BANACH PRESS

JOURNAL OF CONVEX ANALYSIS
Volume 34 (2027), No.1, 458–470
https://journalofconvexanalysis.com

Autonomous navigation in realistic urban environments using machine learning and path planning

Gonçalo Penelas[1,*], Tiago Pinto[1], Luís Barbosa[1], Arsénio Reis[1], João Barroso[1]

1. Universidade de Trás-os-Montes e Alto Douro, Portugal

Abstract: Deep Reinforcement Learning (DRL) has achieved remarkable results in simulated environments for its ability to learn optimal actions through interaction. Yet, applying these techniques to complex urban contexts remains challenging due to the need for efficient exploration and adaptive decision-making. This work applies DRL to autonomous navigation in a realistic simulated environment based on Vila Real, Portugal, comparing the Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms. Alongside their implementation, the mathematical principles underlying both methods are examined to clarify their learning dynamics. The PPO agent achieved a best cumulative reward of 69.37 after 150 million training steps (66.1 hours), while SAC obtained -7.91 after 182.7 hours, showing limited performance. These results demonstrate the effectiveness of PPO-based DRL for autonomous navigation in complex urban environments.

Keywords: Artificial Intelligence, Machine Learning, Deep Reinforcement Learning, Pathfinding Algorithm, Unity.

1. Introduction

Autonomous navigation remains one of the central challenges in Artificial Intelligence (AI), particularly in realistic and dynamic urban environments where uncertainty, traffic complexity, and environmental variability make optimal decision-making difficult. Despite significant advances, many existing studies rely on simplified maps or static scenarios, which limit the agent's ability to generalize and adapt to real-world conditions [1, 2]. This gap highlights the need for approaches that combine realistic simulation environments with advanced learning mechanisms capable of handling high-dimensional and non-deterministic contexts.

AI aims to develop autonomous agents capable of learning optimal behaviors through interaction with their environment. In domains such as healthcare and autonomous navigation, the need for adaptive and responsive systems has grown considerably [3, 4]. Within this domain, Machine Learning (ML) enables systems to automatically learn and improve from experience without being explicitly programmed [5]. Among its main approaches, Deep Learning (DL) and Reinforcement Learning (RL) form the foundation of Deep Reinforcement Learning (DRL), a paradigm that enables agents to learn complex behaviors and make decisions through interaction with their environment [6]. In this work, DRL is instantiated through two representative algorithms, Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).

This work contributes to autonomous driving research by applying DRL in a realistic simulation environment. It builds on previous research that presented the initial development phase of the solution [7]. The setup includes a motorcycle operating within an urban map of Vila Real, Portugal, reconstructed from real data. This environment enables the agent to learn dynamic navigation strategies and adapt to changing conditions. Beyond learned control, a deterministic path-planning layer (A* over Unity NavMesh) structures long-horizon navigation via intermediate checkpoints, improving stability in complex urban layouts. At the same time, the proposed framework provides an applied instance of topics central to optimization and control: the navigation task is formulated as a sequential decision-making problem, where PPO and SAC

*Corresponding author.
E-mail addresses: G. Penelas*: goncalopenelas@utad.pt
https://doi.org/10.68381/jca34031
Received date: October 30, 2025; Accepted date: March 30, 2026
© 2027 Published by Banach Press LLC. (*Dedicated to Alexander Plakhov on the occasion of his 65th birthday.*)

update a parameterized policy through iterative optimization, while the A* layer computes cost-minimizing routes over a discretized domain. In addition, the learning objectives used by PPO involve operations such as clipping, which relate naturally to non-smooth viewpoints that are of interest in the broader analysis and optimization literature.

The remainder of this paper is organized as follows. Section 2 introduces the fundamentals of Reinforcement Learning, while Section 3 reviews the principles of Deep Learning relevant to this study. Section 4 presents the proposed solution, detailing the development of the learning system, including the integration of Deep Reinforcement Learning (PPO and SAC) with the A* path-planning algorithm for autonomous navigation. Section 5 reports and discusses the experimental results obtained, including the comparative analysis of the best-performing PPO and SAC models. Finally, Section 6 concludes the paper, outlining the main findings and directions for future work.

2. Reinforcement Learning

RL is a branch of machine learning that studies how agents achieve predefined goals through trial-and-error interaction with an environment [8].

According to Banerjee et al. [9], the general RL framework can be described as a stochastic sequential decision-making process, typically formalized through a Markov Decision Process (MDP). It consists of the following components:

- A finite set of states $\mathcal{S}$ describing the environment;
- A finite set of actions $\mathcal{A}$ available to the agent;
- A reward function $R(s_t, a_t)$ providing feedback for each state–action pair;
- A transition probability distribution $P(s_{t+1} \mid s_t, a_t)$ defining the environment dynamics;
- A discount factor $\gamma \in [0, 1]$ weighting future rewards relative to immediate ones.

As described by Mashayekhi et al. [10], the agent–environment interaction evolves as follows: at each timestep t, the agent observes the current state s_t and reward r_t, then selects an action a_t from the set of available actions $\mathcal{A}$ according to its policy $\pi(s, a) = \Pr(a_t = a \mid s_t = s)$. After performing the action, the environment transitions to a new state s_{t+1} and provides a new reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}).

The cumulative reward from time t is expressed as:

$$R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i), \tag{1}$$

where $r(s_i, a_i)$ denotes the immediate reward for taking action a_i in state s_i, and γ represents the discount factor that prioritizes present over future rewards. The objective of RL is to learn a policy π^* that maximizes the expected return $J = \mathbb{E}[R_0]$, which can be achieved through stochastic or deterministic optimization methods.

3. Deep Learning

Deep Learning (DL) relies on artificial neural networks (ANNs), i.e., interconnected artificial neurons with trainable weights and nonlinear activation functions [11]. DL provides a computational architecture composed of multiple processing layers capable of learning hierarchical and nonlinear representations of data [12]. A commonly used architecture is the Multi-Layer Perceptron (MLP), a fully connected feedforward network that serves as a general-purpose function approximator [13].

This approach enables the extraction of complex features from large datasets, offering high precision in tasks such as classification, prediction, and data generation [14]. Moreover, it can integrate with other paradigms, such as Reinforcement Learning, where neural networks are used as function approximators or state representation models [15].

The following mathematical formulation is adapted from Petersen and Zech [16].

3.1. Deep Neural Networks

A neural network is composed of interconnected neurons that transform input data through weighted connections. A single neuron is defined as:

$$v(x) = \sigma(w^\top x + b), \tag{2}$$

where $x \in \mathbb{R}^d$ denotes the input vector, $w \in \mathbb{R}^d$ the weight vector, $b \in \mathbb{R}$ a bias term, and $\sigma(\cdot)$ an activation function introducing nonlinearity into the model.

3.2. Gradient-Based Training

The learning process aims to determine the network parameters $\theta = \{w, b\}$ that minimize a loss function measuring the difference between predicted and target outputs. For a training set $S = \{(x_i, y_i)\}_{i=1}^m$, the empirical risk is defined as:

$$\widehat{R}_S(\Phi) = \frac{1}{m} \sum_{i=1}^{m} L(\Phi(x_i), y_i), \tag{3}$$

where $L(\cdot)$ is a differentiable loss function. Parameter updates are performed iteratively through gradient-based methods such as stochastic gradient descent:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \widehat{R}_S(\Phi),$$

with η denoting the learning rate.

3.3. Prediction

After training, the model's performance on unseen data is evaluated through the expected risk:

$$R(\Phi^*) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[L(\Phi^*(x), y) \right], \tag{4}$$

where $\mathcal{D}$ represents the underlying data distribution. A small difference between $R(\Phi^*)$ and $\widehat{R}_S(\Phi)$ indicates good generalization, while a large difference suggests overfitting to the training data [9].

4. Proposed Solution

This section presents the proposed solution for autonomous navigation based on Deep Reinforcement Learning (DRL) integrated with deterministic path planning. The main objective is to enable a motorcycle agent to learn and perform efficient navigation strategies within a realistic urban environment. The proposed architecture is organized into three main components: the environment, the agent, and the learning system. The learning system combines two complementary layers: a learning-based control mechanism, where the agent adapts its behavior through interaction with the environment using DRL algorithms (PPO and SAC), and a planning layer based on the A* algorithm, responsible for structuring long-horizon navigation through dynamically generated checkpoints that guide the agent towards its targets.

4.1. Deep Reinforcement Learning

The new generation of reinforcement learning algorithms, which leverage the computational power of modern GPUs and combine reward–penalty optimization with deep neural representations, is known as Deep Reinforcement Learning (DRL) [17]. DRL integrates Deep Learning (DL) and Reinforcement Learning (RL) to enable agents to learn complex behaviors through interaction with their environment, using neural networks to approximate value and policy functions. In recent years, this approach has achieved remarkable results across several domains, including robotics control, autonomous driving, industrial automation, healthcare, financial trading, cybersecurity, and gaming [18, 19].

To facilitate the implementation of DRL techniques within this project, the Unity ML-Agents Toolkit [20] was employed, as it provides a straightforward framework for applying algorithms such as PPO and SAC. These algorithms were selected for their proven ability to train agents capable of making efficient and robust decisions in complex and dynamic environments.

4.1.1. Proximal Policy Optimization (PPO)

PPO is an on-policy optimization algorithm that uses a neural network to define an optimal policy by associating an agent's observations with the best possible action in a given state. When integrated with frameworks such as TensorFlow or Unity ML-Agents, the algorithm interacts with Python processes to optimize training parameters through gradient descent. By estimating both the direction and magnitude of the cost function change, PPO achieves more efficient and stable training than traditional gradient descent methods [21].

The algorithm follows an actor–critic architecture, where the *actor* is responsible for selecting actions and the *critic* evaluates them. This structure combines policy updates with feedback from value estimation, preventing destructive policy changes common in traditional methods. Through alternating between data collection and gradual policy updates that do not deviate excessively from the previous policy, PPO achieves a balance between simplicity, computational efficiency, and effective data utilization [22].

The core idea of PPO is to optimize a surrogate objective function while constraining policy updates to avoid large deviations from the previous policy [23]. The main objective function is defined as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right], \tag{5}$$

where:

- θ represents the policy parameters (neural network weights);
- $\hat{\mathbb{E}}_t$ is the empirical expectation over sampled timesteps;
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies;
- $\hat{A}_t$ is the estimated advantage at time t;
- ϵ is a small hyperparameter controlling the clipping range.

The objective $L^{CLIP}(\theta)$ ensures that the new policy does not deviate significantly from the old one by taking the minimum between the unclipped objective $r_t(\theta)\hat{A}_t$ and its clipped version. This conservative update maintains a balance between exploration and exploitation, resulting in more stable and reliable learning [23, 24]. In summary, PPO penalizes large policy changes to ensure smooth and stable policy updates while optimizing performance.

4.1.2. Soft Actor-Critic (SAC)

Unlike PPO, the SAC algorithm follows an off-policy approach, meaning it can learn from previously collected experiences stored in a replay buffer. During training, random samples are drawn from this buffer, which significantly improves sample efficiency—often requiring five to ten times fewer samples than PPO to achieve comparable performance. However, SAC typically requires more model updates per episode, making it suitable for environments where each simulation step is computationally expensive [25].

SAC is an off-policy actor–critic algorithm based on the maximum entropy framework. Its objective is to maximize both the expected reward and the entropy of the policy, encouraging stochastic behaviors that improve exploration and stability [26, 27]. The key loss functions in SAC are defined as follows:

Q-function update. The Q-function parameters are optimized by minimizing the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}\left(Q_\theta(s_t,a_t) - \left(r(s_t,a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}\left[V_{\bar{\theta}}(s_{t+1})\right]\right)\right)^2\right], \tag{6}$$

where $Q_\theta(s_t,a_t)$ is the Q-function parameterized by θ, $r(s_t,a_t)$ is the reward, γ is the discount factor, and $V_{\bar{\theta}}(s_{t+1})$ is the target value function.

Value function update. The value function minimizes the difference between the expected Q-value and the entropy-regularized return:

$$J_V(\bar{\theta}) = \mathbb{E}_{s_t\sim D}\left[\frac{1}{2}\left(V_{\bar{\theta}}(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}\left[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t|s_t)\right]\right)^2\right]. \tag{7}$$

Policy update. The policy parameters are optimized to maximize both expected reward and entropy:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D,\, a_t \sim \pi_\phi} \left[\log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t) \right]. \tag{8}$$

Entropy regularization. The entropy of the policy ensures that it remains sufficiently stochastic:

$$\mathcal{H}(\pi(\cdot|s)) = -\mathbb{E}_{a \sim \pi} \left[\log \pi(a|s) \right]. \tag{9}$$

By alternating between these updates—Q-function, value function, and policy—the SAC algorithm achieves efficient and stable learning while maintaining a balance between exploration and exploitation [28, 29, 30, 31].

4.2. Path Planning with A* Algorithm

To improve navigation consistency and reduce suboptimal exploration by the agent, a deterministic path planning layer was integrated using the A* algorithm. This approach relies on Unity's *NavMesh* to discretize the environment into traversable nodes, upon which A* computes an optimal route between the agent's current position and the nearest target. The route is discretized into a sequence of intermediate checkpoints, which is regenerated after each target is collected using the agent's updated position to plan towards the next nearest target, providing structured waypoints that guide the agent throughout the episode.

Mathematically, A* extends Dijkstra's algorithm by introducing a heuristic term that estimates the remaining cost to the goal, effectively balancing exploration and exploitation during path expansion [32]. The parameterization of this method is defined as [32, 33]:

$$f(n) = g(n) + h(n) \tag{10}$$

where:

- $f(n)$ denotes the total estimated cost from the start node to the goal through node n;
- $g(n)$ represents the accumulated cost from the start node to n;
- $h(n)$ is the heuristic estimate of the cost from n to the goal.

In this implementation, nodes n correspond to traversable positions extracted from the *NavMesh*. The real cost $g(n)$ is defined as the Euclidean distance between consecutive nodes, while the heuristic term $h(n)$ is calculated as the Euclidean distance from n to the target [34]:

$$h(n) = \sqrt{(x_n - x_t)^2 + (y_n - y_t)^2 + (z_n - z_t)^2} \tag{11}$$

This heuristic ensures admissibility and consistency, guiding the search efficiently towards the optimal path. At each iteration, A* expands the node with the lowest total estimated cost $f(n)$, iteratively constructing the minimal-cost path until the goal is reached. The sequence of nodes obtained defines the route, subsequently partitioned into checkpoints used by the agent as intermediate learning targets.

4.3. Learning System Implementation

The learning process in this work is structured around four main components: initialization, action selection, reward assignment, and environment interaction. Training begins with the configuration of the environment and its components. At the start of each episode, the agent is repositioned, and the targets are either generated randomly or manually defined to represent a specific scenario, resetting all states to ensure that both the environment and the agent are ready. At this stage, the agent identifies the closest target based on the minimum distance among all available targets. Subsequently, a path of checkpoints is generated to guide the agent during navigation towards the selected target.

The observations collected by the agent during training are crucial for decision-making. These include the motorcycle's forward speed and steering angle towards the nearest checkpoint or, when all checkpoints are collected or absent due to proximity, towards the closest target. Two sensors are integrated into the system: one to detect obstacles, boundaries, checkpoints, and targets, and another to recognize the road. The collected data are processed so that the learning model can act according to the current environment state, using inputs for steering, acceleration, and braking.

The training process is driven by a reward system. The agent receives positive rewards for reaching each checkpoint and target, with an additional bonus when all generated checkpoints and targets are collected. Negative rewards are applied for undesired behaviors, such as leaving the road, colliding with obstacles or environment limits, or maintaining low, reverse, or zero velocity. The agent is also penalized if the distance to the next objective (checkpoint or target) increases. Time plays a crucial role, with a penalty applied whenever the agent exceeds the ideal 30-second time limit for reaching a checkpoint or target.

Equation 12 defines the reward function adopted in this work, including the weights assigned to each component. Due to the high computational cost of each training session, which made an exhaustive optimization of these weights infeasible, the values were set empirically through an iterative trial-and-error process to balance incentives and penalties. The resulting reward design was validated via rollout inspection to confirm that higher cumulative reward aligned with goal-directed progress (checkpoint/target completion), rather than reward hacking or overly conservative policies.

$$\begin{aligned} R_t = r_{checkpoint} \cdot 1_{(reached_checkpoint)} + r_{target} \cdot 1_{(reached_target)} \\ + r_{all_checkpoints} \cdot 1_{(all_checkpoints_collected)} + r_{all_targets} \cdot 1_{(all_targets_collected)} \\ + p_{distance} \cdot 1_{(distance_increased)} + p_{off_road} \cdot 1_{(off_road)} \\ + p_{collision} \cdot 1_{(collision)} + p_{time} \cdot 1_{(timeout)} \\ + p_{velocity} \cdot 1_{(low_reverse_zero_speed)} \end{aligned} \tag{12}$$

where:

- **Rewards:**
  - $r_{checkpoint} = +1$ — reaching a checkpoint;
  - $r_{target} = +2$ — reaching a target;
  - $r_{all_checkpoints} = +2$ — collecting all checkpoints;
  - $r_{all_targets} = +5$ — collecting all targets.
- **Penalties:**
  - $p_{distance} = -0.01 \cdot |dotValue|$ — moving away from the next objective (*dotValue* measures the misalignment between the motorcycle and the objective);
  - $p_{off_road} = -0.2$ — navigating off the road;
  - $p_{collision} = -2$ — collision with an obstacle or environment boundary;
  - $p_{time} = -1$ — exceeding the time limit to reach an objective;
  - $p_{velocity} = -0.005$ — maintaining low, reverse, or zero speed.

The episode ends when the agent collects all targets or commits a critical error, such as colliding with an obstacle or boundary, or exceeding the time limit to reach the next objective.

To complement this setup, we explored hyperparameters in a limited, manual manner, as automated tuning was not feasible in our context and the high computational cost of training constrained further refinement. We therefore tested three configurations per algorithm, selected to cover different points within recommended ranges while keeping additional time and resource costs manageable. The corresponding configuration values are reported alongside the results.

The implementation of this learning system followed an iterative and adaptive logic, providing a solid foundation for the agent to progressively learn to achieve the proposed targets based on the experience gained.

5. Results

5.1. Results of PPO Configurations

The three configurations tested for the PPO algorithm showed overall positive performances, with one configuration standing out clearly (Figure 1).

The model with the configuration *AgentBehaviour_1* clearly stood out as the most efficient. Throughout the training, the agent demonstrated a growing and relatively stable cumulative reward curve, with few oscillations and a final performance significantly superior to the others. This evolution indicates that an effective and adaptable policy was learned, resulting in a robust and convergent training trajectory. The agent achieved a final cumulative reward of 69.37 points after approximately 66.1 hours of training. This performance suggests that the use of a moderate and constant learning rate (0.0003) may have contributed to a good balance between learning speed and stability. The values of batch_size (4096) and buffer_size (409600) may have provided sufficient diversity of experiences without compromising efficiency. The neural network of three layers with 256 hidden units each may have ensured adequate capacity to learn relevant patterns from the environment without overloading the model complexity. Parameters such as epsilon (0.2) and beta (0.007) seem to have favored a good balance between exploration and regularization, contributing to more consistent learning.

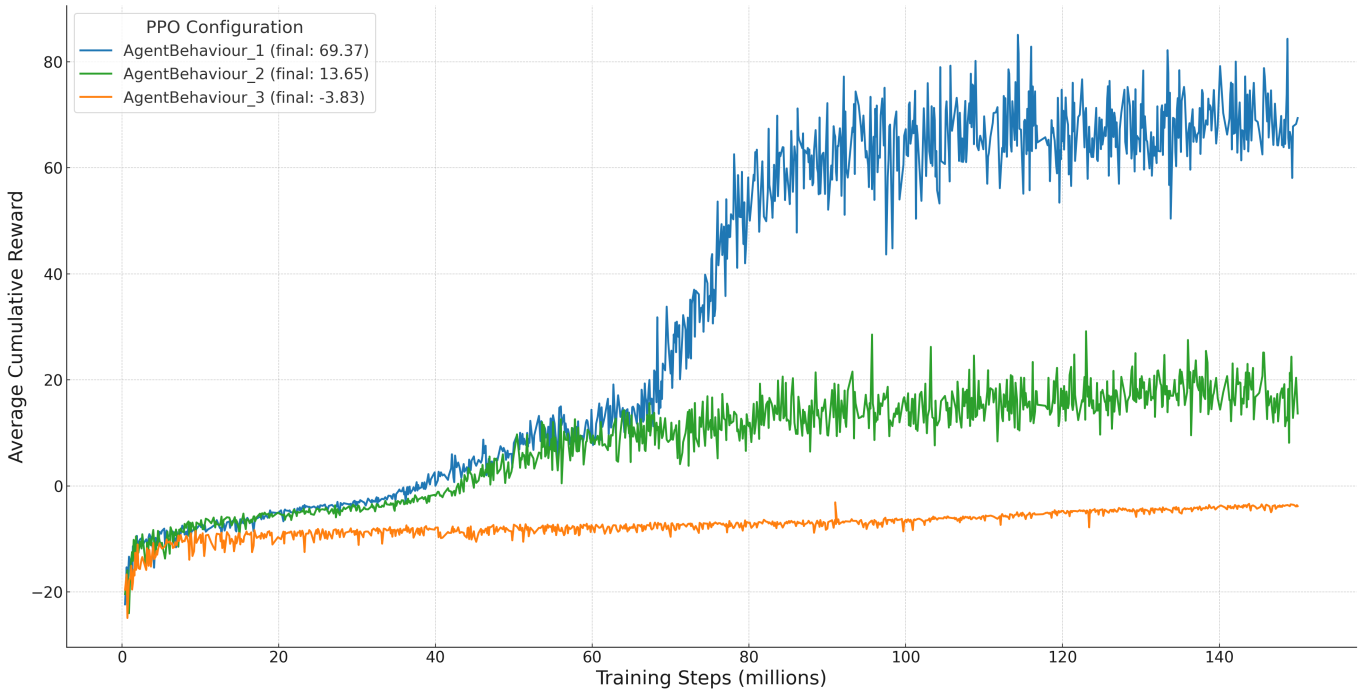

Figure 1. *Comparison of cumulative reward evolution for the PPO configurations.*

The model with the configuration *AgentBehaviour_2* presented an intermediate performance. The agent showed an initial evolution but stagnated before reaching the levels obtained by the most efficient configuration. The final cumulative reward was 13.65 points, with a total training time of about 75.7 hours. This limitation may be related to the use of a higher and linear learning rate (0.0005), which may have favored faster learning in the initial phases but led to premature convergence to a suboptimal policy and reduced learning progress in later stages. The batch_size (2048) and buffer_size (204800), being smaller, may have allowed more frequent updates but with less diversity of experiences. The neural network, composed of two layers with 512 hidden units each, prioritized per-layer capacity over depth, which may have limited generalization. The parameters epsilon (0.15) and beta (0.005) may have contributed to a more conservative and less exploratory policy, leading to the early adoption of suboptimal behaviors. In addition, the reduction of the time_horizon (512), although useful to provide more immediate feedback, may have made the agent unable to plan in the long term.

The model with the configuration *AgentBehaviour_3* registered the worst performance, with a final cumulative reward of –3.83 points after approximately 76.3 hours of training. The learning curve was unstable throughout the process, with frequent oscillations and no evident progress. This negative evolution may be related to the use of a very low learning rate (0.0001), which may have made the agent's adaptation difficult, making learning slow. The high values of batch_size (8192) and buffer_size (819200) may have reduced the frequency of policy updates, delaying the impact of more recent experiences. The neural network, composed of four layers with 128 hidden units each, presented a deep structure but with reduced per-layer capacity, which may have made it difficult to learn more complex patterns from the environment. The parameters epsilon (0.25) and beta (0.01) seem to have encouraged an excessively exploratory policy,

compromising the consolidation of effective behaviors. Finally, the high time_horizon (2048) may have delayed the feedback of actions.

In summary, the results demonstrate that the model with the configuration *AgentBehaviour_1*, combining a moderately complex architecture, a constant learning rate, and balanced exploration and regularization hyperparameters, proved to be more effective for the environment in question. These results highlight the importance of careful and systematic configuration of parameters, essential to achieve robust performance in complex contexts.

5.2. Results of SAC Configurations

The three configurations tested for the SAC algorithm showed overall unsatisfactory performances. Unlike what was observed with PPO, the agents trained with SAC were not able to reach high or even positive levels of cumulative reward, showing learning curves characterized by oscillations, stagnations, and lack of convergence. Although three configurations with different approaches were tested, the results suggest that SAC may not have been the most suitable algorithm for the scenario in question, or that it requires an even more precise configuration to achieve satisfactory performance (Figure 2).

Figure 2. *Comparison of cumulative reward evolution for the SAC configurations.*

The model with the configuration *AgentBehaviour_1* achieved the best relative performance, with a final cumulative reward of –7.91 points after approximately 182.7 hours of training. This behavior may be associated with the use of a constant and moderate learning rate (0.0003), which may have favored stability but also limited adaptability. The values of batch_size (256), buffer_size (4,000,000), and buffer_init_steps (10,000) may have created a conservative and reliable training base, but insufficient to produce significant improvements. The neural network, composed of three layers with 256 hidden units each, may have been balanced but unable to adequately represent the environment's dynamics. The value of init_entcoef (0.2) provided a moderate level of entropy, which may have been insufficient to encourage the exploration of more effective strategies.

The model with the configuration *AgentBehaviour_2* obtained a lower cumulative reward (–8.96 points) and required a significantly longer time than the others, with about 681.5 hours of training, i.e., more than 28 continuous days. The use of a higher and linearly decaying learning rate (0.0005) was intended to accelerate early learning but may have ultimately compromised policy stability over time. Since all SAC configurations were trained under the same experimental conditions, this unusually long duration is most plausibly attributable to the specific hyperparameter configuration (and its associated computational overhead) rather than the intrinsic complexity of the Vila Real environment or Unity–Python communication effects. The values of batch_size (512), buffer_size (5,000,000), and buffer_init_steps (50,000) may have increased the computational and memory demands of training and required a longer warm-up phase before the

updates became fully effective, which may have contributed to slower learning and instabilities. The neural network, composed of two layers with 512 hidden units each, prioritized per-layer capacity over depth, which may have negatively affected generalization. The value of init_entcoef (0.1) and the reduced time_horizon (512) may have favored a more reactive policy but one incapable of consolidating effective long-term behaviors.

The model with the configuration *AgentBehaviour_3*, with a total training time of 182.5 hours, registered the worst performance, with a final cumulative reward of –9.23 points. The combination of a very low learning rate (0.0001) with high values of batch_size (1024) and buffer_size (8,000,000), associated with buffer_init_steps (200,000), resulted in excessively slow training with poor adaptability. The neural network, composed of four layers with 128 hidden units each, offered depth but limited per-layer capacity, which may have hindered the learning of relevant patterns. The value of init_entcoef (0.05) may have contributed to low entropy, promoting an excessively conservative policy, while the high time_horizon (2048) may have further delayed feedback, making policy evolution even more difficult.

Overall, all SAC configurations showed weak performances and long training times, especially the model with the configuration *AgentBehaviour_2*. These results reflect a trend already observed in the testing phase, where SAC showed difficulties in achieving stability and convergence. Although tested with a diverse set of configurations, the performance differences between them were minimal, raising the question of whether the results are due exclusively to the algorithm's sensitivity to hyperparameters or whether SAC may indeed be less suitable for this type of environment.

5.3. Comparison of the Best Models

The comparative analysis of the results obtained for the PPO and SAC algorithms confirms a clear and consistent advantage of PPO throughout the entire experimental process. Already in the initial testing phase, PPO demonstrated greater stability and convergence ability, quickly achieving a positive cumulative reward, while SAC remained oscillating around negative values. This trend continued in the final phase, even after applying optimized configurations (Figure 3).

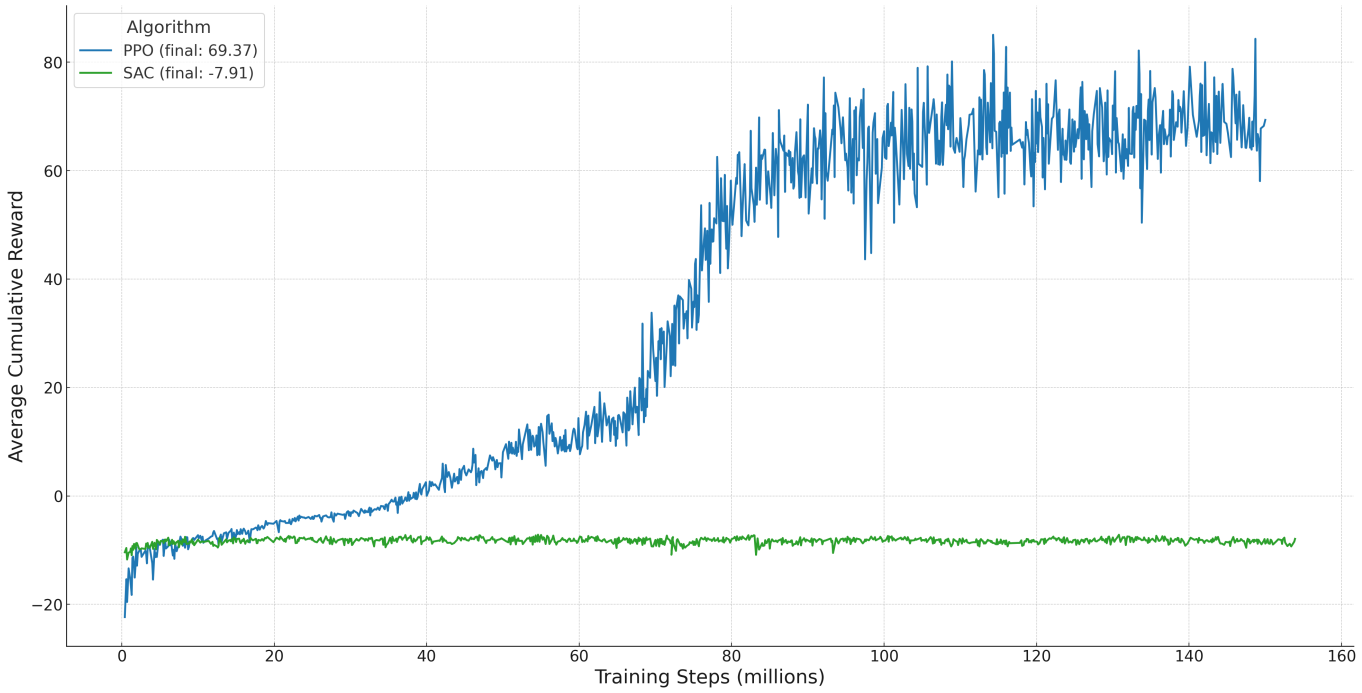

Figure 3. *Comparison of the best-performing PPO and SAC models.*

Considering the best configurations of each algorithm (*AgentBehaviour_1*), PPO reached a final cumulative reward of 69.37 points in 66.1 hours, while SAC ended with –7.91 points in 182.7 hours (Table 1). This difference not only demonstrates the superior capability of PPO in terms of performance but also its greater computational efficiency and faster convergence. PPO was able to extract relevant patterns and stabilize the policy effectively, even in a complex environment, while SAC proved more prone to instabilities and adaptation difficulties.

Table 1. *Comparison between the best PPO and SAC models.*

Algorithm	Final Cumulative Reward	Training Time (hours)
PPO	69.37	66.1
SAC	–7.91	182.7

Although SAC theoretically possesses a more flexible and sophisticated exploration mechanism, the obtained results raise doubts about its practical adequacy for this type of task. The overall weak performance of SAC, combined with the small variation among its tested configurations, raises the question of whether the results are due solely to its sensitivity to hyperparameters or whether they reflect a deeper limitation of the algorithm's suitability for the type of task and environment under study. This question becomes even more relevant since it was not possible to employ automatic hyperparameter tuning methods, which would allow a more precise adjustment of the models and more solid conclusions regarding the impact of different combinations and algorithms.

In summary, the results indicate that PPO showed to be the most suitable option, standing out as the most effective—and the only successful—approach for navigation and decision-making tasks in continuous spaces. The algorithm demonstrated not only high stability and consistency in performance but also efficient adaptability, requiring less configuration effort and significantly lower computational resources. These characteristics position PPO as a robust solution for demanding environments where learning efficiency and decision-making reliability are essential.

5.4. Application of the Model in a Realistic Scenario

To validate the performance of the trained model, a hypothetical scenario with realistic characteristics was designed, representing a courier riding an A-MoVeR motorcycle, rented by the respective employer, to deliver products to citizens with reduced mobility in the city of Vila Real (route shown in Figure 4).

Figure 4. *Route of the simulated delivery scenario in Vila Real.*

In this context, the driver completes a delivery and receives, through the platform developed by the A-MoVeR team [35], two new delivery requests (represented in the figure by START). The first comes from a user living on *Dr. José Figueiredo* street, who requested some essential food items (represented in the figure by 2T), and the second from a resident on *Cidade de Portimão* street, who requested the delivery of prescribed medication (represented in the figure by FINISH). In a real scenario, this information would be displayed to the driver through the interface integrated into the motorcycle, which would automatically generate a route with pickup and delivery points, optimizing the path dynamically and efficiently. In this case, the stop points are the supermarket (represented in the figure by 1T), located on University Avenue, and the Portugal Pharmacy (represented in the figure by 3T), located at the Nervir roundabout. Based on the generated route, the driver would collect the products at the respective locations and deliver them to their destinations.

This process is simulated in the virtual environment through the navigation of the autonomous motorcycle, which travels through this urban and logistical scenario[1]. In this context, the trained model demonstrated the ability to follow the generated route, correctly passing through the pickup and delivery points and adapting its navigation to the city's geometry and task conditions. During navigation, 68 checkpoints were generated to assist the agent: 5 up to the first target, 22 up to the second, 28 up to the third—where a greater distance was observed—and finally 13 up to the fourth and last delivery point. It is also important to note that the agent traversed practically the entire city from one end to the other, completing the navigation in 2 minutes and 51 seconds, maintaining a constant and appropriate speed according to the route characteristics and the situations encountered along the path. This simulation verified that the agent successfully met the defined objectives, showing stable behavior, consistent responses to environmental variations, and a trajectory aligned with the expected plan.

The agent proved capable of achieving the objectives successfully, maintaining a stable and coherent trajectory throughout the route. Even when facing small variations in terrain or direction, it maintained consistent and appropriate decisions, without navigation failures or significant deviations. The simulation thus confirmed that the behavior learned by the agent showed signs of generalization within the simulated environment to a realistic scenario aligned with the defined requirements.

6. Conclusion and Future Work

This work demonstrated the integration of Deep Reinforcement Learning (DRL) techniques with deterministic path planning methods for autonomous navigation in realistic urban environments. Using a virtual representation of the city of Vila Real, an intelligent agent was trained to control a physics-based electric motorcycle inspired by the A-MoVeR project model. The agent's navigation relied primarily on the PPO algorithm, complemented by an auxiliary A* pathfinding system that generated a sequence of checkpoints to structure the learning process.

Although the PPO agent achieved consistent autonomous navigation, some limitations persisted, mainly related to the high computational cost of training and the manual tuning of hyperparameters. In particular, SAC required substantially longer training times, and Configuration 2 reached 681.5 hours, which is not acceptable for this type of development scenario. While the intended workflow is to train offline and deploy only a lightweight policy for real-time inference, such prolonged training durations remain a practical limitation and must be reduced to improve real-world applicability. Future research should explore automated hyperparameter optimization techniques and hybrid architectures that combine DRL with differentiable planning layers to improve efficiency and robustness, including evaluation of the learned policy on different urban maps and under varied conditions to assess cross-layout generalization and transferability.

Further extensions include the introduction of dynamic obstacles and multi-agent scenarios, allowing the application of game-theoretic formulations to model cooperative and competitive interactions. This would require extending the reward function with safety/near-collision terms and augmenting the checkpoint planning with online replanning or a local avoidance layer to account for moving agents. Additionally, alternative path-planning strategies and heuristics could be evaluated for adaptive route planning in more dynamic and geometrically complex environments, including approaches such as Ant Colony Optimization (ACO) or fuzzy-logic–based methods. Moreover, future work should investigate training-efficiency improvements to reduce the computational burden observed in realistic urban simulations.

[1]Demonstration video available at: https://youtu.be/XVgXq2MMxfM (restricted access via direct link)

Overall, the results confirm the mathematical and practical feasibility of combining DRL and A* pathfinding in realistic urban simulations, providing a solid foundation for future research on autonomous mobility systems.

Acknowledgments: The study was developed under the project A-MoVeR – "Mobilizing Agenda for the Development of Products and Systems towards an Intelligent and Green Mobility", operation n.º 02/C05-i01.01/2022.PC646908627-00000069, approved under the terms of the call n.º 02/C05-i01/2022 – Mobilizing Agendas for Business Innovation, financed by European funds provided to Portugal by the Recovery and Resilience Plan (RRP), in the scope of the European Recovery and Resilience Facility (RRF), framed in the Next Generation UE, for the period from 2021 -2026

References

[1] Y. Zhu, W. Z. Wan Hasan, H. R. Harun Ramli, N. M. H. Norsahperi, M. S. Mohd Kassim, Y. Yao: *Deep Reinforcement Learning of Mobile Robot Navigation in Dynamic Environment: A Review*, Sensors 25 (2025) 3394. DOI: 10.3390/s25113394.

[2] M. A. Alsuwaiket: *Optimizing Autonomous Vehicle Navigation Through Reinforcement Learning in Dynamic Urban Environments*, World Electric Vehicle Journal 16 (2025) 472. DOI: 10.3390/wevj16080472.

[3] B. Anderson, E. Sutherland: *Collective action for responsible AI in health*, OECD Artificial Intelligence Papers, No. 10, OECD Publishing, Paris (2024). DOI: 10.1787/f2050177-en.

[4] P. Costa, J. Ferdiansyah, H. D. Ariessanti: *Integrating Artificial Intelligence for Autonomous Navigation in Robotics*, International Transactions on Artificial Intelligence 3(1) (2024) Article 657. DOI: 10.33050/italic.v3i1.657.

[5] IBM: *What is machine learning?*, Available at: https://www.ibm.com/think/topics/machine-learning. Accessed: 13 Oct. 2025.

[6] J. Shuford: *Deep Reinforcement Learning: Unleashing the Power of AI in Decision-Making*, Journal of Artificial Intelligence General Science (JAIGS) 1 (2024) Article 36. DOI: 10.60087/jaigs.v1i1.36.

[7] G. Penelas, L. Barbosa, A. Reis, J. Barroso, T. Pinto: *Machine Learning for Decision Support and Automation in Games: A Study on Vehicle Optimal Path*, Algorithms 18 (2025) 106. DOI: 10.3390/a18020106.

[8] K. Hu, M. Li, Z. Song, K. Xu, Q. Xia, N. Sun, P. Zhou, M. Xia: *A review of research on reinforcement learning algorithms for multi-agents*, Neurocomputing 599 (2024) 128068. DOI: 10.1016/j.neucom.2024.128068.

[9] C. Banerjee, K. Nguyen, C. Fookes, M. Raissi: *A Survey on Physics Informed Reinforcement Learning: Review and Open Problems*, Expert Systems with Applications 287 (2025) 128166. DOI: 10.1016/j.eswa.2025.128166.

[10] H. Mashayekhi, M. Nazari, F. Jafarinejad, N. Meskin: *Deep Reinforcement Learning-Based Control of Chemo-Drug Dose in Cancer Treatment*, Computer Methods and Programs in Biomedicine 243 (2024) 107884. DOI: 10.1016/j.cmpb.2023.107884.

[11] C. Janiesch, P. Zschech, K. Heinrich: *Machine learning and deep learning*, Electronic Markets 31 (2021) 685–695. DOI: 10.1007/s12525-021-00475-2.

[12] I. H. Sarker: *Machine Learning: Algorithms, Real-World Applications and Research Directions*, SN Computer Science 2 (2021) 160. DOI: 10.1007/s42979-021-00592-x.

[13] C. M. Bishop, H. Bishop: *Deep Learning: Foundations and Concepts*, Springer, Cham, 1st edn. (2024). DOI: 10.1007/978-3-031-45468-4.

[14] N. Sharma, R. Sharma, N. Jindal: *Machine Learning and Deep Learning Applications – A Vision*, Global Transitions Proceedings 2(1) (2021) 24–28. DOI: 10.1016/j.gltp.2021.01.004.

[15] S. E. Li: *Deep Reinforcement Learning*, in Reinforcement Learning for Sequential Decision and Optimal Control, Springer, Singapore (2023). DOI: 10.1007/978-981-19-7784-8_10.

[16] P. Petersen, J. Zech: *Mathematical theory of deep learning*, arXiv: 2407.18384 (2025).

[17] K. Souchleris, G. K. Sidiropoulos, G. A. Papakostas: *Reinforcement Learning in Game Industry — Review, Prospects and Challenges*, Applied Sciences 13 (2023) 2443. DOI: 10.3390/app13042443.

[18] A. Sebastianelli, M. Tipaldi, S. L. Ullo, L. Glielmo: *A Deep Q-Learning based approach applied to the Snake game*, Proc. 29th Mediterranean Conference on Control and Automation (MED), Puglia, Italy (2021) 348–353. DOI: 10.1109/MED51440.2021.9480232.

[19] T. Li, X. Wan, M. M. Özbek: *AgentFuzz: Fuzzing for Deep Reinforcement Learning Systems*, Proc. IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Charlotte, NC, USA (2022) 110–113. DOI: 10.1109/ISSREW55968.2022.00049.

[20] Unity Technologies: *ML-Agents Toolkit Documentation, Version 4.0*, Available at: https://docs.unity3d.com/Packages/com.unity.ml-agents@4.0/manual/index.html. Accessed: 13 Oct. 2025.

[21] M. Lukas, I. Tomicic, A. Bernik: *Anticheat System Based on Reinforcement Learning Agents in Unity*, Information 13 (2022) 173. DOI: 10.3390/info13040173.

[22] A. Alagha, S. Singh, R. Mizouni, J. Bentahar, H. Otrok: *Target localization using Multi-Agent Deep Reinforcement Learning with Proximal Policy Optimization*, Future Generation Computer Systems 136 (2022) 342–357. DOI: 10.1016/j.future.2022.06.015.

[23] X. Liu, A. Fotouhi, D. Auger: *Application of advanced tree search and proximal policy optimization on Formula-E race strategy development*, Expert Systems with Applications 197 (2022) 116718. DOI: 10.1016/j.eswa.2022.116718.

[24] OpenAI: *Proximal Policy Optimization (PPO) — Spinning Up Documentation*, Available at: https://spinningup.openai.com/en/latest/algorithms/ppo.html. Accessed: 13 Oct. 2025.

[25] Unity Technologies: *ML-Agents Overview*, Available at: https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/. Accessed: 13 Oct. 2025.

[26] C.-C. Wong, S.-Y. Chien, H.-M. Feng, H. Aoyama: *Motion Planning for Dual-Arm Robot Based on Soft Actor-Critic*, IEEE Access 9 (2021) 26871–26885. DOI: 10.1109/ACCESS.2021.3056903.

[27] A. Kathirgamanathan, E. Mangina, D. P. Finn: *Development of a Soft Actor Critic deep reinforcement learning approach for harnessing energy flexibility in a large office building*, Energy and AI 5 (2021) 100101. DOI: 10.1016/j.egyai.2021.100101.

[28] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine: *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, arXiv: 1801.01290 (2018).

[29] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, S. Levine: *Soft Actor-Critic Algorithms and Applications*, arXiv: 1812.05905 (2019).

[30] D. Xu, Y. Cui, J. Ye, S. W. Cha, A. Li, C. Zheng: *A soft actor-critic-based energy management strategy for electric vehicles with hybrid energy storage systems*, Journal of Power Sources 524 (2022) 231099. DOI: 10.1016/j.jpowsour.2022.231099.

[31] OpenAI: *Soft Actor-Critic (SAC) — Spinning Up Documentation*, Available at: https://spinningup.openai.com/en/latest/algorithms/sac.html. Accessed: 13 Oct. 2025.

[32] S. R. Lawande, G. Jasmine, J. Anbarasi, L. I. Izhar: *A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games*, Applied Sciences 12 (2022) 5499. DOI: 10.3390/app12115499.

[33] S. L. Pardede, F. R. Athallah, Y. N. Huda, F. D. Zain: *A Review of Pathfinding in Game Development*, Journal of Computer Engineering (CEPAT) 1(01) (2022) 47–56. DOI: 10.25124/cepat.v1i01.4863.

[34] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, E. Gunawan: *A Systematic Literature Review of A* Pathfinding*, Procedia Computer Science 179 (2021) 507–514. DOI: 10.1016/j.procs.2021.01.034.

[35] A. Vigário, et al.: *Information and Management Platform for Green, Sustainable and Intelligent Mobility*, in: R. Mehmood, et al. (Eds.), Distributed Computing and Artificial Intelligence, Special Sessions I, 21st International Conference (DCAI 2024), Lecture Notes in Networks and Systems, vol. 1198, Springer, Cham, 2025. DOI: 10.1007/978-3-031-76459-2_24.