, sampling-based SAC results in consistently lower costs than both Dijkstra and mean-based SAC, particularly for the more complex cost functions that have a higher focus on noise mitigation. These findings highlight some limitations in mean-based evaluation for distribution models and indicate potential performance benefits that can be obtained with better-tailored evaluation strategies.

Keywords: Reinforcement Learning; Soft Actor-Critic; BlueSky Simulator; Path Planning; Air Traffic Management

Abbreviations: RL: Reinforcement Learning, SAC: Soft Actor-Critic, ATM: Air Traffic Management

1. Introduction

For the future of air traffic management (ATM) and air traffic control, the European ATM Master Plan, developed by the SESAR Joint Undertaking, envisions a future characterized by advanced automation, with certain aspects of air traffic control potentially operating entirely autonomously without human intervention [1]. An advantage of these higher levels of automation is the potential broad implementation of free route airspace, where aircraft can fly individually optimized paths, tailored to their own operational requirements, such as origin and destination, or aircraft performance and airline preferences. Compared with the current airspace structure, which relies on predetermined routes such as the Standard Terminal Arrival Routes (STARs) and fixed waypoints, free routing can lead to a reduction in emissions and operational costs through utilization of the entire continuous airspace, rather than being constrained by discretized waypoints [2]. Achieving these higher levels of automation, as outlined in the European ATM Master Plan, could rely on the potential of ma-

chine learning (ML) and artificial intelligence (AI) for various ATM tasks. This in turn has led to an increase in studies that investigate applications of ML/AI in ATM [3, 4].

Reinforcement Learning (RL), a branch of ML that focuses on problems which require sequential decision making, has been shown to be successful in various ATM related challenges [5]. For example, RL algorithms have been applied to the tasks of conflict detection and resolution, or maintaining safe separation between aircraft, demonstrating high levels of performance compared to analytical methods, especially at higher traffic densities [6]. For the conflict resolution task most RL applications investigated have focused solely on the safety aspect of the task, as the complexity of multi-criterion cost functions can lead to policies that inadvertently prioritize efficiency over safety, which is unacceptable in ATM [4]. Because of this, RL for path planning in aviation with a focus on other cost factors unrelated to safety has not been investigated extensively. However, to obtain the most benefits of RL, it is also necessary to study the application of RL to single aircraft path planning over longer path horizons, with more complex cost functions. The findings in both areas can then eventually be combined, for example, through the use of hierarchical frameworks such as hierarchical RL [7], ensuring both safe and cost-effective paths.

Additionally, RL for single-agent path planning has demonstrated success in various domains [8] including that of minimally invasive surgery, surpassing traditional path planning methods for continuous environments, such as Rapidly-exploring Random Trees * (RRT*) [9]. Although these applications may appear unrelated, the task of path planning in a known terminal airspace shares various similarities with path planning on a brain map. In both scenarios, the starting point is variable, either the entry point in the airspace or the initial incision location in the skull, while the destination is fixed, such as an airport or a tumor. This requires the planning of multiple paths within a known environment, essentially memorizing the entire problem space. Furthermore, both problems involve regions with varying associated costs, leading to a trade-off between path efficiency and avoidance of sensitive areas, without direct obstacle avoidance. These similarities justify further investigation of RL as a method for path planning in the terminal airspace.

The set of RL algorithms that showed the best performance on the minimally invasive surgery tasks are distribution models, specifically Soft Actor-Critic (SAC) [10] and Proximal Policy Optimization (PPO) [11], which are RL algorithms that learn a stochastic policy by modeling actions as Gaussian distributions. During evaluation, the mean of the learned distribution is then used to determine the actions used for planning the path. Because the underlying cost distributions are complex and may not be accurately captured in a normal distribution, it is hypothesized this mean-based evaluation may not always yield the optimal path.

This paper therefore compares the use of the mean of the learned action distribution to using a sampling strategy from the learned distribution to generate an ensemble of paths. These paths are then compared with the paths generated by the mean of the learned distribution to identify if potential performance improvements can be obtained. To do this, first, the impact of the neural network architecture used for the SAC RL algorithm on a path-planning task within the Dutch National Airspace, is evaluated. This is done to ensure that the base model used for evaluation sufficiently captures the environment. These models are then compared with benchmark paths generated with Dijkstra for various discretization resolutions, in order to verify that the initial paths generated by mean-sampled SAC resemble the optimal paths in discrete space. Finally, the ensemble of the paths generated through sampling from the learned distribution will be evaluated and compared with the results from both mean-sampled SAC and Dijkstra.

The remainder of this paper is structured as follows. Section 2 will define the path planning problem used for this study, which is a multi-criterion path planning problem that focuses on a combination of noise and fuel mitigation. Section 3 will describe the methodology, which consists of the associated

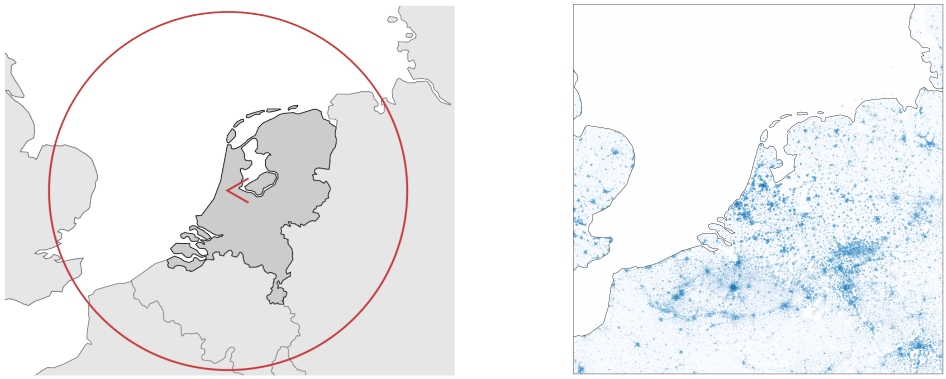
cost functions and simulator used to determine the fitness of the generated paths, as well as the specifics and implementations of the SAC and Dijkstra algorithms. Section 4 outlines the conducted experiments and their results. Section 5 will discuss the observed results and finally Section 6 will conclude the paper and highlight potential future works.

2. Problem description and formulation

To compare the performance of the two path planning methods, a case study is conducted for the Dutch national airspace. This section will describe the environment and formalize it into a path planning problem by providing the problem input, possible actions, constraints and finally the cost function used for determining the fitness of the different paths.

2.1 Environment: Dutch airspace

The environment used for this study is a simplified representation of the Dutch airspace. To simplify the environment, a circular border, with a radius of 277 km (150 NM), centered at Schiphol (52.31° lat, 4.78° lon) is used instead of the actual border of the Dutch airspace. This airspace is also shown in Figure 1. In this figure the red lines in the center of the environment represent the approach constraints, further explained in Section 2.2.



(a) (b)

Figure 1. (a) Simplified representation of the Dutch airspace used for this research. (b) Population data used for the research as obtained from Eurostat [12]

Within this airspace, the goal of the path planning algorithm is to generate paths from the border of the environment to the beginning of runway 27, located at the end of the red approach lines in the center of Figure 1, while optimizing the bi-objective cost function of fuel and noise emissions. This requires a trade-off between path efficiency and avoiding highly populous areas. Figure 1b shows the population data used for calculating the noise emissions, which is obtained from Eurostat [12].

The beginning of runway 27 was chosen as the terminal state of the environment as it requires aircraft to approach over land, increasing the relevance of the noise emission cost.

The path planning problem is solved as a 2D problem, as the vertical component is isolated and follows the optimized vertical profile for continuous descent operations.

2.2 Constraints

The constraints used in the environment are illustrated by the red lines shown in Figure 1. These constraints are the circular border with a radius of 277 km centered at 52.31° lat, 4.78° lon, to ensure

that agents do not exit the airspace, and two lines of 37 km (20 NM) originating at the end of runway 27 at $+-30.0°$, which ensures that aircraft approach the runway from an appropriate angle. Any of these constraints are considered violated when the shortest path between 2 consecutive waypoints intersects these lines. In this study waypoints are defined by any (lat,lon) coordinate and serve as the next, intermediate, target location.

2.3 Cost function

For the cost function, a bi-objective problem of minimizing fuel cost and noise emissions is formulated. The general function of the per-segment cost function is given in eq. 1, here $N_{s,s+1}$ and $F_{s,s+1}$ are the normalized noise and fuel cost associated with a segment flown described by the waypoints 's' and '$s+1$' respectively and 'λ' is the weight attributed to the individual cost elements. The cost components are normalized to ensure that, for any random path, the contribution of the individual elements to the total cost is equal, given that $\lambda = 0.5$. Note that this is not necessarily true for optimized paths, as it is easier to mitigate noise costs than fuel costs by avoiding highly populous areas.

$$Cost(s_i, s_{i+1}) = \lambda N_{s_i,s_{i+1}} + (1-\lambda)F_{s_i,s_{i+1}}; \qquad 0 \leq \lambda \leq 1 \tag{1}$$

The total cost of any path can then be calculated by summing all segments describing the path.

2.3.1 Noise cost

To determine the noise cost, $N_{s,s+1}$, the noise emitted by the aircraft, $Noise_{AC}$, is calculated by combining the noise-power-distance database of Eurocontrol [13] with the thrust of the aircraft as determined by the Open Aircraft Performance (and Emission) Model, OpenAP [14]. This emitted noise is then convoluted over the population density map from Eurostat (Figure 1b) using eq. 2, which follows the inverse square law for noise dissipation, at a sampling rate of 1hz to obtain the noise pollution during 1 second, n_t. $P_{i,j}$ and $D_{i,j}$ refer to the population in, and distance to, cell 'i,j' in the population data respectively. This is also illustrated in Figure 2.

$$n_t = \sum_{j=0}^{j=j_{max}} \sum_{i=0}^{i=i_{max}} P_{i,j} \cdot \frac{Noise_{AC}}{D_{i,j}^2} \tag{2}$$

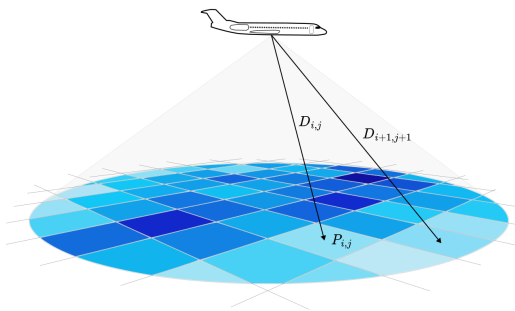


Figure 2. Visualization of the noise projection over the population map following the calculations done in eq. 2.

These noise costs, calculated at 1-second intervals, are then summed to obtain the total noise cost of a single segment, $n_{s_i,s_{i+1}}$.

Finally, the noise cost for a single segment is normalized by dividing it by the expected noise cost for flying over a random segment of length $|s_i, s_{i+1}|$, as is illustrated in eq. 3, resulting in the normalized noise cost $N_{s_i,s_{i+1}}$. This expected noise is calculated by assuming that the population is homogeneously divided over the environment, it therefore represents the average cost for any segment of length $|s_i, s_{i+1}|$, independent of the values of s_i and s_{i+1}.

$$N_{s_i,s_{i+1}} = \frac{n_{s_i,s_{i+1}}}{\mathbb{E}[n_{s_i,s_{i+1}}]} \tag{3}$$

It is acknowledged that this method for calculating noise cost does not consider the hearing and annoyance thresholds of humans, or looks at more sophisticated noise models. However, for the purpose of this research, the implementation is considered sufficient and easier to reproduce. The used methods however, are independent from the used noise cost metric, and can also be used for different noise calculations.

2.3.2 Fuel cost

For the fuel cost, $F_{s_i,s_{i+1}}$, OpenAP is used to obtain an estimate of the fuel flow, ff, based on the used aircraft model, engine type, altitude and airspeed. To get the fuel cost per flown segment of the aircraft, this fuel flow is summed over time, as is shown in eq. 4, where Δt is the simulation timestep. Starting at $t = t_{s_i}$ and ending at $t = t_{s_{i+1}}$, which are the times at which the aircraft starts from the current and arrives at the next waypoint respectively.

$$F_{s_i,s_{i+1}} = \sum_{t_{s_i}}^{t_{s_{i+1}}} ff_t \Delta t \tag{4}$$

3. Methodology

3.1 Simulator: BlueSky Air Traffic Simulator

For conducting the experiments and evaluating the paths, the BlueSky open-source Air Traffic Simulator is used [15]. This simulator allows the usage of OpenAP as the performance model of the aircraft specified, making it easy to evaluate the cost of the flown paths and ensuring that the dynamics of the aircraft are not ignored in the final paths. The code and plugins used for this research can be found in the 4TU repository [16]. The most recent version of BlueSky can be found on GitHub [17].

3.2 Method: Soft Actor-Critic

SAC is an RL algorithm developed by Haarnoja et al. [10]. The main premise of the method is to reduce the brittleness of performance with respect to hyperparameters and explicitly incorporate exploration by including a maximum entropy term into the reward function. The SAC algorithm uses three neural networks, two of which are only used during the training phase of the method. These networks are called the critic, value and policy networks.

To incorporate the entropy component of the method, the policy network learns an action distribution, outputting both a mean and a standard deviation for a given input state. During the training phase, actions are sampled from this distribution, which aids in the exploration of unseen states when compared with deterministic RL methods. During the evaluation phase, normally the means are used directly instead, as it is assumed that for a converged model, these are the optimal actions. However, sampling can still be used to generate an ensemble of solutions, which is useful when new paths should be created due to detected conflicts in a multi-agent path planning setting. The

impact of sampling is evaluated by analysing the performance differences over ensembles of paths in Section 4.7.

3.2.1 Input representation

For the input representation, the coordinates are translated to the Cartesian coordinate system with $x, y = 0$ corresponding to the end of runway 27 at 52.31° lat, 4.78° lon, such that the terminal state of the agent approaches 0,0 in Cartesian. This is similar to the state representation often used in grid-world examples [18]. To ensure that the input values are between -1 and +1, which aids in stability during the initial phase of the training, the input states are normalized by dividing them by the maximum distance from the environment centre. This state representation is shown graphically in Fig. 3, as defined by the axes values. The state representation used in this study is deliberately kept simple because this design choice allows the next state to be determined simply from the current state and the selected action, enabling recursive path generation without requiring intermediate simulation steps. While more complex representations may potentially offer improved performance, this simple state representation is considered sufficient for the purpose of this study.

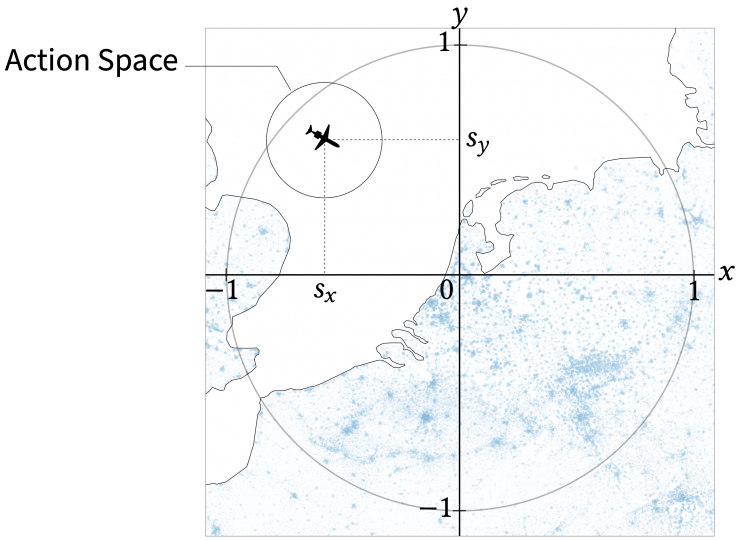


Figure 3. Graphical illustration of the agent's state and action space. The state of the aircraft is represented by s_x and s_y.

3.2.2 Action representation

The action space of the model is defined as $[\Delta x, \Delta y] < 50$km, as is shown in Fig. 3. An upper limit is given to this action space to limit the size of the action space, as it was found that a larger action space results in slower convergence of the model during training. Additionally, because any path can be defined by an infinite set of waypoints, optimizing this value was not considered part of the scope of this research, as it is hypothesized that it will not influence the quality of the paths. For future research, it can however still be worthwhile to investigate the validity of this hypothesis and the influence on convergence rate.

3.2.3 Cost representation

For the cost representation, or reward function, the same costs as described in section 2.3 are used, they are however, multiplied with -1 to change the problem from a minimization to a maximization problem. Additionally, the constraints mentioned in section 2.2 are modelled as a cost of '-1', to

ensure that any optimal solution, if it exists, does not violate the constraints, similar to the big M method used in operations research. Finally, a successful approach is rewarded '+5' to stimulate early termination of the episodes during initial training. This alteration does not mathematically alter the optimal paths as this reward is given to every valid path generated by the model. Both a successful approach and constraint violations result in terminal conditions for the path planning method. The entire reward function is given in eq. 5.

$$f_{reward}(s_i, s_{i+1}) = -\lambda N_{s_i,s_{i+1}} - (1-\lambda)F_{s_i,s_{i+1}} + \begin{cases} -1 & \text{constraint violation} \\ +5 & \text{successful approach} \end{cases} \tag{5}$$

3.2.4 Training

To train the models, randomly generated scenarios are used, and simulated in the BlueSky simulator. In these scenarios, the initial states are randomly sampled uniformly from the entire state space. This is in contrast with the evaluation scenarios, where the initial states are located on the border of the environment. This strategy is employed to improve the exploration of the method over the entire state space. The model then interacts with the environment for a total of 15 actions or until a terminal condition is obtained, after which a new initial state is generated. This process is repeated for a total of 200.000 actions, after which the current model is saved and used for evaluation. The algorithm used for training is shown in algorithm 1.

An additional advantage of training in the simulator is that it ensures that the paths generated by the method can be flown by the dynamics of the aircraft model used. This means that for higher-fidelity simulators no performance constraints have to be specified for the problem, as is commonly done in aircraft related path planning problems [19, 20].

Algorithm 1 Training loop for the SAC algorithm

```
Ensure: buffer → empty
  model = SAC( hyperparameters )
  steps = 0
  max_steps = 200.000
  new_episode = True
  while steps < max_steps do
    if new_episode then
      s_i ← spawn_aircraft()
      new_episode = False
    end if
    a_i ← model(s_i)
    obtain s_{i+1}
    obtain r_i and terminal condition from f_reward(s_i, s_{i+1})
    add (s_i, a_i, r_i, s_{i+1}) to buffer
    update model
    steps += 1
    s_i = s_{i+1}
    if terminal condition then
      delete_aircraft()
      new_episode = True
    end if
  end while
```

3.2.5 Hyperparameters and network architecture

The hyperparameters used for the SAC algorithm are given in Table 1. These hyperparameters are the same as those used in the original paper by Haarnoja et al. with four exceptions; It was found that a higher learning rate (3e-3 instead of 3e-4) and higher discount factor (0.995 instead of 0.99) leads to

faster convergence of the training. Additionally, four different network architectures are tested, to analyse the impact of the number of neurons (256 & 1024) and layers (2 & 3) on the performance of the method (Section 4.4). This is done in an analogy to the discretization resolution used for discrete methods such as Dijkstra.

Table 1. Hyperparameters for the Soft Actor-Critic Algorithm

Parameter	Value
Optimizer	Adam
Learning rate	3e-3
Discount factor (γ)	0.995
Memory buffer size	10e6
Sample size	256
Smoothing coefficient (τ)	5e-3
Network update frequency	1
Number of layers	[2,3]
Neurons per layer	[256, 1024]
Nonlinearity	ReLU

3.3 Baseline method: Dijkstra

In this research, the Dijkstra algorithm will be used for generating optimal single-agent paths in discrete space. These paths function as a comparison for the paths generated by the SAC algorithm. For the algorithm, the Dijkstra implementation of the open-source Python package NetworkX is used. As Dijkstra requires the environment to be discrete, the following sections will first describe the discretization method applied to the problem. This is followed by an explanation of the post-processing used for enhancing the performance of the solutions in continuous space.

3.3.1 Problem discretization

To use the Dijkstra algorithm for path planning, the environment is required to be discretized. To discretize the environment, a square grid is used, where for each cell the total population is summed and assumed to be distributed homogeneously throughout the cell. From any cell in this square grid (excluding border cells) the Dijkstra algorithm can select all neighbouring cells (including diagonally connected cells) as well as cells reachable via an "L"-shaped move analogous to a knight's move in chess. These transitions are then used to define the edges of the graph to which the Dijkstra algorithm is applied.

The resolution used for the discretization of the problem is nontrivial. Lower resolutions might not sufficiently capture the entire solution space, leading to sub-optimal solutions, whereas higher resolutions might not capture the noise dissipation in continuous space and have more frequent heading changes, leading to deficiencies when evaluated in the continuous environment simulated in BlueSky. Therefore all discretization resolutions shown in Table 2 are used for initial evaluation. Section 4.5 will go over the experimental setup and results for these different resolutions.

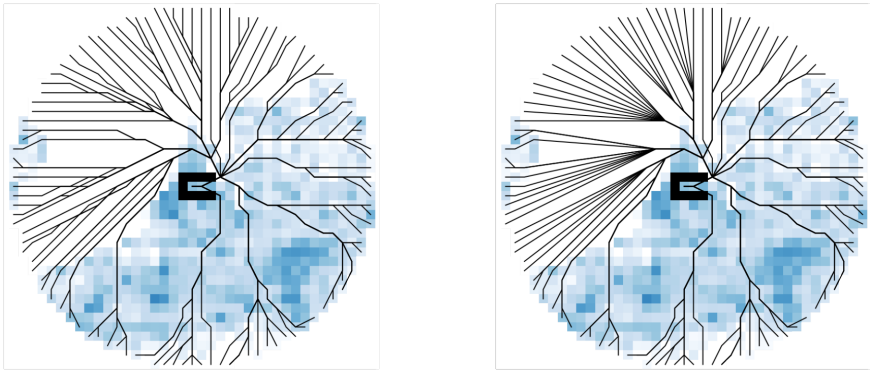
3.3.2 Solution post-processing

As a result of the discretization of the problem, the shortest path above zero population areas (such as large bodies of water) is limited by the action resolution available to the Dijkstra method. Therefore a check is made to see which nodes cross zero-population areas, and if that is the case, these nodes are removed from the final output. This results in direct routing above the sea and other zero-population areas. The difference between the solutions with and without this post-processing is shown in Figure

Table 2. Discretization resolutions evaluated for the Dijkstra method.

cell size (km)	grid-world size
13	40 x 40
10	52 x 52
8	65 x 65
5	104 x 104
4	130 x 130
2	260 x 260
1	520 x 520

4. A similar method for smoothing the paths can not be used for areas with a non-zero population count, as it is impossible to guarantee that this will improve the solution due to the bi-criterion nature of the cost function also relying on noise abatement.



(a) (b)

Figure 4. (a) Naive Dijkstra solutions for the problem from the edge states of the environment. (b) Dijkstra solutions for the discretized environment after post-processing

4. Simulation experiments and results

To determine the efficacy of the different methods, four different experiments are conducted within the BlueSky Air Traffic Simulator. All experiments are evaluated using the same experimental scenario and control variables, described in Section 4.1 and Section 4.2 respectively. The dependent variables used to determine the quality of the paths are explained in section 4.3.

The experiments conducted are:

- The impact of neural network architecture for SAC. (Section 4.4)
- The impact of discretization resolution for Dijkstra. (Section 4.5
- Performance differences between SAC and Dijkstra for various cost weight ratios. (Section 4.6)
- Sampling-based path planning with the learned action distribution. (Section 4.7)

4.1 Experimental scenario

For the experimental scenario used for evaluating the paths, 36 initial states are generated on the border of the circular environment described in Section 2.1, with an equal spacing of 10 degrees to

ensure complete coverage of the entire heading range. The paths generated from these initial states to the terminal state (beginning of runway 27) are then simulated in BlueSky and evaluated against the total noise and fuel cost specified in Section 2.3.

4.2 Control variables

In all experiments, the dynamics, noise and fuel calculations are based on an A320 with an assumed constant mass of 66,000 kg. The groundspeed of the aircraft is held constant at 463 km/h (250 kts). Finally, for the noise and fuel calculation, a fixed altitude of 4.57 km (15,000 ft) is used, as for the vertical component it is assumed that aircraft follow their optimal vertical profile for continuous descent operations after the horizontal path is planned. This does influence the optimality of the solutions closer to the airport where the altitude is lower, as in practice the noise cost there should be higher than close to the border of the environment. However, closer to the airport the freedom for path changes is lower due to the approach angle of the runway, it is therefore assumed that this influence is not big enough for the purpose of this study, as all compared methods are subjected to these same conditions. Future studies should however investigate and evaluate its impact.

4.3 Dependent variables

4.3.1 Fuel, noise and total cost

The primary performance indicators used for evaluating the quality of the paths are the different cost components. These values will be presented as normalized, unit-less values, which allows comparing the values on the same scale. The total cost is the main performance indicator, as it is what is used to calculate the cost of the paths, however, the isolated noise and fuel costs can still provide additional insights into the generated paths, as it is possible to compensate for noise emissions with a shorter path, leading to different potential policies for the SAC based method.

4.3.2 Computation time

Although not the primary focus of this study, the computation time required for generating a single path is also considered an indicator of the feasibility of the methods. Especially for the sampling-based method, or multi-agent scenarios with re-planning, which both require multiple paths to be generated, computational requirements are an important factor. For SAC the computation time is based on a fully trained model, as the environment is considered static, and once learned can be used for any initial state in the environment. Hence training time is not included in this metric.

4.3.3 Number of turns

The last indicator used for evaluating the methods is the number of turns. The number of turns is an indicator of path complexity, and is observed as an additional objective measure for comparing the different paths. The main purpose of this indicator is to compare the different paths generated by the different network architectures for SAC in Section 3.2.5.

The number of turns is defined as the number of heading changes required to fly the path, regardless of the size of the heading change.

4.4 Experiment 1: Impact of neural network architecture (SAC)

For the SAC algorithm, it is hypothesized that the path quality is related to the network architecture used for the model. This is because more complex network architectures allow for more fine-tuned decisions, similar to a higher resolution for the Dijkstra method. To test this hypothesis, the SAC algorithm is trained with four different network architectures underlying the model structure. For this experiment λ is set at 0.5, ensuring equal weight is given to both the fuel and noise cost. The

network architectures evaluated in this experiment are given in Table 3 and are all trained according to the training procedure outlined in Section 3.2.4.

Table 3. Network architectures tested for the SAC algorithm. The same architecture is shared for all networks in the model.

number of layers	neurons per layer
2	256
2	1024
3	256
3	1024

The path cost during training is given in Figure 5 and shows the evolution of performance for the four different models for the first 5000 episodes. It can be observed that all network architectures have a steep increase in performance between episodes 1000 and 2000, however, no clear distinction can be observed for the different network architectures during the early stages of training.

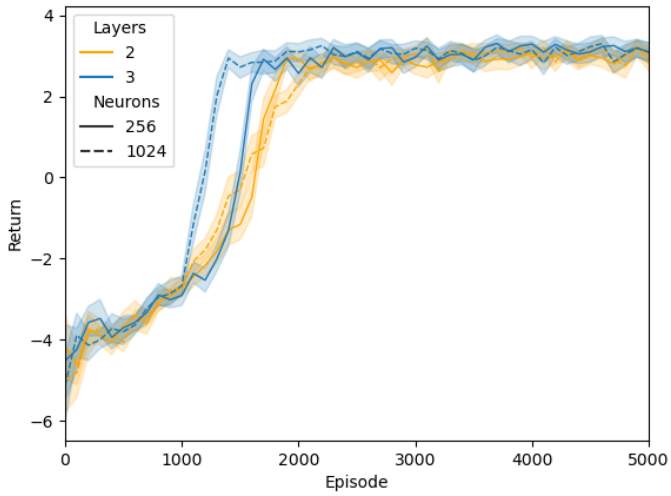


Figure 5. Evolution of the episodic return for the different network architectures for the first 5000 episodes. It can be observed that utilizing three layers is beneficial for the early training, however, the different networks quickly converge to similar values.

To better distinguish between the performance of the models after training, Table 4 shows the performance of the different network architectures after 200,000 actions. From this table, it can be observed that the network architecture of 3 layers with 256 neurons yields the highest performance. Interestingly, the paths generated by the 3 layers 1024 neurons network on average have a higher noise cost, which contradicts the hypothesis that more complex network architectures can generate more complex paths, which are required for higher noise abatement. The results in the table also show an increase in computational time for network architectures of higher complexity, which is expected based on the increase in the number of required computations for more complex networks.

Finally, Figure 6 shows the paths generated by the different networks after training. In this figure the main areas of difference are encircled in red, highlighting slight variations in the policies of the different methods. Figure 6a explains the higher noise cost associated with the 2 layers, 256 neurons model, as the paths are relatively direct and straight. This shows that for paths of higher quality, a

Table 4. Resulting costs and number of turns for the different network architectures.

Architecture	Total Cost	Noise Cost	Fuel Cost	Computation time (s)	# Turns
2 layers, 256 neurons	42.1	32.0	**52.1**	**11.0e-4**	5.4
2 layers, 1024 neurons	40.8	29.4	52.2	12.2e-4	5.6
3 layers, 256 neurons	**40.3**	**28.3**	**52.1**	11.5e-4	5.3
3 layers, 1024 neurons	40.6	29.7	51.5	14.5e-4	5.4

slightly more complex network architecture such as the 3 layers, 256 neurons model is the preferred option, although the exact configuration likely depends on the underlying problem/environment.

4.5 Experiment 2: Impact of discretization resolution (Dijkstra)

Similar to the network architectures for SAC, the discretization resolution used for Dijkstra directly affects the costs of the generated paths when evaluated in the continuous environment simulated in BlueSky. To ensure that the paths used for comparing Dijkstra with SAC are representative of the method, this section will evaluate the quality of the paths resulting from the discretization resolutions given in Table 2 for $\lambda = 0.5$.

The results of this experiment are shown in Table 5. This table shows that a resolution of 65 x 65 (or a cell size of 8 km) leads to the highest performance. Why higher resolutions lead to higher total path costs can be attributed to two components. The first component is that in the continuous environment, noise spreads equally according to the inverse square law, reaching beyond the area of a single cell. Because of this, higher-resolution Dijkstra solutions do not fully capture the total cost of their path accurately. Additionally, the higher-resolution solutions have a higher number of required turns, because the discrete environment does not capture the dynamics required to fly these routes, this creates additional performance losses when evaluated in the simulator.

Table 5. Costs and computation time for various discretization resolutions for Dijkstra.

Cell size (km)	grid-world size	Total Cost	Noise Cost	Fuel Cost	Computation time (s)
13 x 13	40 x 40	42.4	34.7	50.1	**3.98e-3**
10 x 10	52 x 52	43.7	38.5	48.9	4.57e-3
8 x 8	65 x 65	**39.3**	**30.3**	49.1	8.52e-3
5 x 5	104 x 104	45.2	41.1	49.4	4.44e-2
4 x 4	130 x 130	44.0	39.7	48.2	7.26e-2
2 x 2	260 x 260	45.0	42.2	47.9	0.385
1 x 1	520 x 520	47.7	48.3	**47.2**	2.79

4.6 Experiment 3: Comparing SAC and Dijkstra for different cost weight ratios

To validate the performance and implementation of SAC for the path planning task, this experiment will compare the paths generated by SAC in the continuous environments with the optimal single-agent paths in discrete space as determined by Dijkstra. This experiment will use a network architecture of 3 layers with 256 neurons for SAC and a discretization resolution of 65 x 65 (8 x 8 km per cell) for Dijkstra, based on the results described in Sections 4.4 and 4.5.

Because it is hypothesized that SAC has an advantage over Dijkstra for lower values of λ (at $\lambda = 0$ SAC can fly straight to the runway, whereas Dijkstra is limited by the action resolution of the discrete space), this experiment is carried out for five different values for λ: $\lambda = [0.125, 0.25, 0.5, 0.75, 0.875]$.

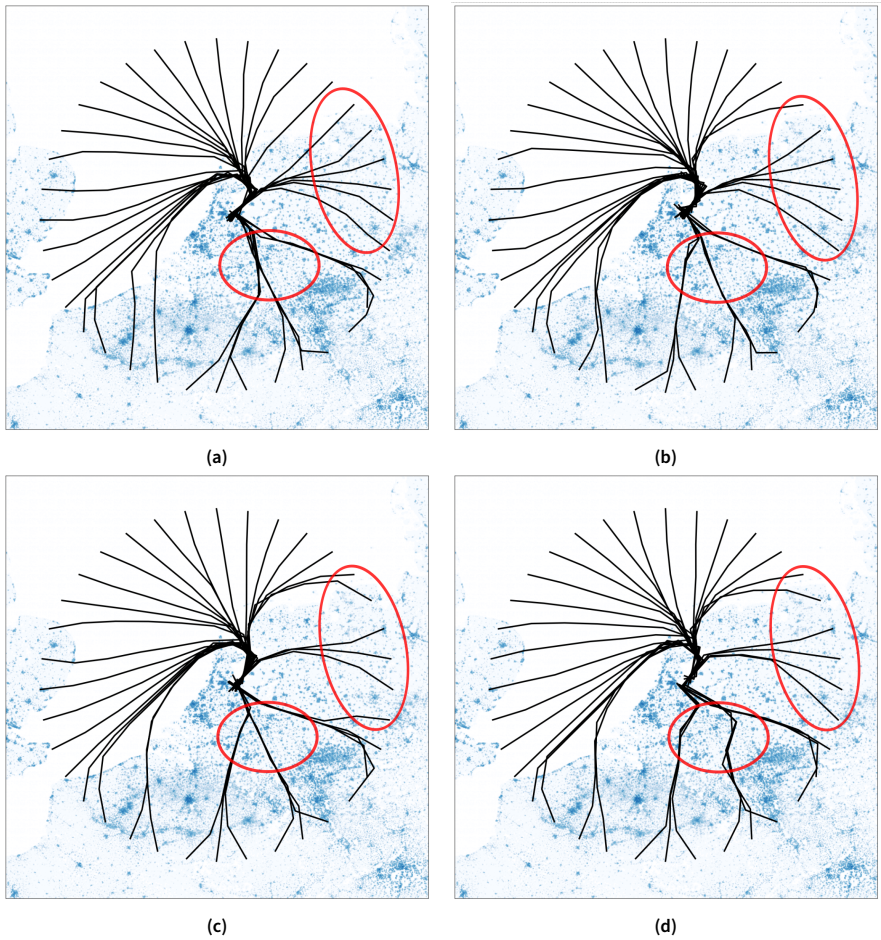


Figure 6. Paths generated by the SAC methods for network architectures: (a) 2 layers, 256 neurons, (b) 2 layers, 1024 neurons, (c) 3 layers, 256 neurons, (d) 3 layers, 1024 neurons. Main areas where differences are observed are encircled in red

The results of this experiment are given in Table 6. Comparing the results of the methods for the different values of λ, it can be observed that the total cost is lowest for Dijkstra, regardless of the value used for λ. Interestingly the paths generated by Dijkstra also score better on the fuel cost for all values of λ, this is in contrast with the hypothesis that SAC would outperform Dijkstra at the lower values of λ and highlights the strength of discrete solvers like Dijkstra, even when applied to problems that appear continuous in nature.

Overall using Dijkstra for the given discretized version of the problem outperforms the mean-based sampling method of SAC for all values of λ. However, comparing the total cost of the SAC method at $\lambda = 0.5, 40.2$, with the results shown in Table 5, we can see that Dijkstra only outperforms SAC for this specific discretization resolution, highlighting the importance of effective discretization when working with discrete solvers.

Additionally, comparing the results from Table 4 with Table 5 it is observed that the computational time for SAC is consistently lower than Dijkstra. This is advantageous for applications where many paths have to be generated in the same stationary environment, such as for the case of a fixed airspace

Table 6. Results for SAC and Dijkstra for $\lambda = [0.125, 0.25, 0.5, 0.75, 0.875]$.

λ	Method	Total cost	Noise cost	Fuel cost	Number of turns
0.125	SAC	45.8	**39.6**	46.7	4.8
	Dijkstra (8x8 km)	**44.7**	41.0	**45.2**	9.2
0.25	SAC	45.3	**35.0**	48.8	5.0
	Dijkstra (8x8 km)	**43.8**	36.2	**46.3**	9.4
0.5	SAC	40.2	**28.3**	52.1	5.3
	Dijkstra (8x8 km)	**39.2**	29.3	**49.1**	11.7
0.75	SAC	34.6	27.2	56.6	5.7
	Dijkstra (8x8 km)	**32.4**	**25.5**	**53.0**	13.7
0.875	SAC	30.5	26.5	58.5	6.3
	Dijkstra (8x8 km)	**30.0**	**26.1**	**57.1**	15.1

or the minimally invasive surgery example given in the introduction, as retraining is not required in that case. It is important to note that this computational time does not consider training time, so for nonstationary environments or environments which are only encountered once, the computational time requirements increase significantly to the point where SAC is not a usable stand-alone method.

Finally, Figure 7 compares the paths generated by the Dijkstra and SAC methods for $\lambda = 0.125$ & $\lambda = 0.875$. This figure highlights some key differences between the methods, mostly the resolution of the solutions, especially around small areas of high population density, where Dijkstra is more successful at navigating around them than SAC. However, some similarities can also be observed, especially in the general structure of the paths and the location of the inflection points.

4.7 Experiment 4: Sampling based evaluation of the learned distribution

Because SAC learns a stochastic policy that follows a normal distribution for exploration during training, using the mean of the learned distribution during evaluation time might result in a sub-optimal action sequence when the underlying action-cost distribution is not normal. It is therefore hypothesized that generating an ensemble of paths through random sampling from the learned distribution can result in paths that improve on the mean-sampled path.

To investigate this effect, this section compares ensembles of paths, generated through random sampling from the learned distribution, with the paths generated when using the mean of the learned distribution. These ensembles consist of 25 paths for each of the initial states described in Section 4.1, for both $\lambda = 0.125$ and $\lambda = 0.875$, using the network architecture of 3 layers and 256 neurons, also used in Section 4.6.

The resulting paths and their total costs are shown in Figure 8. Comparing the paths generated for $\lambda = 0.875$ in Figure 8c, it becomes clear that the width of the learned distribution is related to the allowed margin of error of the given state, with greater deviations in the paths observed above areas of low population density. In contrast, around regions of high population density, the paths generated through sampling from the distribution follow the paths generated by the mean of the distribution (red in the figures) relatively closely.

As hypothesized, the results in Figures 8b and 8d show that for almost all bearings the best sampled path from the ensemble has a lower total cost than the mean paths, indicated by the black line in the boxplots. To highlight this difference in path quality, Table 7 revisits the results from Table 6 but with an additional method for SAC (sampling-based). In this case the best path from each ensemble is compared with the other two methods for the most extreme cases in the cost function.

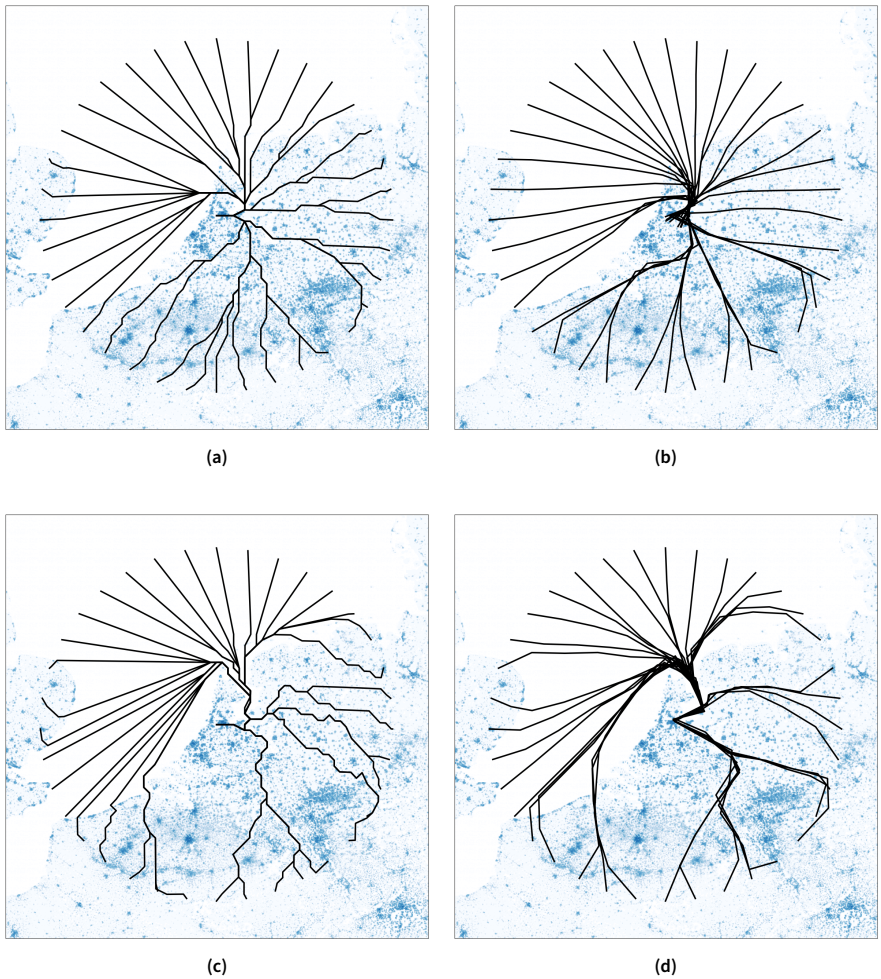


Figure 7. Paths generated for varying noise weight values by the Dijkstra and SAC methods: (a) $\lambda = 0.125$, Dijkstra, (b) $\lambda = 0.125$, SAC, (c) $\lambda = 0.875$, Dijkstra, (d) $\lambda = 0.875$, SAC.

As this sampling is done randomly, it does not guarantee the optimal path has been sampled from the learned distribution. Nevertheless, the results in this table show that a significant increase in performance can be obtained when evaluating multiple potential solutions as generated by SAC through sampling, especially for $\lambda = 0.875$, which puts a higher weight on the noise cost.

Table 7. Results for SAC, SAC (sample based) and Dijkstra for $\lambda = [0.125, 0.875]$.

λ	Method	Total cost	Noise cost	Fuel cost
0.125	SAC	45.8	39.6	46.7
	SAC (sample based)	**44.1**	**35.5**	45.4
	Dijkstra (8x8 km)	44.7	41.0	**45.2**
0.875	SAC	30.5	26.5	58.5
	SAC (sample based)	**25.8**	**21.2**	57.8
	Dijkstra (8x8 km)	30.0	26.1	**57.1**

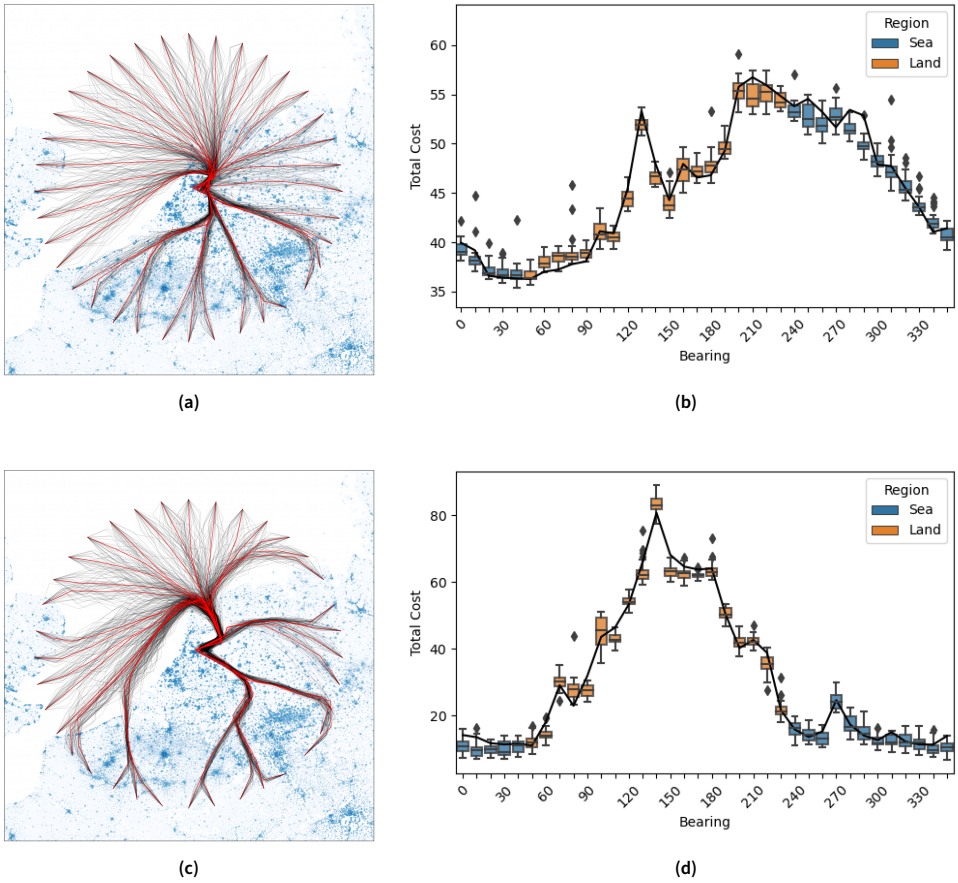


Figure 8. (a), (c) paths generated through sampling from the learned distribution for $\lambda = 0.125$ and $\lambda = 0.875$ respectively, the red paths represent the mean of the distribution. (b), (d) boxplots for the total costs associated with the sampled paths for $\lambda = 0.125$ and $\lambda = 0.875$ respectively, the black lines represent the cost associated with the red paths in (a) and (c).

5. Discussion

The results in Section 4.4 show that for path planning with SAC the choice of network architecture must be carefully considered, as a too simple network architecture can lead to significant decreases in the obtained performance. The network architecture, however, is only a single component of the hyperparameters, and it is possible that more performance improvements for the SAC method can be obtained through careful consideration of the other parameters. Additionaly, it is possible that different formulations of the state, action space and reward function corresponding to the problem can lead to different results. An example would be to use a polar coordinate system instead of cartesian, which might be better suited, considering that all paths have to end up at the origin. Future research should investigate to what extend these additional parameters influence the maximum attainable performance of SAC for a method for path planning.

For the Dijkstra method, Section 4.5 illustrated that the best performance is obtained at a cell size of 8x8 km, this is, however, also a consequence of the discretization method used. Similar to the state representation of the SAC method, a better suited discretization method might have been a polar coordinate based discretization or the expanding tree discretization by Chevalier et al. [20]. As these discretization methods will result in fewer required turns to reach the destination. For

future research it is interesting to isolate the discretization method as an independent variable, to identify how this influences the performance of discrete planners in open airspace. As it is currently unknown which discretization method is best suited for the bi-criterion path planning problem presented in this paper.

When comparing the results of Dijkstra with the mean-based SAC it is found that Dijkstra does result in lower cost paths for the used discretization resolution. Especially at higher noise costs weights Dijkstra demonstrates better performance than SAC, resulting in both better noise and fuel mitigating capabilities. This is likely because noise mitigation requires a high action resolution to avoid small areas of high population. Although in theory SAC is capable of these small adjustments, with the defined action space allowing waypoints at an infinitesimal distance from the current state to be selected, it is observed that the tested SAC policies tend to favour larger actions, as is illustrated by the number of turns in Table 6. This behavior can potentially be explained by the discount-factor, γ, present in the SAC algorithm. This discount-factor reduces the contribution of future rewards and penalties on the expected sum of rewards generated by the value function, in order to mitigate the impact of uncertainty. Because of this, policies that terminate the episode early are favoured over policies that require more actions per episode. To limit the impact of this phenomenon on the performance of the method, it is recommended that future studies investigate the importance of the action space on the performance. By limiting the size of the action space, the method is forced into a smaller resolution similar to that of Dijkstra, potentially resulting in better performance at higher noise cost weights, at the cost of longer training and computation time.

Another interesting observation is that, because SAC relies on neural networks for generating the path, the computational time required after training is lower than that of the Dijkstra implementation of NetworkX. This benefit however is only valid in static environments and environments that require paths to be planned from many different states, as otherwise retraining would be required, negating the low computation time.

Finally, the main contribution of this paper was to identify if using a sampling-based strategy on the learned action distribution of SAC can result in improvements over mean-based evaluation, which is commonly done for distribution methods such as SAC and PPO, especially when the underlying cost distribution is not Gaussian. Section 4.7 evaluated 72 paths through ensembles of 25 paths for each of the initial states, generated through sampling from the learned distribution. Comparing the results obtained from this sampling strategy, Table 7 highlighted the total cost from the best paths of each ensemble, showing a reduction in total cost of -3.8% for $\lambda = 0.125$ and -15.4% for $\lambda = 0.875$, outperforming both naive SAC and Dijkstra. It is interesting to see the difference in performance for this method between the two values for λ. This can be explained by the fact that a distance driven cost as a function of action, as is the case for $\lambda = 0.125$, is relatively normally distributed. Therefore taking the mean of the learned distributions is closer to the optimal action for a given state in this scenario. For the more complex cost function, the one driven by noise, this is not the case, leading to more potential improvements for alternative evaluation strategies.

These results highlight some shortcomings of distribution-based RL methods such as SAC and PPO. However, they also indicate opportunities and show that significant performance improvements can be obtained by altering the evaluation strategies used after training. Future research should therefore investigate if there are better evaluation strategies, for example by sampling from the learned distribution, and then using the critic networks of the methods to evaluate the different samples. The highest evaluated action could then be selected as the actual action instead of the mean of the learned distribution, which is especially helpful in cases where you are unable to fully roll out various solutions, such as real-time applications.

6. Conclusions

This paper compared SAC, an RL algorithm that trains a Gaussian distribution for the action policy, with Dijkstra for a global path planning task in a terminal airspace. SAC has seen previous successes in path planning in continuous environments, outperforming other planning algorithms in terms of solution quality. Because the policy used for generating the paths follows a Gaussian distribution it was hypothesized that for more complex cost functions the mean of the distribution, which is normally used during evaluation, does not result in the best performance when the underlying cost function is not normally distributed. To test this, this paper also evaluated a sampling-based strategy on the learned policy to generate an ensemble of solutions with different associated costs.

It was found that a mean-based evaluated of SAC does not outperform a Dijkstra implementation which has been tuned for the appropriate discretization resolution. However, when using the trained SAC policy to generate an ensemble of solutions through sampling, it is found that the best solutions in the ensemble outperform both Dijkstra and mean evaluated SAC, with the performance differences growing for more complex cost functions. This strengthens the hypothesis that mean-based evaluation for distribution models such as SAC and PPO might lead to suboptimal performance.

Author contributions

- D.J Groot: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing (Original Draft)
- J. Ellerbroek: Conceptualization, Project Administration, Supervision, Writing (Review and Editing)
- J.M. Hoekstra: Conceptualization, Project Administration, Supervision, Writing (Review and Editing)

Funding statement

Not applicable

Open data statement

All code used for generating the results of this paper is available through the 4TU data repository [16]:
https://data.4tu.nl/datasets/b68d7aa9-235b-4f8b-b8c8-a977eaceba50
doi: 10.4121/b68d7aa9-235b-4f8b-b8c8-a977eaceba50.v1

Reproducibility statement

The code is structured in 3 folders, all using the same population data, obtained from Eurostats. The discrete_environment folder contains all of the code related to the discretization, Dijkstra solutions and postprocessing of the Dijkstra output. The continuous_environment folder contains a fork of the BlueSky Open Air Traffic Simulator repository, with all of the plugins related to training the Deep Reinforcement Learning algorithm, and evaluating of the paths in the continuous environment. Finally the policy_plotter folder contains all of the tools for generating the visuals presented in the paper.

References

[1] SESAR Joint Undertaking et al. "European ATM master plan: the roadmap for delivering high performing aviation for Europe: executive view: edition 2015." In: (2016).

[2] S Aneeka and ZW Zhong. "NOX and CO2 emissions from current air traffic in ASEAN region and benefits of free route airspace implementation". In: *Journal of Applied and Physical Sciences* 2.2 (2016), pp. 32–36.

[3] Weili Zeng, Xiao Chu, Zhengfeng Xu, Yan Liu, and Zhibin Quan. "Aircraft 4D trajectory prediction in civil aviation: A review". In: *Aerospace* 9.2 (2022), p. 91.

[4] Zhuang Wang, Weijun Pan, Hui Li, Xuan Wang, and Qinghai Zuo. "Review of deep reinforcement learning approaches for conflict resolution in air traffic control". In: *Aerospace* 9.6 (2022), p. 294.

[5] Pouria Razzaghi, Amin Tabrizian, Wei Guo, Shulu Chen, Abenezer Taye, Ellis Thompson, Alexis Bregeon, Ali Baheri, and Peng Wei. "A survey on reinforcement learning in aviation applications". In: *Engineering Applications of Artificial Intelligence* 136 (2024), p. 108911.

[6] Jan Groot, Joost Ellerbroek, and Jacco Hoekstra. "Analysis of the impact of traffic density on training of reinforcement learning based conflict resolution methods for drones". In: *Engineering Applications of Artificial Intelligence* 133 (2024), p. 108066.

[7] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. "Hierarchical reinforcement learning: A comprehensive survey". In: *ACM Computing Surveys (CSUR)* 54.5 (2021), pp. 1–35.

[8] Ye Zhang, Wang Zhao, Jingyu Wang, and Yuan Yuan. "Recent progress, challenges and future prospects of applied deep reinforcement learning: A practical perspective in path planning". In: *Neurocomputing* 608 (2024), p. 128423.

[9] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[10] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[12] European Commission. *GISCO: GEOGRAPHICAL INFORMATION AND MAPS*. data retrieved on 26/10/2022 from Eurostat, https://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/grids. 2018.

[13] Eurocontrol Experimental Center. *The Aircraft Noise and Performance (ANP) Database*. data retrieved on 26/10/2022 from Eurocontrol, https://www.aircraftnoisemodel.org/home. 2020.

[14] Junzi Sun, Jacco M Hoekstra, and Joost Ellerbroek. "OpenAP: An open-source aircraft performance model for air transportation studies and simulations". In: *Aerospace* 7.8 (2020), p. 104.

[15] Jacco M Hoekstra and Joost Ellerbroek. "Bluesky ATC simulator project: an open data and open source approach". In: *Proceedings of the 7th international conference on research in air transportation*. Vol. 131. FAA/Eurocontrol USA/Europe. 2016, p. 132.

[16] Jan Groot. *Code accompanying the paper on aircraft path planning in continuous environments with deep reinforcement learning*. https://data.4tu.nl/datasets/b68d7aa9-235b-4f8b-b8c8-a977eaceba50. 2023.

[17] TUDelft-CNS-ATM. *BlueSky Open Air Traffic Simulator*. https://github.com/TUDelft-CNS-ATM/bluesky. 2023.

[18] Andrew G Barto. "Reinforcement Learning: An Introduction. By Richard's Sutton". In: *SIAM Rev* 6.2 (2021), p. 423.

[19] Tobias Andersson Granberg, Tatiana Polishchuk, Valentin Polishchuk, and Christiane Schmidt. "Automatic design of aircraft arrival routes with limited turning angle". In: *16th Workshop*

on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.

[20] Jeremie Chevalier, Daniel Delahaye, Mohammed Sbihi, and Pierre Marechal. "Departure and arrival routes optimization near large airports". In: *Aerospace* 6.7 (2019), p. 80.