Navigation in Non-Static Environments with Autonomous Drones: A Kalman Filter Reinforcement Learning Approach

DOI: 10.13009/EUCASS2023-326
Aerospace Europe Conference 2023 – 10TH EUCASS – 9TH CEAS

Navigation in Non-Static Environments with Autonomous Drones: A Kalman Filter Reinforcement Learning Approach

Francesco Marino †, Afshin Zeinaddini Meymand* and Giorgio Guglieri**
** Politecnico di Torino*
10129 Turin, Italy
francesco_marino@polito.it – a.zeinaddini@studenti.polito.it – giorgio.guglieri@polito.it
† Corresponding Author

Abstract

Autonomous drone systems have grown in various industries, but their effectiveness in dynamic environments remains challenging. This study addresses the issues faced in path planning and state estimation for drones operating in non-static environments. To tackle these challenges, a solution combining Kalman Filters and Advanced Reinforcement Learning (RL) Algorithms is proposed. Three RL algorithms are compared to evaluate their performance. Combining the Kalman Filter and RL techniques improves path planning and decision-making, resulting in successful navigation in simulated dynamic scenarios. The approach is designed for continuous state-action environments.

1. Introduction

Interest in autonomous vehicles covering aerial, terrestrial, and underwater areas has dramatically increased due to their prospective ability to transform many industries. The development of reliable path-planning methods is crucial for task optimization and the enhancement of the versatility and efficiency of these vehicles. Autonomous drones show substantial potential, able to conduct intricate tasks across a wide array of environments. However, the constantly changing nature of these environments introduces distinct obstacles, making path planning for drones an intriguing field of study. Autonomous drones are utilized in various duties, from surveillance and delivery services to search and rescue missions. To carry out these tasks effectively, they require path planning optimization considering the immediate environment and the inherent dynamism of real-world settings. Several path planning methodologies have been investigated, such as minimum snap path planning and minimum time path planning. Despite their efficacy, these strategies are predominantly designed for static environments. Conversely, drones often operate in dynamic environments where objects and conditions can shift unpredictably. As a result, path planning for dynamic environments is inherently more intricate and demanding. To enable effective path planning and navigation, estimating the status of dynamic objects is made even more challenging by the cluttered nature of these environments. Existing research has started addressing these challenges.

Path planning and state estimation are central to the operations of autonomous drones. Path planning involves determining the optimal route a drone should take to accomplish its task, which can greatly differ depending on the task and environment specifics. A common method involves defining specific waypoints the drone must reach and then determining the most efficient route. State estimation complements path planning by supplying vital information about the current and future states of the environment. This information facilitates informed decisions about the drone's path. State estimation techniques can be employed to identify significant features in the environment, such as the center of goal objects. Predicting the future location of these objects, especially in dynamic environments, continues to be a field-wide challenge.

In this study, we put forth a novel methodology to tackle the challenges of state estimation and path planning in dynamic environments. Our strategy harnesses the capabilities of Kalman filter algorithms for estimating and forecasting the states of dynamic objects in the environment. More specifically, we use these algorithms to estimate and forecast the positions of waypoints, which are defined as the centers of gates in our context. The Kalman filter facilitates estimation of the current state of these waypoints, and their future states over a variable time horizon, contingent on the drone's distance from each gate.

To complement state estimation, we also suggest the employment of advanced actor-critic reinforcement learning (RL) algorithms for optimal path planning. We implement three RL algorithms in a continuous action-state environment to discern the optimal path through the predicted waypoints. The path's optimality is defined based on a reward function provided to the RL algorithms. The combination of state estimation and path planning techniques is intended to

DOI: 10.13009/EUCASS2023-326

significantly enhance the performance of autonomous drones in dynamic environments. In this paper, our primary objective is to explore the effectiveness of a combined approach that uses the Kalman filter to predict future positions of gates over changing time horizons and different Reinforcement Learning (RL) algorithms for optimal path planning in dynamic environments. We aim to assess and compare the performance of three specific RL algorithms under varying environmental conditions, with a particular emphasis on dynamic settings.

1.1 Related works

The landscape of available training environments includes several three-dimensional reinforcement learning simulation environments, such as Gym [2] and AirSim [16], each with their challenges. For instance, despite AirSim's realistic and detailed nature, it demands a high-performance system due to its computational intensity. On the other hand, Gym, which is less computationally intensive, lacks environments specifically dedicated to drone path planning, with some environments supporting only discrete action spaces. Our research aims to surmount these hurdles by formulating a customized continuous action-state environment using the Python Pygame library. This tailor-made environment reduces the computational demand compared to AirSim and is also precisely suited to our specific task. We have integrated a field-of-view feature commonly absent in Pygame environments but vital for authentic drone perception to enhance its realism.

Various studies have explored autonomous drone path planning, utilizing the Potential Field Method (PFM), Rapidly exploring Random Tree (RRT), and the Voronoi diagram (VD). Each of these techniques presents its limitations. For example, the Potential Field Method is susceptible to local minima, causing a deadlock where the drone gets trapped in a position surrounded by obstacles [7, 19]. RRT, although efficient in navigating high-dimensional spaces, may need help finding the optimal path, and its performance can be compromised in densely cluttered environments [8, 12]. Voronoi Diagrams, while ensuring a safe distance from obstacles, do not necessarily yield the shortest or most efficient path, and their generation can be computationally intense, particularly in complex or dynamic environments [1, 13].

There has also been considerable effort in applying reinforcement learning to path planning [9, 15, 17]. The Song Et al. study didn't test the algorithm's performance in dynamic environments. It depended on Vicon cameras for gate state prediction, which may not be universally applicable in real-world contexts [17]. Another study applied three reinforcement learning algorithms in a static environment [9].

Our research strives to bridge these gaps by leveraging Kalman Filters for state estimation and prediction [6]. We aim to design a reinforcement learning environment that meets specific needs, such as supporting a continuous action state environment, providing specific state elements, and defining the agent's reward function based on the Kalman filter prediction. Additionally, we incorporate the field of view of real cameras as boundary conditions for the environment to enhance the realism and practicality of our approach.

2. Methodology

We employed a comprehensive methodology with a 3D simulation environment to address the issues identified in the abstract. This environment was used to emulate the dynamic conditions that drones may encounter in real-world scenarios. This 3D simulation allowed us to explore all possible outcomes under various scenarios and conditions, thereby providing a robust and extensive platform for the training and testing of our autonomous drone systems.

Our methodology also involved the use and comparison of three different reinforcement learning (RL) algorithms: Deep Deterministic Policy Gradient (DDPG), Soft-Actor Critic (SAC), and Proximal Policy Optimization (PPO). These algorithms were trained to focus on Path planning and state estimation, two critical factors for successful drone operation in dynamic environments. The three algorithms were evaluated on the same scenarios within the 3D simulation to ensure a fair comparison. The performance of each algorithm was then analyzed and compared, with particular attention paid to their respective ability to navigate successfully in the dynamic simulated scenarios. Moreover, the Kalman Filter, a well-known technique for state estimation, was integrated with the RL techniques. The goal was to enhance the drones' decision-making process and path planning. Combining these two techniques, we aimed to create an autonomous drone system that could adapt and respond effectively to non-static environments.

2.1 Simulation Environment

The simulation framework is set within a three-dimensional drone navigation game where the drone's task is maneuvering through a pair of gates. This framework was coded from scratch to allow the maximum customization possible. It is encapsulated within a class, which establishes several parameters, including the drone's initial location, the initial positions of the gates, the drone's field of view (FoV), and the overall dimensions of the simulation environment (represented as width and height, length). The system comprises four unique scenarios defined in the continuous state-action space, each presenting differing behaviors for the drone and the gates.

- The first scenario involves a static drone starting point and immobile gates.

DOI: 10.13009/EUCASS2023-326

- The second scenario maintains a fixed drone starting point but introduces dynamic moving gates.
- The third scenario switches to a randomized drone starting point but reverts to static gates.
- The fourth scenario features both a randomized drone starting point and dynamic gates.

Algorithm 1 Autonomous Drone Navigation

```
Initialize drone position ▷ Sets the starting point of the drone in the environment
Initialize Gates position ▷ Sets the position of the Gates in the environment
Initialize Gates Speed Type and their Values ▷ Sets the speed to be static or variational
while drone not at goal do
    current_state ← get_drone_environment_state() ▷ Gathers current drone and gates states
    approximated_time_horizon ← time_horizon(current_state) ▷ Estimates the time until the drone reaches its goal based on the current state
    kalman_prediction ← kalman_filter_prediction(current_state, approximated_time_horizon) ▷ Predicts the future state of the drone using a Kalman filter
    predicted_state ← kalman_filter_update(current_state, kalman_prediction) ▷ Updates the current state with the Kalman filter prediction
    RL_state ← predicted_state ▷ The state used for reinforcement learning is set to the predicted state
    reward ← calculate_reward(current_state) ▷ Calculates the reward based on the current state
    if (RL_algorithm = DDPG) then
        action ← DDPG(RL_state, reward) ▷ Selects the next action using the DDPG algorithm
    else if (RL_algorithm = SAC) then
        action ← SAC(RL_state, reward) ▷ Selects the next action using the SAC algorithm
    else if (RL_algorithm = PPO) then
        action ← PPO(RL_state, reward) ▷ Selects the next action using the PPO algorithm
    end if
    navigate_drone(action) ▷ Moves the drone based on the selected action
    if drone collides then
        reset_environment() ▷ Resets the environment to the initial states in case of a collision
        collision_penalty() ▷ A negative reward (penalty) is given to the drone for colliding
    end if
    if drone reaches goal then
        success_reward() ▷ A positive reward is given to the drone for reaching the goal
    end if
end while
```

Figure 1 Pseudocode of the algorithm implemented.

In a tri-dimensional context, the study centers on continuous action states. When dealing with stationary gates, the state is expressed as (x, y, z), that represent the relative coordinates of the drone with respect to the gate along the corresponding x, y and z axes. This model provides enough data to steer the drone in a 3D setting with unchanging gate positions. Nevertheless, in scenarios featuring movable gates, the state definition requires to be expanded to encompass supplementary details about the relative velocities in the y and z axes. Consequently, the state is presented as $(x, y, z, \dot{y}, \dot{z})$. The added elements $(\dot{y}, \dot{z})$, account for the relative speed in the y and z directions, respectively, empowering the drone to respond more effectively to shifting gates. By incorporating both immobile and moving gates in a three-dimensional space, the analysis becomes more extensive and versatile, supporting the operation of the drone into various circumstances. This strategy enables a more resilient understanding of drone navigation and control, which can be advantageous in use-cases like drone racing, search and rescue operations, and environmental surveillance, where the capability to navigate through intricate environments is critical.

DOI: 10.13009/EUCASS2023-326

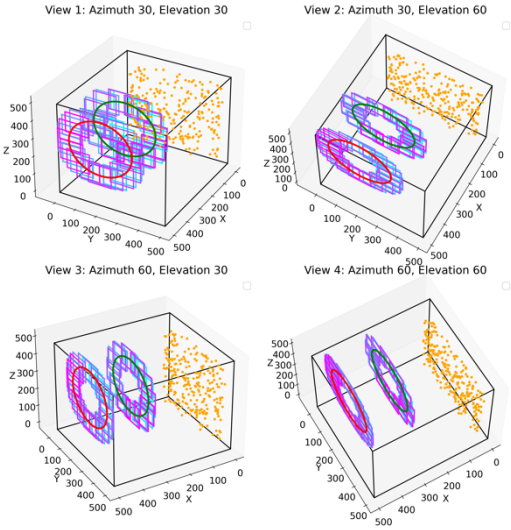


Figure 2 View of the 3D Simulation Environment. It is possible to observe some samples of the initial positions of the drone, represented by the small circles and the trajectories of the first and the second gates, respectively, in green and red.

The action space, encompassing both stationary and movable gates, involves the magnitude of displacement along the x, y and z axes. Each of these magnitudes lies within the interval of $[-1, 1]$, signifying the shift in the respective direction. By balancing the total of all movements, the overall displacement's magnitude at each step equals one unit. Within this action space, the drone can traverse in any direction within the tri-dimensional space by blending the appropriate magnitudes for each axis. This adaptable methodology enables the drone to navigate and adjust to various scenarios, irrespective of whether they involve stationary or movable gates. Maintaining a normalized range for the magnitudes ensures uniformity and comparability across different movement directions and environments. This depiction of the action space is apt for drone navigation and control applications, as it allows the system to react effectively to a broad range of situations.

With each time increment, the reward function has been crafted to motivate the drone to gravitate toward the gate's center. A generous positive reward is allocated when the drone progresses closer to the gate's center, while a negative reward is doled out if the drone strays further from the center. Additionally, suppose the drone collides with any boundaries, including the environment's limits and areas where the gate centers are outside the camera's field of view. In that case, the reward will be a notably negative value at that step. The formula for the reward function is presented as follows:

$$Reward(t) = \begin{cases} -30 \; in \; case \; of \; collitions \\ +30 \; if \; complete \; the \; episode \\ d_{t-1}^{rel} = (s_{t-1}, a_{t-1}) - d_t^{rel}(s_t, a_t) \; otherwise \end{cases} \quad (1)$$

Where $Reward(t)$ represents the reward at time t, $d_t^{rel}(s_t, a_t)$ is the relative distance at time t given the state s and the action a and (s_{t-1}, a_{t-1}) denotes the state respectively and the action at the previous time step $(t-1)$. This reward function incentivizes the drone's navigation system to streamline its path toward the gate's center, enhancing its overall performance across various environments.

2.2 Single-Instruction Multi-data Kalman

The Kalman filter is a mathematical algorithm that provides an efficient computational solution to estimate the state of a process in a way that minimizes the mean of the squared error. It is used in a wide range of engineering [3] and data analysis applications to filter out noise and provide accurate data about the state of a system over time.

The Kalman filter becomes even more powerful when integrated with the SIMD concept. The SIMD (Single Instruction, Multiple Data) Kalman filter leverages the power of modern processors to perform multiple operations simultaneously. The SIMD Kalman filter significantly increases the data processing speed by packing several data items into one register and operating on all of them simultaneously. This approach is particularly beneficial in scenarios where operations can be naturally parallelized, such as high-energy physics experiments where large volumes of data need to be processed quickly. Using the SIMD Kalman filter, these operations can be performed much faster, leading to more efficient data processing and analysis.

DOI: 10.13009/EUCASS2023-326

This makes it especially useful when large volumes of data need to be processed quickly.
The SIMD (Single Instruction, Multiple Data) Kalman filter operates in several steps to leverage the power of modern processors for efficient data processing. Initialization: The algorithm begins with an initial approximation of the system's state. This approximation is represented as a vector, and its covariance matrix is set to a large positive number, indicating a high level of uncertainty. Prediction: The algorithm predicts the state of the system at the next time step. This prediction is based on the current state and the system's dynamics, represented by the prediction matrix. The prediction step produces an estimated state vector and its associated covariance matrix, which represents the uncertainty of the prediction. Process Noise: This step accounts for the probabilistic deviations in the system's state due to process noise. The estimated state vector and covariance matrix are updated to reflect this noise. Filtration: The state vector is updated with the new measurement to get the optimal estimate of the system's state and covariance matrix. The Kalman gain matrix, which determines how much weight to give to the new measurement based on its uncertainty, is calculated and applied. The total deviation of the obtained estimation from the measurements is also calculated. The filter's complete mathematical formulations are reported in [4].

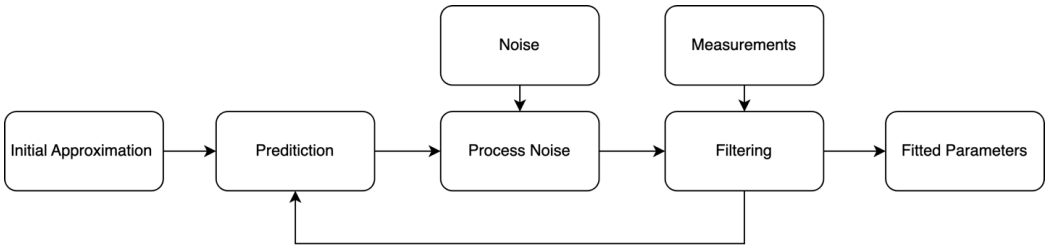


Figure 3 Block diagram representation of the Kalman Filter processing steps.

In the study discussed in this paper, the SIMD version of the Kalman filter has been employed to filter the data related to the position of the gates. This application enabled us to acquire a precise estimate of the gate's motion after a limited number of observations. The downstream algorithm subsequently utilized this information to approximate the gate's position at a specific time point. In the prediction step of the Kalman filter, an estimate of the new position of the gate is made based on the previous state. This step incorporates the state transition model and the control input, if any, to predict the current state of the gate. This prediction forms the prior estimate for the state of the system. The observations from the environment then come into play during the update step. These observations adjust the prediction based on what was measured, which can be particularly beneficial when there is uncertainty or noise in the system. In this case, the measurements pertain to the gate's movement. The difference between the prediction from the prior state and the observation from the environment, also known as the innovation or residual, is then used to update the state estimate. This updated estimate is a weighted average of the prior estimate and the current measurement, with more weight given to estimates with more certainty. Through several iterations, the above process yields an accurate estimate of the gate's movement, effectively tracking the gate's position over time. This ability to use observations to predict and correct the position estimate makes the Kalman filter an excellent tool for dealing with the dynamics of moving objects like the gate in our study.

2.3 Deep Reinforcement Algorithms Trained

Reinforcement Learning (RL) in robotics is an application of machine learning where an agent learns to make decisions by interacting with its environment. The core idea is to have robots learn how to perform tasks by trying out different actions and learning from the outcomes rather than being explicitly programmed to carry out specific tasks.
In a typical RL setup, the robot, or the agent, observes the state of the environment and takes actions based on these observations. The environment then provides feedback in the form of rewards or penalties. The agent's goal is to learn a policy, which is a mapping from states to actions, that maximizes the cumulative reward over time.
In this work, two different types of RL algorithms were implemented, Actor-Critic and Policy-based. The Actor-Critic method is a popular approach used in RL, combining the benefits of value-based and policy-based methods. This approach divides the RL agent into two parts: an actor and a critic.
The "actor" is responsible for choosing actions given the current state of the environment. In other words, the actor is the component that decides how the agent should behave (its policy). The "critic", on the other hand, estimates the value of taking these actions, typically using something like the Bellman equation. In essence, the critic evaluates the performance of the actor's chosen policy. Policy-based Deep Learning algorithms, such as pure policy-gradient methods, offer a unique approach to RL by directly optimizing stochastic policies. Central to these methods are stochastic policies, which define a probability distribution over actions for each state. This characteristic promotes exploration in the action space, allowing the agent to discover potentially superior actions over time. By continuously

DOI: 10.13009/EUCASS2023-326

updating and refining the policy based on observed outcomes, pure policy-gradient methods offer a direct and efficient means of learning optimal strategies for agents operating in complex and dynamic environments.
One advantage of pure policy-gradient methods is their ability to bypass the need for intermediate value estimations. While accurately determining the values of states in reinforcement learning can be a complex task, employing a rough approximation can prove beneficial. By incorporating approximate state values, the variance of the policy-gradient objective can be reduced, leading to enhanced stability and convergence of the learning process.
Deterministic Policy Gradient (DDPG) and Soft-Actor Critic(SAC) are designed to handle continuous action spaces, making them well-suited for tasks such as drone navigation, where the range of possible actions is not discrete but continuous. also, these algorithms are off-policy, meaning they learn from past experiences stored in a replay buffer. This is beneficial as it allows the algorithms to learn from a broad range of experiences and policies, and not just the policy that is currently being executed. This can lead to more robust learning and better performance. SAC introduces an entropy term into the reward function to encourage exploration, which is crucial in dynamic environments where the agent must constantly explore new strategies to adapt to changing conditions. On the other hand, Proximal Policy Optimization (PPO) introduces a novel approach to policy optimization, including a clipping mechanism to prevent overly large policy updates. By comparing these three algorithms, you can highlight the strengths and weaknesses of different techniques in RL and provide insight into what techniques are most effective for drone navigation in dynamic environments.

2.3.1 Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) [22] algorithm builds upon the Deep Q-Network (DQN) concepts yet is tailored for continuous action spaces. It uses a replay buffer to store and randomly sample experiences, thus allowing off-policy training of an action-value function and employing target networks for added stability. However, DDPG distinguishes itself from DQN by utilizing an approximate deterministic policy that symbolizes the best possible action, qualifying it as a deterministic policy-gradient technique aptly suited for continuous action spaces. Like DQN, DDPG gathers experiences online and stows them in a replay buffer. These experiences, gathered in mini-batches, are leveraged to compute a bootstrapped Temporal Difference (TD) target and refine a Q-function. Contrary to DQN's approach, DDPG adopts a target deterministic policy function trained to mimic the greedy action instead of invoking the $argmax$ operation on the target Q-function. This deterministic policy function bears multiple advantages. It circumvents the need for an $argmax$ operation, which can be computationally demanding in continuous action spaces. Furthermore, it directly approximates the best action, potentially resulting in expedited convergence. It can also mitigate the exploration-exploitation dilemma by immediately targeting the optimal action for each state.
Within DDPG, a policy network is utilized to propose the optimal action for a given state. This network must be differentiable concerning the action, necessitating the action to be continuous for efficient gradient-based learning. The optimization goal involves leveraging the expected Q-value derived from the policy network. The agent's objective is to ascertain the action that maximizes this value. In our rendition of DDPG, we streamline the architecture and training process by using online networks for policy (action selection) and value function (action evaluation), in place of target networks.

2.3.2 Soft-Actor Critic (SAC)

The Soft Actor-Critic (SAC) [21] algorithm is an off-policy technique in RL that combines aspects from deterministic and stochastic policy frameworks. Much like the Deep Deterministic Policy Gradient (DDPG), SAC operates off-policy, leveraging a replay buffer to learn from various experiences generated by varying behavioral policies. However, a key difference lies in SAC's use of a stochastic policy, which promotes exploration, as opposed to the deterministic policy employed by DDPG. A defining characteristic of SAC is incorporating an entropy term into the Bellman equations, effectively folding the entropy of the stochastic policy straight into the value function. This process of concurrently maximizing both expected cumulative reward and expected cumulative entropy inherently foster varied behaviors while ensuring the agent consistently seeks to maximize the expected return. By considering entropy, SAC promotes the agent to explore a wide spectrum of potential actions and strategies, enhancing its versatility and proficiency in tackling complex environments.
This expectation spans the reward, the subsequent state, and the subsequent action. It encompasses the reward and the discounted value of the upcoming state-action pair, in addition to the entropy of the policy at the forthcoming state.

2.3.3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization [20] represents a notable breakthrough in reinforcement learning, presenting a surrogate objective function that facilitates several gradient steps utilizing the same mini-batch of experiences. This starkly contrasts conventional on-policy approaches like Advantage Actor-Critic (A2C) [], which necessitate the disposal of experience samples following a single optimization maneuver. The surrogate objective function evaluates the new and

DOI: 10.13009/EUCASS2023-326

old policy via a likelihood ratio, striving to maximize expected return while ensuring policy updates remain confined to a trust region.
An integral aspect of PPO is its clipped objective function, which regulates the extent of policy updates. This clipping device inhibits drastic alterations post each optimization step, guaranteeing cautious and measured updates. This methodology alleviates the issue of performance deterioration, a frequent occurrence in on-policy gradient methods. A prevalent issue with the typical policy gradient is its sensitivity to tiny changes in the parameter space, which can dramatically influence performance. This mismatch between changes in the parameter space and their resulting impact on performance necessitates using small learning rates in policy-gradient methods. Nevertheless, these methods may still exhibit high variance. Clipped PPO addresses this by restricting the objective, ensuring the policy change remains within defined bounds at each training step.
We can also extend this clipping strategy to the value function. This approach shares the same central idea: limit the changes in the parameter space such that the alterations in Q-values are controlled and don't exceed a certain threshold. As a result, this clipping approach ensures smooth variation in the aspects we are concerned with, regardless of whether changes in the parameter space are smooth or not.

3. Experiments and Results

In our research, we employ three main criteria to evaluate the performance of the reinforcement learning algorithms: cumulative score per episode, the count of episodes in which the agent accomplished the objective, and the duration per episode ($episode\ elapsed$). These criteria can be further divided into five metrics defined as follows:

1. Cumulative Score Per Episode: The cumulative score per episode directly measures the agent's performance in its assigned task. Superior scores suggest that the agent optimizes its actions to fulfill its objective.
2. Count of Episodes to Accomplish the Objective: This criterion assesses the agent's learning proficiency. An agent that reliably accomplishes the objective in fewer episodes proves to be quicker in resolving the task and adaptation to the environment.
3. Duration per Episode ($episode\ elapsed$): By evaluating the duration spent on each episode, we can gauge the agent's computational efficiency. This becomes crucial when comparing algorithms that garner similar scores, as computational efficiency could be the deciding factor in algorithm choice.
4. Average Reward per Action: The average reward per action offers a measure of the quality of the decisions made by the agent. An increased average reward per action indicates that the agent is making superior decisions, leading to higher rewards for each action. This becomes particularly noteworthy in scenarios where the agent has a limit on the number of actions it can execute in an episode or when the objective is to garner the highest reward in the least possible time or number of steps. This metric balances the speed (number of steps) and the quality (reward) of the actions, preventing strategies that excessively prioritize one aspect at the detriment of the other.
5. Success Rate: This represents the ratio of episodes where the agent accomplishes its objective. A higher success rate indicates that the agent can more reliably complete its task.

Finally, by multiplying Success Rate and Average Rewards together, you get a new metric that can be interpreted as follows:

- High Average Reward per Action, High Success Rate: If the product is high, it suggests that the algorithm is both proficient (it receives a high reward per action) and successful (it frequently accomplishes its goal). This is the optimal situation.
- Low Average Reward per Action, High Success Rate: It might imply that the algorithm frequently accomplishes its goal but could be more proficient (it receives a low reward per action).
- High Average Reward per Action, Low Success Rate: Conversely, a moderate product might suggest that the algorithm is proficient (it receives a high reward per action) but does not frequently accomplish its goal.
- Low Average Reward per Action, Low Success Rate: If the product is low, it suggests that the algorithm needs to be proficient and successful. This would imply that the algorithm's performance is subpar.

These metrics offer a comprehensive perspective of the effectiveness and efficiency of the reinforcement learning algorithms under study.

DOI: 10.13009/EUCASS2023-326

3.1 Results Obtained

In reviewing the outcomes of our experiments, several key findings were discernable. The performance of the reinforcement learning algorithms was primarily evaluated through metrics: cumulative score per episode, the number of successful episodes, and the episode duration. The cumulative score per episode served as a crucial indicator of the overall efficiency of the algorithms, offering insights into the ability of the agent to accumulate rewards over the course of an episode. The count of successful episodes measured the agent's reliability in achieving the objectives. Lastly, the duration per episode or *episode elapsed*, allowed us to gauge the speed of the agent's learning process, with shorter durations reflecting faster convergence to optimal behaviors. These criteria collectively provided a comprehensive understanding of the algorithms' capabilities in various aspects of reinforcement learning.

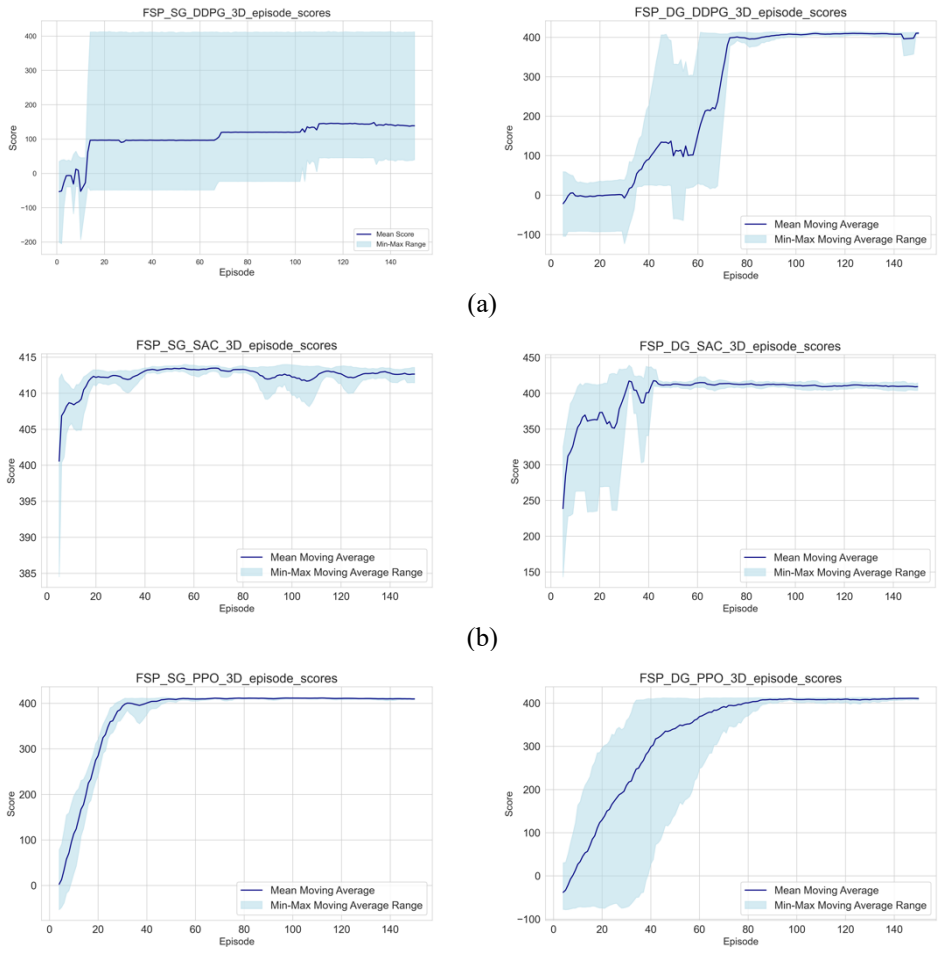


Figure 4 Overview of the results for the fixed starting point (FSP) the environment with static (SG) and dynamic (DG) gates and the three reinforcement learning algorithms: DDPG (a), SAC (b) and PPO (c).

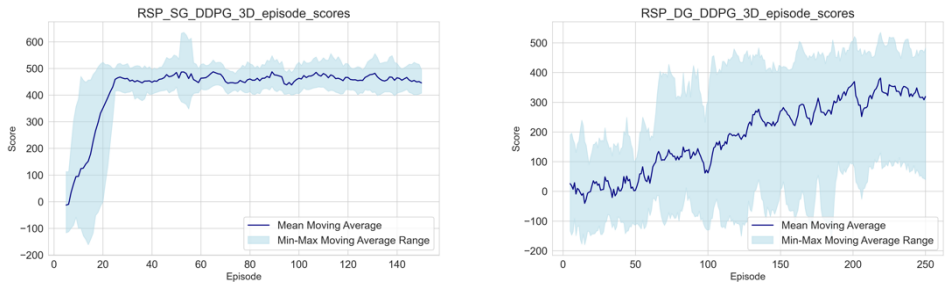
Figure (4) offers a comprehensive summary of the results attained using the three selected algorithms, all originating from a consistent starting point across all episodes. The Soft Actor-Critic (SAC) algorithm stands out for its rapid convergence to high rewards with a minimal episode count in both static (SG) and dynamic (DG) environments. However, despite this swift convergence, the SAC manifests some instability, even at high episode counts. On the other hand, the Proximal Policy Optimization (PPO) algorithm demonstrates a slower convergence rate but eventually reaches similar end-of-training rewards as SAC and exhibits less uncertainty in resolving the environment. The slower convergence of PPO is attributed to its inherent design that encourages more conservative policy updates. This approach, while leading to a slower learning rate, also contributes to the algorithm's superior stability, as it avoids drastic policy changes that could potentially disrupt the learning process and result in increased uncertainty.

DOI: 10.13009/EUCASS2023-326

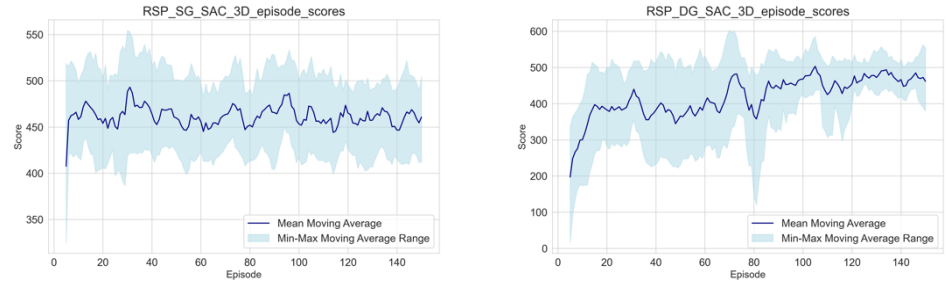
Table 1 Summary of the results obtained with the fixed starting point (FSP) environment.

Algorithm	Environment	Total Episodes	Episodes where Goal was Reached	Average Elapsed Time	Average Reward per Step	Success Rate (%)
FSP_SG_DDPG_3D	Static	1000	**432**	6,768	0,778	0,432
FSP_SG_SAC_3D		1000	990	**9,223**	0,709	**0,990**
FSP_SG_PPO_3D		984	886	0,640	**0,795**	0,900
FSP_DG_DDPG_3D	Dynamic	1000	**770**	10,801	**0,997**	0,770
FSP_DG_SAC_3D		1000	944	**20,647**	0,694	**0,944**
FSP_DG_PPO_3D		984	802	1,986	0,743	0,815

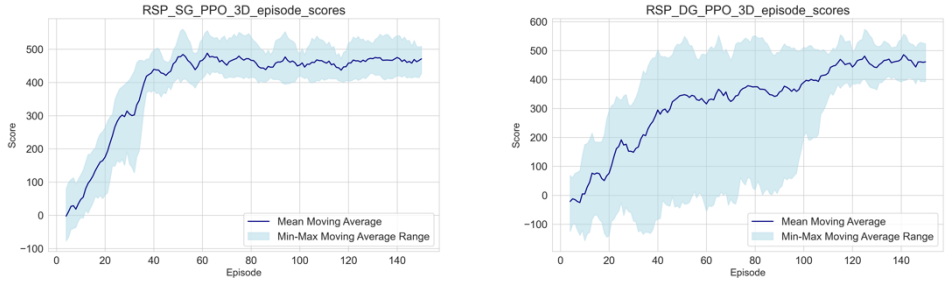
The results compiled in Figure (5) provide a detailed overview of the outcomes achieved using the three selected algorithms, each initiated from a random starting point defined at the beginning of each episode. The Soft Actor-Critic (SAC) algorithm is noteworthy due to its speedy convergence toward high rewards, requiring only a low number of episodes in both static (SG) and dynamic (DG) settings. Conversely, the Proximal Policy Optimization (PPO) algorithm displays a slower convergence rate but eventually attains comparable end-of-training rewards to SAC and shows less volatility when navigating the environment.



(a)



(b)



(c)

Figure 5 Overview of the results for the random starting point (RSP) the environment with static (SG) and dynamic (DG) gates and the three reinforcement learning algorithms: DDPG (a), SAC (b) and PPO (c).

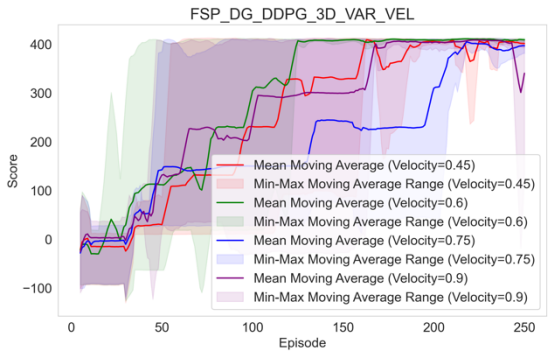
DOI: 10.13009/EUCASS2023-326

Table 2 Summary of the results obtained with the random starting point (RSP) environment.

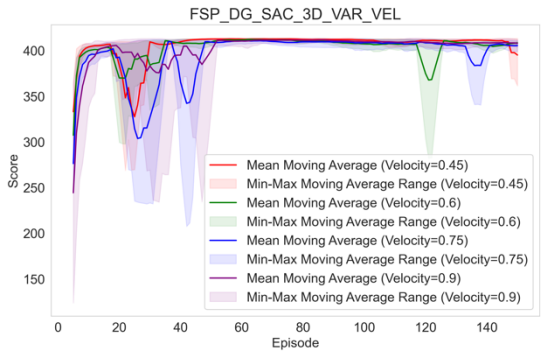
Algorithm	Environment	Total Episodes	Episodes where Goal was Reached	Average Elapsed Time	Average Reward per Step	Success Rate (%)
RSP_SG_DDPG_3D	Static	1000	941	5,475	**1,038**	0,941
RSP_SG_SAC_3D		1000	994	9,688	0,752	**0,994**
RSP_SG_PPO_3D		984	**822**	**0,721**	0,805	0,835
RSP_DG_DDPG_3D	Dynamic	1000	**414**	14,652	0,743	0,414
RSP_DG_SAC_3D		1000	759	34,967	0,542	**0,759**
RSP_DG_PPO_3D		984	726	**2,327**	**0,746**	0,738

3.2 Effects of Gate Movement Speed

In summarizing our research efforts, we scrutinized the performance of various reinforcement learning algorithms under the diverse dynamism of our environment. Specifically, we focused on the gate speeds' variability effect on the reward function. For the sake of this experiment, we parametrized the velocity, setting it to a low limit of 0.45 rad/s, which denoted the starting point, scaling up to a high end of 0.90 rad/s.
Our observations exposed those algorithms, such as DDPG (Deep Deterministic Policy Gradient) and SAC (Soft Actor-Critic), which appeared to be considerably swayed by the varying speeds of the gates. Consequently, a broad spectrum was evident in the range of rewards during the different trials, indicating high variability and inconsistency in the outcomes.

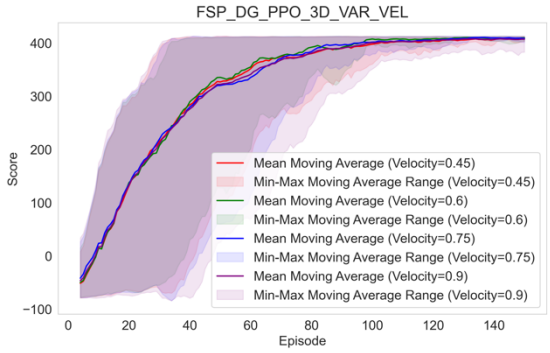


(a)



(b)

DOI: 10.13009/EUCASS2023-326



(c)

Figure 6 Performances of the algorithms under different conditions of gates speed (rad/s). DDPG (a), SAC (b), and PPO (c).

In stark contrast, the PPO (Proximal Policy Optimization) algorithm showcased a more robust performance. PPO produced a more stable learning process. It yielded a persistent decrement in the spread across multiple trials, which signifies a smoother, more predictable learning curve. This finding is consistent with what we noted for the static and dynamic gate simulations. Hence, it implies that PPO might be a more suitable choice when dealing with environments characterized by rapid, unpredictable changes. This reinforces PPO's status as an advantageous approach in reinforcement learning, particularly in highly dynamic environments.

4. Discussions and Conclusions

Our research findings offer valuable insights into the performance of Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO) algorithms in intricate environments. Dynamic environments pose significant challenges for robotics systems, primarily due to their unpredictability and high adaptability required for successful operation. These challenges stem from such environments constantly changing, with the robot needing to react and adapt in real-time to achieve its objective.

In particular, DDPG's deterministic policy approach showed limitations in these settings. While beneficial in directly approximating the optimal action and possibly hastening convergence, DDPG's exploration strategy could benefit from further refinement to enhance performance in more complex, evolving scenarios.

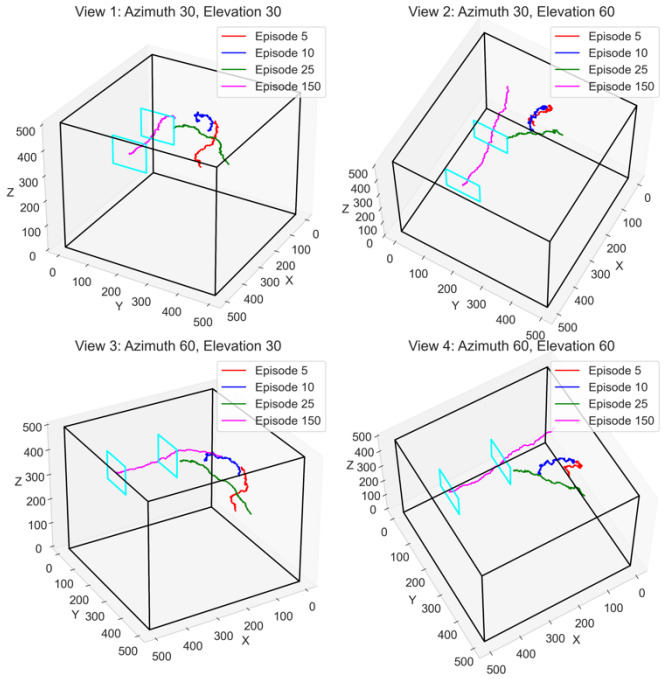


Figure 6 Trajectory obtained with the algorithm PPO in a dynamic environment for increasing episodes.

DOI: 10.13009/EUCASS2023-326

On the other hand, SAC displayed rapid learning, attributed to its unique integration of a stochastic policy and an entropy component within its objective function. By continually seeking to balance reward maximization and action diversification, SAC skillfully negotiates the exploration-exploitation trade-off, fostering speedy learning in complex environments.

While not the fastest to learn, PPO demonstrated robust and steady performance after adequately approximating the optimal policy and value functions. PPO's stability is derived from its clipped objective function, which regulates policy updates to avoid drastic changes that could lead to performance volatility.

We also leveraged the Kalman filter to accurately predict the gate's position in space, effectively managing the dynamic nature of the moving gate. The filter uses an iterative process of prediction and correction based on previous states and current measurements, demonstrating its prowess in handling dynamic motion tracking.

However, these findings are based on simulations, and as we move forward, we need to consider the next steps: transitioning from simulation to hardware testing. Despite the fidelity of our simulations, real-world scenarios can introduce unforeseen variables and challenges that need to be accounted for in the simulation environment. Issues such as sensor noise, mechanical failures, and real-world physics discrepancies may arise and can significantly impact the performance of the robotics systems and the algorithms controlling them.

To bridge this 'reality gap', future research will focus on hardware-in-the-loop testing, where the algorithms will be integrated with physical systems in controlled environments. These tests will provide valuable data on how these algorithms perform under real-world conditions and how they can be further optimized. Additionally, we anticipate refining our models to handle environmental changes better, incorporating more advanced perception capabilities, and increasing the robustness of the systems. We aim to enhance our robotics systems' practical applicability and performance in dynamic environments by carefully iterating our designs and algorithms based on this new data. Finally, a GitHub page[1] was created to support the visualization of what was obtained throughout the RL agents in this paper.

References

[1] Riyadarshi Bhattacharya and Marina L. Gavrilova. Roadmap-based path planning - using the Voronoi diagram for a clearance-based shortest path. IEEE Robotics & Automation Magazine, 15, 2008.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Shengyong Chen. Kalman filter for robot vision: A survey. IEEE Transactions on Industrial Electronics, 59:4409–4420, 2012.

[4] Sergey Gorbunov, Udo Kebschull, Ivan Kisel, Volker Lindenstruth, and Walter F. J. Müller. Fast simdized kalman filter based track fit. Comput. Phys. Commun., 178(5):374–383, 2008.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. pages 1861–1870, 2018.

[6] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[7] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In Proceedings. 1991 IEEE International Conference on Robotics and Automation, pages 1398–1404 vol.2, 1991.

[8] Steven M. LaValle. Rapidly-exploring random trees: a new tool for path planning. The annual research report, 1998.

[9] Myoung Hoon Lee and Jun Moon. Deep reinforcement learning-based uav navigation and control: A soft actor-critic with hindsight experience replay approach. arXiv preprint arXiv:2106.01016, 2021.

[10] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. CoRR, 09 2015.

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. pages 1928–1937, 2016.

[12] Ram Noreen, Amna Khan, and Zulfiqar Habib. Optimal path planning using rrt* based approaches: A survey and future directions. International Journal of Advanced Computer Science and Applications, 7(11), 2016.

[13] Mitsuhiro Ozaki, Jagannath Aryal, and Paul Fox-Hughes. Dynamic wildfire navigation system. ISPRS International Journal of Geo-Information, 8(4), 2019.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[15] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. Annual Review of Control, Robotics, and Autonomous Systems, 1, 05 2018.

[1] https://francescomrn.github.io/AEC2023-Non-Static-Environments-with-Autonomous-Drones/

DOI: 10.13009/EUCASS2023-326

[16] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and Service Robotics, 2017.
[17] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1205–1212. IEEE, 2021.
[18] Andrew Trask. Grokking Deep Learning. Manning Publications Co., USA, 1st edition, 2019.
[19] Tharindu Weerakoon, Kazuo Ishii, and Amir Ali Forough Nassiraei. An artificial potential field based mobile robot navigation method to prevent from deadlock. Journal of Artificial Intelligence and Soft Computing Research, 5:189 – 203, 2015.
[20] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv, abs/1707.06347*.
[21] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. *ArXiv, abs/1812.05905*.
[22] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. & Riedmiller, M.. (2014). Deterministic Policy Gradient Algorithms. Proceedings of the 31st International Conference on Machine Learning, in Proceedings of Machine Learning Research 32(1):387-395 Available from https://proceedings.mlr.press/v32/silver14.html.