A Method for Accelerated Simulations of Reinforcement Learning Tasks of UAVs in AirSim

Alberto Musa, Luca Zanatta, Francesco Barchi, Andrea Bartolini, Andrea Acquaviva
Università di Bologna
Viale del Risorgimento, 2, Bologna, Italy
alberto.musa@unibo.it, luca.zanatta3@unibo.it, francesco.barchi@unibo.it,
a.bartolini@unibo.it, andrea.acquaviva@unibo.it

***Abstract*—Reinforcement Learning (RL) is widely used for training Unmanned Aerial Vehicles (UAVs) involving complex perception information (e.g., camera, lidar). RL achievable performance is affected by the time needed to learn from the direct interaction of the agent with the environment. AirSim is a widely used simulator for autonomous UAV research, and its photorealism is suitable for algorithms using cameras for making or assisting flying control decisions. This work aims to reduce the RL time by reducing the simulation time step. This impairs simulation accuracy, so the impact on RL training must be quantitatively assessed. We characterise the AirSim acceleration impact on RL training time and accuracy while performing an obstacle avoidance task in a UAV application. We observed that using 5x as the Airsim acceleration factor, the RL task performance degrades by 95%. The observed performance increase is due to the latencies present in the AirSim command chain. We overcome this limitation by proposing a new command approach which allows accelerating without performance degradation until 10x. When pushing the acceleration factor to the extreme (100x), the RL task performance degrades by 38% with a measured speed-up of 15x.**

***Index Terms*—Simulation Engine, Reinforcement Learning, Airsim, Profiling.**

I. Introduction

Reinforcement Learning (RL) is a branch of Artificial Intelligence (AI) that aims to extract knowledge from the interaction of an agent with the environment. In robotics, specifically Unmanned Aerial Vehicles (UAVs), RL algorithms are used to train and implement control tasks such as navigation, landing, and obstacle avoidance [1]. These tasks require accounting for complex relationships between different sensors and complex dynamics that are difficult to catch with a dataset of use case samples. To bypass this problem, RL enables an agent's training in a task thanks to a reinforcement signal.

RL training requires the agent to directly interact with the environment experiencing the task to be learned and learn the task through a reinforcement signal. Practically, this requires performing the task (called game) repeatedly until the RL algorithm has learned the task. This can easily require thousands or millions of games [2]. According to which RL algorithm is used, stochastic gradient descent and back-propagation step are repeated continuously after profiling a series of actions and collecting the rewards within each game. In practical terms, several weeks to train the drone are required; most of the training time is spent in the simulation steps.

Software simulators facilitate the setting up of the training environment. Simulators are designed to replicate the agent capabilities (e.g., the UAV flight) and real environments through complex mathematical models that simulate agents' real-world physics rules and perception capabilities. Simulators implement all the forces that act in the simulated scenario, such as gravity, rotors actuation, and collisions.

If the task to be performed by the agent leverages camera sensors, accurate and photorealistic rendering of the scene becomes mandatory to obtain a simulation close to the real scenarios. RL combined with a photorealistic simulator reduces time-to-market, increasing the chances that trained models in the simulation can be deployed in the field as it is. According to a recent review [3], the most suitable simulators for UAV applications are: AirSim [4], Flightmare [5], Gazebo [6] and Webots [7]. These are all open-source simulators.

Gazebo and Webots are used by the robotic community but are not photorealistic. Airsim and Flightmare leverage graphical engines for the video game industry to provide photorealism. While Flightmare focuses on multi-drone simulations, Airsim natively supports hardware and software in co-simulation for rapid flight control and RL-trained solution rapid prototyping.

Simulators allow controlling an agent in the simulation by issuing commands in real-time. For this reason, the simulation time is synchronous to the "wall clock" time spent. Moreover, simulators can simulate physics faster than in real-time, provided that they run on adequate computing resources. However, photorealistic simulators rely on game engines for the rendering part, designed to maximize the number of rendered frames under the real-time bound.

A recent survey [8] identifies simulation environments that are photorealistic, accelerable and support the employment of RL algorithms. RaiSim [9] is based on the Unity game engine and allows for regulating the simulation speed. Isaac Gym [10] is a physics-accurate and photorealistic simulation tool developed by NVIDIA that supports automatic acceleration. However, at the time of the writing, Isaac Gym was a preview release, and NVIDIA plans to integrate it into the NVIDIA Omniverse Platform. Unity ML [11] is a plugin of the Unity game engine that provides the tools to set up virtual environments for RL. Unity ML allows to speed up the simulation by changing a time scale parameter. However, Unity simulates reality by showing a series of discrete simulation frames at variable time intervals that depend on rendering time. This implies that by increasing the time scale in a simulation,

 ISBN: 978-1-68558-001-8

the rendering time does not change. The analysis reported in the survey [8] focuses on RL simulators for generic robotic applications and does not consider AirSim among the possible valid alternatives. Not all general-purpose robotics simulators currently provide hardware support and software stack for Flight Control systems. Although they provide some UAV models, co-simulation may be decisive for adapting to real employment. Finally, AirSim from 2016 to 2020 is confirmed among these candidates as the most cited simulation environment [3]. The scenario at the moment makes us think of AirSim as one of the most suitable UAV simulators for RL.
In this regard, in the following sections, we will answer two research questions: RQ1: *Is it possible to accelerate RL training for UAVs in AirSim?* RQ2: *Which one is the implication in terms of accuracy of the simulated drone flight, interaction with the environment, and trained RL algorithm accuracy?*
Referring to the above-mentioned research questions, in this paper, we achieved the following contributions:
1) We discovered that the accuracy of trajectories degrades fast with the simulation time step reduction due to the latency incurring in the communication between the UAV agent and the RL algorithm.
2) We proposed a new method of controlling the agent in the simulator through asynchronous commands and time barriers (code available at [12]) In this way, on a pre-defined deterministic path, we obtained an effective speed-up of 62x with an error on the trajectories of 9% w.r.t. real-time simulation.
3) Compared to the results we obtained on a pre-defined deterministic path, the attainable acceleration speed-ups are significantly lower w.r.t. the real-time simulation. We investigated the issue, discovering that the rendering pipeline of the game engine introduces approximations and errors when the simulation is accelerated. To mitigate this problem, we added a valid check on the image provided by the camera API and re-issued the rendering if needed.

We performed the same experiment of an obstacle avoidance RL task in a lane of 150 m presented in [13]. In this, we evaluated the impact of the simulation acceleration with and without the proposed solution. With the default AirSim asynchronous commands, our results show that it is impossible to train an RL algorithm with accelerated simulation as the accelerated training fails over any acceleration factors. On the contrary, we obtain the same real-time training performance from a week of training to less than two days with the proposed synchronous command.

In Section 2, we describe the basic concepts of RL and the key characteristics of AirSim. A motivational example regarding acceleration issues, a possible solution to them and a validation strategy are explained in Section 3. In Section 4 the results of the tests carried out are shown. We conclude our work in Section 5.

II. Background

A. Reinforcement Learning

Deep RL aims at solving complex robotic tasks by mimicking the human training behaviour with the use of the neural networks [1].

Thanks to the policy π, described by a non-linear universal function approximator called a neural network, the agent chooses an action a_t, modifying the environment. The environment sends to the agent the new visible information (called observation) s_t and the reward r_t that measures the goodness of the action [1].

In particular, a well-known algorithm is the Deep Q-Learning (DQN) [14]. The goal of this algorithm is to maximize the action-value function:

$$Q^{\pi}(s,a) \approx r + \gamma \max_{a'} Q^{\pi}(s',a') = Q^{\pi}_{tar}(s,a) \qquad (1)$$

where $Q^{\pi}_{tar}(s,a)$ is the target action-value function and $Q^{\pi}(s',a')$ is produced by the neural network that is used for choosing the actions a in a state s. Since the duration of the task can be unlimited, the rewards are multiplied by a discount factor γ.

An RL algorithm is composed of two phases called training and inference. In the training phase, a policy is learned by the policy network using a loss defined as mean square error between $Q^{\pi}_{tar}(s,a)$ and $Q^{\pi}(s,a)$. In the inference, a trained network described by the parameters θ is used for acting according to the policy π. In the following, we refer to this network as a "neural network (π_θ)".

B. AirSim

AirSim is an open-source simulator, developed by Microsoft, oriented to vehicle simulation and specifically for UAV [4]. It is built on Unreal Engine 4 and its target is the experimentation of RL for autonomous vehicles. Unreal Engine is in charge of updating the state of the UAV agent. The physical state of a UAV agent in AirSim comprises six measures: position, orientation, linear velocity, linear acceleration, angular velocity, and angular acceleration. In addition, AirSim implements sensors models such as Cameras and Lidars.

In the rest of the paper, we will refer to the "state of the UAV" in a broad sense, including both the physical state of AirSim and sensor outputs. Simulation settings of AirSim can be configured statically throughout a JSON file. Among the many parameters, this file allows to (i) enable/disable the scene rendering while keeping the rendering for the sensor's camera always active, and (ii) changes the ratio between simulation and the wall clock time (*ClockSpeed* setting). As an example $ClockSpeed = 1$ implies real-time simulation, while $ClockSpeed = 10$ implies an 10x acceleration factor. We will refer to it also as the AirSim accelerator factor and leverage it to speed-up RL training tasks.

Although external Flight Control Systems are supported, by default, AirSim provides its own, called *Simple Flight*. The peculiarity of the *Simple Flight* is using a stoppable clock to pause the simulation at any point. AirSim provides a set of APIs (Client APIs) to interact with this built-in autopilot. The Mission Computer algorithm uses these Client APIs to issue commands and observe the UAV state. We will refer to this in the next section as Mission Computer Application (MCA). According to the complexity of the MCA, following multiple UAV states could be needed to compute a single command.

 ISBN: 978-1-68558-001-8

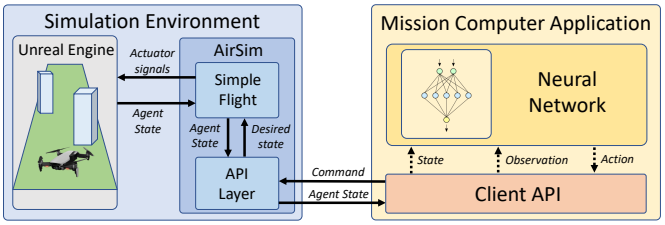

Fig. 1. The full pipeline of the RL setup. The blue box is the simulation environment. The yellow box represents the MSA in which the policy neural network (π_θ) is who chooses the agent's actions. The API provided by AirSim ables the communication between both components.

The Client API offers an asynchronous command allowing the MCA to set the desired UAV speed. The asynchronous command requires a set of input parameters: the velocity vector and the command duration. After this command is issued, the *Simple Flight* starts to execute it, and the MCA is free to continue with its program flow. Before launching another command, the MCA must wait for the previous command completion (using the *join()* method of the Client API); otherwise, the previous one will be overwritten - practically enforcing a synchronous control.

To know the UAV state, AirSim offers two data gathering methods: i) On-demand query via Client API; ii) Built-in periodic recording in a log file. The MCA can gather multiple UAV states during the command execution by issuing a concurrent thread that pauses the simulation at specific time intervals and gathers the UAV states. For this reason, additional input is required to specify the UAV state sampling time (T_{sample}). We refer to this control sequence as AirSim Default Synchronous Command (ADSC). In the following sections, we will study how the ADSC impairs the accuracy of the simulation when accelerated with the $ClockSpeed > 1$.

III. Methodology

A. Motivational Example

In the RL paradigm, no dataset is provided since the agent has to learn directly from his experience. Figure 1 depicts the communication flow between the MCA and the simulation environment. The policy neural network (π_θ) provides the chosen action to the environment. When the simulator executes the action, the environment evolves following its transition function; then, it sends the new state and a reward score to the agent. The reward score represents the goodness of the chosen action.

The simulated environment must replicate as faithfully as possible the real-world physics. Client API allows the RL algorithm to query the simulation environment to collect the current agent state and to send commands to the agent. The *Simple Flight* pilots the UAV agent according to the directives received by the neural network, which acts as a MCA.

To explain how *ClockSpeed* affects the simulation, we illustrate the callback timing of the synchronous commands and the velocity trend on different acceleration values. We use a path composed of three consecutive movements with ADSC along the same axis (two steps forward and one step back) in

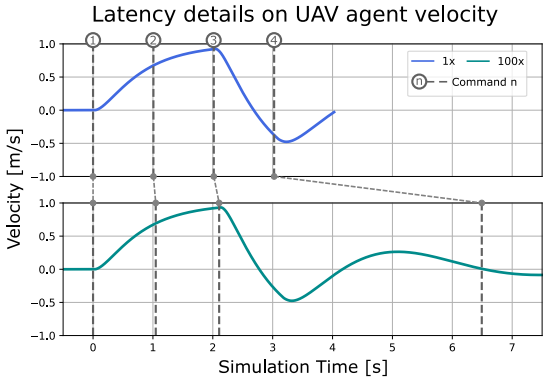

Fig. 2. Latency comparison between a real-time game (in the top) and a 100x accelerated game (in the bottom). The x-axis represents the simulation time in s, and the y-axis the agent velocity in m/s. Vertical lines indicate the time expired command.

the standard environment provided by AirSim, disabling the rendering. The data relating to the agent's speed is collected every 0.05 s of simulation time using the record mode, while the timing of the callbacks is gathered by client API. In this example, the imposed movements act in the same axis at a velocity of 1 m/s for 1 s.

Figure 2 shows on the x-axis the simulation time and on the y-axis the speed of the UAV agent. With dashed lines, the plot reports the time command expiration of the UAV agent action in a real-time simulation (the top plot) and a 100x accelerated simulation (the bottom plot). The figure shows that the accelerated simulation is subject to command delays as the dashed lines do not happen precisely on the instants in which the command was issued. We have identified the cause of the delays in: i) Latency between the simulator and the application during the data gathering; ii) Latency of the *Simple Flight* in communicating the move command to the UAV agent. These latencies are involved in ADSC as the MCA waits for the synchronous command completion.

The delay in the move command increases with the accelerator factor. Latencies also exist in real-time simulations (*ClockSpeed*=1) but become noticeable only when the acceleration increases. During the command delay, the simulator continues to evaluate the UAV.

Figure 3 shows the coordinates of the drone flights in real-time (blue line) and in 100x acceleration (green line). In the plot axes, the 3D Cartesian coordinates are reported. We can see the impact of the latencies on the agent's trajectory by accelerating the simulation in a single game.

Another issue related to the acceleration of the simulation is the quality of the images collected by the agent camera. Enabling the rendering in an accelerated game and visually inspecting the camera images, we noticed that some images are affected by salt-and-pepper noise and disturbances in brightness in very dark scenes. We also noted issues with the rendered colours of the objects in the scene. Unreal Engine renders the scene by following a pipeline in which processing is computed in multiple phases. Accelerated simulation reduces the rendering time and the time available for each stage, making some of them hard to be completed and perturbing the

 ISBN: 978-1-68558-001-8

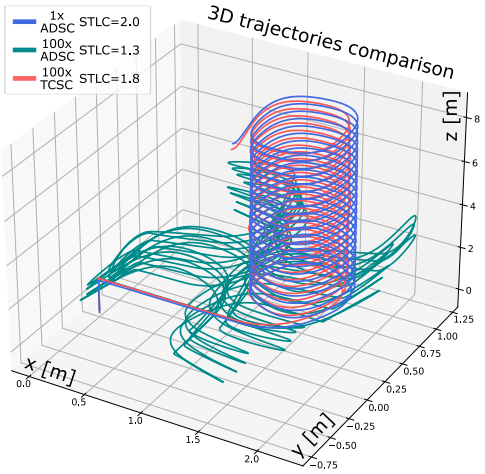

Fig. 3. Comparison of trajectories between a real-time game (in blue) and a 100x accelerated games. The axes x-y-z represents the 3D coordinates of the simulation environment.

returned images. This issue must be mitigated since the quality of camera images is essential for considered RL tasks.

B. Speed-Up Effects Mitigation

We demonstrated how communication latencies affect the accelerated simulation modifying the UAV agent trajectories. While the MCA waits for the end of the ADSC, the simulation continues at the selected *ClockSpeed*. The error in the accuracy of the trajectories exacerbates as the Airsim acceleration factor increases.

To overcome this issue, we propose a new command called Time-Controlled Simulation Command (TCSC) in this paper. The TCSC uses the stoppable clock of the *Simple Flight* to control the simulation until the command expires. This control is implemented by periodic stimulation interrupts interleaved with command expired-time checks.

Figure 4 details the trellis diagram of TCSC. The MCA issues the TCSC with the associated inputs: the velocity vector, the expected command duration (T_{move}), and the UAV state sampling time (T_{sample}), then:
1) TCSC measures the start time (T_start) and issues the asynchronous command in the Clint API with a command duration equal to T_{move}. 2) Internally the TCSC leverages the Client API of AirSim to continue the simulation until a predefined time ($T_{wakeup} = minT_{move}, T_{sample}$) is expired.
3) When this happens, the simulator is paused, and the TCSC uses the Client API to collect the current simulation time (T_{sim}). If T_{sim} is lower than the $T_{start} + T_{wakeup}$, the TCSC continues the simulation for the differences between the two times ($T_{wakeup} = T_{start} + T_{wakeup} - T_{sim}$). If this difference is below a minimum threshold (T_{th}), set as the Client API latency, than T_{sim} is considered larger than $T_{start} + T_{wakeup}$.
4) If T_{sim} is larger than the $T_{start}+T_{wakeup}$ the TCSC gathers the UAV state. At this point if $T_{sim} \geq T_{start} + T_{move}$ the TCSC ends, otherwise it sets $T_{move} = T_{move} - T_{sample}$, and returs to (2.).

To avoid interference between the AirSim acceleration factor and the camera images in the simulation, we disabled the

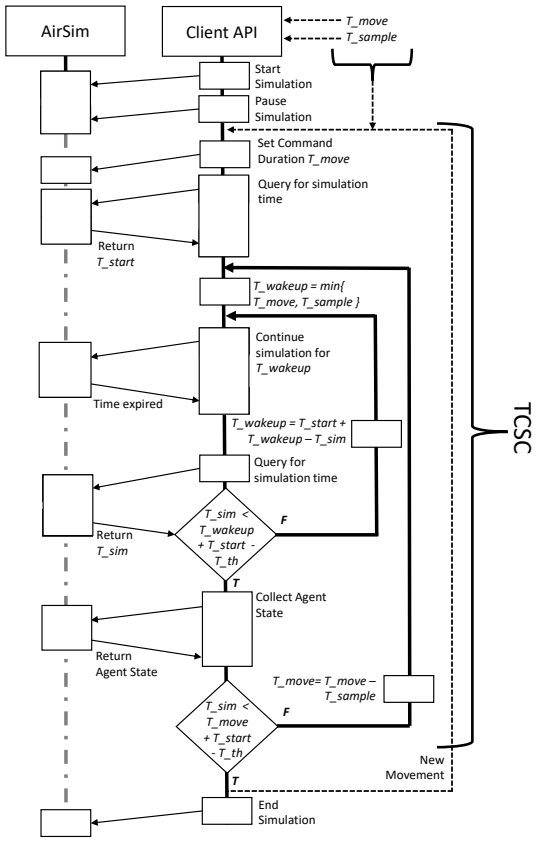

Fig. 4. This trellis diagram represents the TCSC. Inputs are the time duration command T_{move}, agent state sampling time T_{sample} and a velocity vector (not used in the graph).

camera auto exposure. Moreover, we implemented a simple method based on the average image brightness to solve the camera image perturbation. The assumption is that this parameter slowly changes across consecutive camera images. Suppose the average image brightness changed more than the 10% w.r.t previous camera image. In that case, the image is considered perturbed, and a new camera image is requested to the rendering engine through the Client API. In addition, the average image brightness is compared with a minimum threshold to ensure that the camera image is not too dark. During these checks, the simulation is paused; thus, the simulation accuracy is not impaired.

C. Characterization Methodology

Evaluating the impact of the simulation acceleration on RL tasks requires assessing different effects: (i) the trajectory accuracy; (ii) the visual perception accuracy; (iii) their impact on the agent performance trained in RL.

For this purpose, we defined the following methodology:

- Replicating a State-of-the-Art (SoA) UAV RL task [13], consisting of learning an obstacle avoidance policy based on images with AirSim.
- Defining two pre-defined deterministic control sequences (path) on which to assess the (i) accuracy of a trajectory and (ii) the visual perception accuracy.

 ISBN: 978-1-68558-001-8

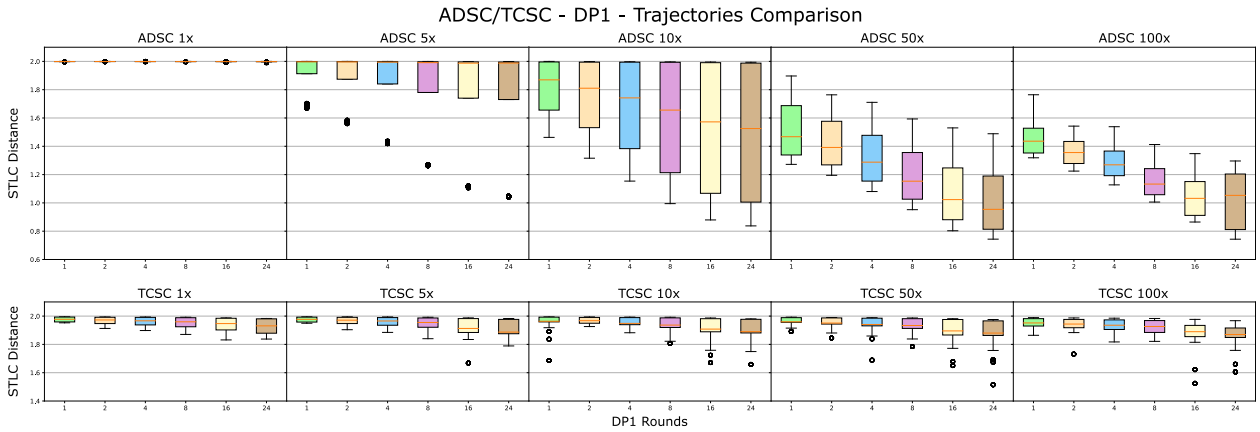

Fig. 5. The plot reports the STLC distance (y-axis) of the trajectories recorded, computed in averaging pair-wise, for each game replica combination performed in DP1. The references are the ADSC real-time games. In the sub-plots x-axis, the DP1 command length (DP1-1r to DP1-24r) is marked. Sub-plot rows identify the command used to perform the games (ADSC and TCSC). Sub-plots columns identify the AirSim accelerated factor. STLC distance has a score of 0 and 2, where 2 means trajectories completely overlapped.

- Comparing the accuracy of an agent trained in RL with different simulation acceleration.

1) Environment Setup: To improve how TCSC affects the simulation and the RL performance, we replicate the experiments performed in [13]. The task involves the avoidance of obstacles along a lane of 150 meters. We will refer to this simulated environment as the RL AirSim environment (AirSim-RLE). We mimic a Dynamic Vision Sensor (DVS) in which every pixel detects the variation of the log brightness. For this purpose we used the v2e [15] DVS model. If the variation exceeds a certain threshold, an event is created. An event is defined as a tuple of four elements: the coordinates of the pixel (x,y), the timestamp, and the polarity (which can be $+$ or $-$). We simulated a UAV flying by steps of 1 s at a speed of 1 m/s and collecting three frames per command. For this reason we set $T_{move} = 1s$ and $T_{sample} = 0.33s$. Then a completed game requires more than two minutes.

In addition to the AirSim-RLE, we leverage an additional AirSim environment, the default AirSim one, Blocks, on which we perform the trajectory accuracy tests. We will refer to this environment as the AirSim default environment (AirSim-DE).

2) Performance Metrics:

a) Trajectory accuracy: To quantify the effect of the latency on the trajectories, we sampled the position of an agent in the AirSim-DE with rendering disabled. The test is performed for both commands, ADSC and TCSC. The paths are composed of a repetition of four movements designed to move on a square perimeter (Deterministic Path #1 - DP1). The UAV will move on the path for an increasing number of times per game: we considered the following number of rounds (1,2,4,8,16,24). We will later refer to these combinations as (DP1-1r, DP1-2r, ..., DP1-24r). The agent speed is 1 m/s while the command duration varies from 0.1 s to 1 s. We perform twenty games per path for each AirSim acceleration factor and command duration. The data collected are composed of a set of 3D coordinates with the corresponding simulator time with a sampling time of 0.05 s. We use the Spatiotemporal Linear Combine distance (STLC) [16] to compare the different trajectories. We use as a reference the real-time trajectory. For each AirSim acceleration factor, we compute an average STLC distance by averaging the pair-wise STLC distance computed for each game replica combination (resulting in a 20x20 matrix). This metric has a score of 0 and 2, where 2 is the maximum value indicating that the trajectories are entirely overlapped. As a reference, the trajectories reported in Figure 3 have an STLC distance with the trajectory A of 1.3 for the trajectory B and 1.86 for the trajectory C. We conclude that STLC distances above 1.8 means visually similar trajectories.

b) Visual perception accuracy: Assessing the visual perception accuracy of a UAV flight simulation is of primary interest if a camera input guides the drone control task. We thus need to assess if the AirSim acceleration factor introduces visual perception accuracy distortion on top of the UAV trajectory errors. We set the UAV to follow a pre-defined deterministic path which reaches the end of the AirSim-RLE (without obstacle collision). We will refer to this as the Deterministic Path #2 (DP2). We repeated this simulation one time for each AirSim acceleration factor considered. The UAV commands used is TCSC with $T_{move} = 1s$ and $T_{sample} = 0.33s$. For each UAV state gathered, we collected the camera image and the UAV coordinates. We then compute for each sample: (i) The Euclidean distance between the coordinates at which the camera image is taken in the accelerated simulation and real-time simulation; (ii) The difference in the total number of DVS pixels in each polarity between the camera image taken in the accelerated and real-time simulation. This difference is reported as a percentage of the total number of pixels in the camera images with a resolution of 256x144 pixels. We will refer to this metric as DVS Event Dissimilarity (DVS-ED). The introduced metrics allow us to understand how the quality of the images changes in relation to the difference in the UAV coordinates when the image is taken. For nearby acquisition points, we expect a slight difference in the images obtained concerning an accelerated simulation. Otherwise, we can conclude that the AirSim acceleration factor induces an additional perturbation in the gathered camera images.

3) RL agent performance: The strategy of validating RL algorithms on accelerated simulations involves training the

 ISBN: 978-1-68558-001-8

UAV agent in RL for the designated task (obstacle avoidance) at different AirSim acceleration factors. The RL training procedure performs 5 000 games on the previously described AirSim-RLE. This test is replicated using both the ADSC and the TCSC. During the training, we record the simulation, image processing (DVS computation), and the policy neural network (π_θ) inference times.

After the RL training ends, as described in Section II-A, a policy neural network (π_θ) is obtained. We evaluate the accuracy of the UAV's flight with the trained policy neural network, and we measure the number of times the UAV reaches the end of the environment without collisions over 100 games - later, we will refer to this metric as the UAV success ratio.

First, we compared the two commands, ADSC and TCSC, in real-time. Using a real-time simulation, we trained the policy neural network in RL using the proposed TCSC. Then we evaluated the UAV success ratio using both TCSC and ADSC in inference over 100 games. This test provides evidence of the interchangeability of commands when the simulation happens in real-time, where the effects of latencies are negligible.

Second, we compare the UAV success ratio computed in real-time for the policy neural network (π_θ) trained with different AirSim acceleration factors. UAV success ratios equivalent to those obtained in the real-time training (reference case) are considered valid AirSim acceleration factors. To check the wall-time gained with the simulation acceleration, we calculate the effective speed-up of training procedures with different AirSim acceleration factors compared to the ADSC real-time one.

IV. Results

A. Experimental Setup

In this section, we describe the results obtained from the characterization methodology described in Section III-C. The tests have been performed in a server equipped with a 24-GB RAM NVIDIA Quadro RTX 6000 GPU, Intel(R) Xeon(R) Gold 5220 CPU @ 2.20 GHz and 188 GiB of RAM. In this section we considered the following AirSim acceleration factors: 1x (Real-time - Baseline), 5x, 10x, 50x, 100x.

B. Trajectory accuracy and speed-up

The first characterization test focuses on evaluating the impact of the AirSim acceleration factors on the trajectory accuracy for both the ADSC and the proposed TCSC. This is done in the AirSim-DE with the UAV flying following the pre-defined deterministic path DP1.

Each command considered (ADSC/TCSC) has been simulated at each AirSim acceleration factors [1x, 5x, 10x, 50x, 100x] for different command durations (0.1 s, 0.2 s, 0.5 s, 1 s) for an increasing command length (DP1-1r to DP1-24r). Each simulation setting has been repeated for 20 times.

Figure 5 reports the distribution of the STLC distance for each setting computed against the ADSC real-time case (ADSC-1x), which we consider as the baseline. This means that for all ADSC-1x steps, the STLC distance reaches near maximum value, $STLC_d = 2$. Slight differences are due to the non-determinism of the simulator, for which the trajectories vary between repetitions. In other tests, the variance of the STLC distance distributions increases as the AirSim acceleration factor increases. In the figure, we group the computed STLC distance based on the command length. The top plot refers to the ADSC case, while the bottom plot refers to the TSCS case. Different sub-plots refer to different AirSim accelerator factors, while the x-axis reports the command length on which the STLC distance has been computed.

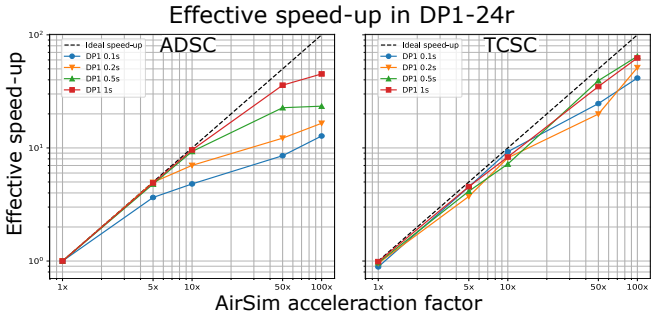

Fig. 6. Effective speed-up of the trajectories accuracy test on DP1-24r. Effective speed-ups (y-axis) are distinguished by the command duration. The x-axis identifies the set AirSim acceleration factor. The effective speed-up is computed for both commands, ADSC (on the left) and TCSC (on the right). The black line indicates the ideal curve.

For a given AirSim acceleration factor, the trajectory accuracy worsens with the command length, suggesting the error accumulates between consecutive commands. This is expected as the source of the trajectory error is the command latency. For the same reason, the proposed TCSC command significantly outperforms the ADSC trajectory accuracy.

Let us consider achieving an STLC distance higher than 1.8 in the median case (reported with a red line in the box plot). The proposed TCSC provides acceptable trajectory accuracy for all the AirSim acceleration factors while the default ADSC reaches it for acceleration lower than 5x.

To evaluate how an AirSim acceleration factor translates to simulation time reduction, we measured the time taken to perform the simulation in the above-tested settings and computed the effective speed-up against the real-time baseline. Figure 6 reports this value for the ADSC (left plot) and TCSC (right plot) for the different AirSim acceleration factors (x-axis). Different lines refer to different command durations. In the y-axis, we report the average effective speed-up computed against the real-time ADSC case, averaged among the repetitions and command lengths. With the dashed black line, we report the ideal effective speed-up curve.

From the plot, we can notice that, in general, the TCSC outperforms the ADSC showing an effective speed-up closer to the ideal curve. For the ADSC, the effective speed-up increases by increasing the command durations. This effect is more visible at high AirSim acceleration factors. This can be explained by the fact that for each command, the time spent in it is composed of the time of processing the command, the communication and callback latency, and the simulation time. The AirSim acceleration factor reduces only the simulation time. For shorter command durations, the simulation time is weightless on the total time; thus, the simulation acceleration

 ISBN: 978-1-68558-001-8

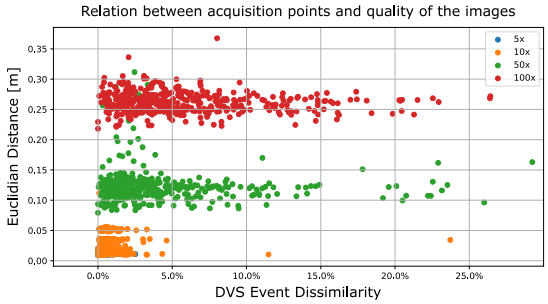

Fig. 7. Correlation between UAV position error measured and DVS-ED (x-axis) in DP2. Distance from positions is computed with euclidean distance (y-axis). Colors indicate the set Airsim acceleration factor.

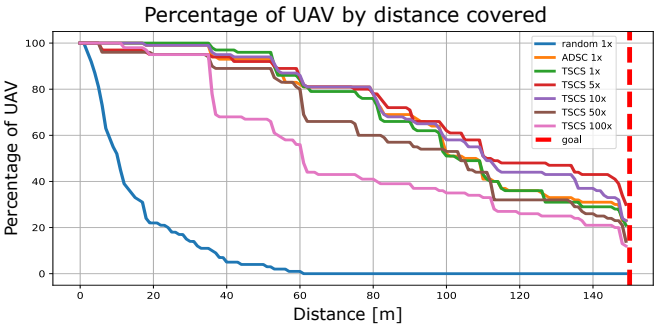

Fig. 8. Distance traveled in meters (x-axis) by the UAV agents during the 100 games inference (y-axis). The lines refer to the inference executed on different AirSim acceleration factors training. The red line represents the goal of the AirSim-RLE lane.

benefits reduce. It is interesting to note that the proposed TCSC shows better scalability. This can be explained by the fact that the ADSC has an additional wait callback w.r.t. the TSCS, which is the source of a more significant amount of latency at high AirSim acceleration factors.

C. Visual perception accuracy

In this subsection, we evaluated the impact of the AirSim acceleration factor on the visual perception accuracy simulations. This is done by simulating the different AirSim acceleration factors of a UAV moving on the pre-defined deterministic path DP2 on the AirSim-RLE.

Figure 7 reports on the x-axis the DVS-ED and the y-axis the UAV position error measured as the euclidean distance. Both metrics are computed for each DVS camera image captured by the drone in the DP2 against the real-time AirSim acceleration factor one. Each point corresponds to a DVS camera image. Different colour refers to the different AirSim acceleration factors.

TABLE I
SUCCESS RATIO RELATING TO DIFFERENT AIRSIM ACCELERATION FACTORS TRAINING EVALUATED IN REAL-TIME INFERENCE. THE SUCCESS RATIO, COMPUTED IN AIRSIM-RLE IS PERFORMED WITH THE SAME TRAINING COMMAND (ADSC AND TCSC).

	AirSim Acceleration Factor				
	1x	5x	10x	50x	100x
ADSC	23	1	0	0	0
TCSC	21	30	23	14	13

As shown in Figure 7, the points relative to an AirSim acceleration factor of 5x are close to the origin: the euclidean distance is within 5 cm, the DVS-ED is below 3% from the DVS camera image obtained in real-time. This means that the simulation accelerated to 5x is almost identical to the real-time simulation. Increasing the AirSim acceleration factor to 10x slightly increases the DVS-ED to 5% while still maintaining the euclidean distance below the 5 cm. Also, there is not a big difference from the real-time simulation. It is interesting to notice that when the AirSim acceleration factor is increased to higher values (50x and 100x), the Euclidean distance increases but less than the DVS-ED. Moreover, at these AirSim acceleration factors, many points have similar euclidean distances but significantly different DVS-ED. This means that the difference in the images seen from the UAV's camera does not depend only on the position error induced by the simulation acceleration but that the latency in command can lead to a different camera orientation while the UAV is in a relatively similar position.

D. RL agent performance

This last section evaluates how the effects discussed impact the performance of an agent trained in RL in an accelerated simulation environment. As described in Section III-C3 we trained with RL a policy neural network (π_θ) to avoid obstacles based on the DVS camera input. In real-time with the ADSC (ADSC-1x), the training of 5000 games takes one week and 14 hours. The agent achieves a UAV success ratio of 23%.

Table I reports the UAV success ratio obtained when training the policy neural-network (π_θ) at different AirSim acceleration factors. All the UAV success ratios have been computed in real-time in the AirSim-RLE.

We can notice that the TCSC-1x has a similar performance to the baseline achieving the 21% of UAV success ratio. Moreover, the UAV success ratio is preserved till AirSim acceleration factors of 10x. For higher accelerations (50x and 100x), the score reduces to 62%. This indicates that the proposed command can preserve the RL training accuracy until a simulation acceleration of 10x. It is interesting to notice that the AirSim acceleration factors of 5x improve upon the ADSC-1x, suggesting that for this acceleration, the trajectory noise helps the trained model to generalize. In contrast, at higher acceleration factors, the loss in the DVS accuracy (DVS-ED) increases, jeopardizing the training accuracy.

Figure 8 reports on the x-axis the distance travelled in meters by the UAV agent in 100 trials (games) simulated in real-time using the policy neural network (π_θ) trained with accelerated simulations. For a given AirSim acceleration factor (reported in the plot with different colours), all the 100 trials have been simulated using the policy neural network weights (θ) obtained during the corresponding accelerated training. The y-axis reports the number of UAV agents sharing the same, which reached a given distance. In the plot, we report as well the reference cases performed in real-time with ADSC and TCSC are plotted. The lines have a similar trend with peaks corresponding with obstacle locations. The figure validates the TCSC experiment by demonstrating that simulation acceleration applied to RL algorithms does not alter

 ISBN: 978-1-68558-001-8

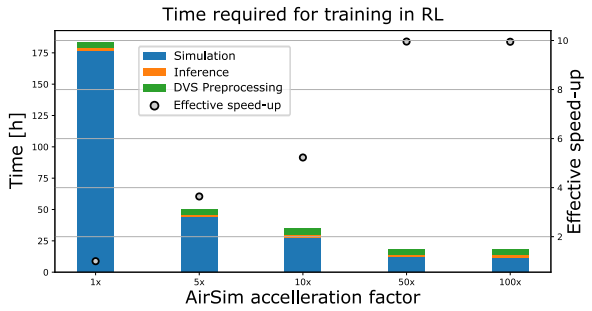

Fig. 9. Time in hours required for each training phase and effective speed-up of the training. The x-axis indicates the set AirSim acceleration factor. Y-axes indicate the hours required for the training separated by phases (on the left) and the effective training speed-up (on the right).

the simulation environment. By increasing the acceleration of the simulation, the trajectories and perception perturbations increase, impacting the performance of the RL.

Finally, Figure 9 reports on the left y-axis the simulation time required by the RL training for each AirSim acceleration factor (x-axis). On the right x-axis is reported the effective speed-up measured w.r.t. the real-time training time. With different colours, we account for the time spent in the RL training in performing the UAV agent in simulation, the policy neural network (π_θ) inference (choice of the following action), and image pre-processing to compute the DVS image. From it, we can notice that in real-time, the training time is dominated by the simulation time, which gets reduced by the AirSim acceleration factor. However, when comparing the effective speed-up with the one previously obtained with the rendering disabled (Figure 6) we can recognize a significant drop in the scalability. Indeed, with the rendering disabled, the effective speed-up for the TCSC was 50x. For a 100x AirSim acceleration factor (*ClockSpeed*) in the RL training (w. rendering enabled), we achieve for the same AirSim acceleration factor a 15.4x of effective speed-up. For the other AirSim acceleration factors, when considering only the simulation time, we obtain 4x speed-up with *ClockSpeed* set to 5x and 6.4x speed-up with 10x. At 50x *ClockSpeed* the achievable acceleration saturates to 14.8x of speed-up.

We can conclude that thanks to the proposed TCSC approach, the training time of one week has been reduced to two days (3.6x with $ClockSpeed = 5$) or one day and a half (5.2x with $ClockSpeed = 10$) without impairing the accuracy of the trained policy. In contrast, we empirically demonstrated that theefault ADSC does not allows performing an RL training with accelerated simulation. *ClockSpeed* values of 50x and 100x do not equal real-time but still achieve excellent results.

V. Conclusions

In this work, we described a method to speed-up the training of UAV agents trained in RL by reducing the simulation time. We refer to this method as a Time-Controlled Simulation Command (TCSC) in opposition to the AirSim Default Synchronous Command (ADSC). At the same time, we mitigate the issues related to the simulation acceleration. The proposed TCSC mitigates the error on the trajectories of UAV agents in accelerated simulation up to 65x diverging within 10 % compared to those computed in real-time. To validate the TCSC on RL algorithms, we replicated the work done by [13]. The designated task involves avoiding obstacles in a 150m lane through the use of camera images. The acceleration of the simulation generates perturbations in the camera images. We mitigated this noise by introducing a method based on image brightness. We performed accelerated training with TCSC and ADCS, comparing them in 100 games inferences on real-time simulations. Training with TCSC on accelerated simulation up to 6.4x has the same UAV success ratio of the inference with training performed in real-time. By accelerating the simulation to 4x in training, we have improved the UAV success ratio of the inference with training performed in real-time. Training with *ClockSpeed* of 50x and 100x achieved effective speed-up of 14.8x and 15.4x and replicated more than 62% of the UAV success ratio in real-time. Accelerated simulation training with ADSC failed to complete games in real-time inference. With TCSC, a training that in real-time simulation requires one week of simulations has been replicated in less than two days.

References

[1] W. L. Keng and L. Graesser, "Slm lab," https://github.com/kengz/SLM-Lab, 2017.

[2] N. Salvatore *et al.*, "A neuro-inspired approach to intelligent collision avoidance and navigation," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pp. 1–9.

[3] J. Collins *et al.*, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51 416–51 431, 2021.

[4] S. Shah *et al.*, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," *CoRR*, vol. abs/1705.05065, 2017.

[5] Y. Song *et al.*, "Flightmare: A flexible quadrotor simulator," in *Proceedings of the 2020 Conference on Robot Learning*, 2021, pp. 1147–1157.

[6] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[7] Webots, "http://www.cyberbotics.com," open-source Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[8] T. Kim, M. Jang, and J. Kim, "A survey on simulation environments for reinforcement learning," in *2021 18th International Conference on Ubiquitous Robots (UR)*, 2021, pp. 63–67.

[9] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018. [Online]. Available: www.raisim.com

[10] V. Makoviychuk *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021. [Online]. Available: https://arxiv.org/abs/2108.10470

[11] A. Juliani *et al.*, "Unity: A general platform for intelligent agents," 2018. [Online]. Available: https://arxiv.org/abs/1809.02627

[12] L. Zanatta and A. Musa, "Ecs-lab / uav reinforcement learning with airsim · gitlab." [Online]. Available: https://gitlab.com/ecs-lab/uav-reinforcement-learning-with-airsim.git

[13] L. Zanatta *et al.*, "Artificial versus spiking neural networks for reinforcement learning in uav obstacle avoidance," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, ser. CF '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 199–200. [Online]. Available: https://doi.org/10.1145/3528416.3530865

[14] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[15] Y. Hu *et al.*, "v2e: From video frames to realistic dvs events," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1312–1321.

[16] H. Su *et al.*, "A survey of trajectory distance measures and performance evaluation," *The VLDB Journal*, vol. 29, no. 1, pp. 3–32, oct 2019. [Online]. Available: https://doi.org/10.1007\%2Fs00778-019-00574-9

 ISBN: 978-1-68558-001-8