A Nonlinear UAV Control Tuning Under Communication Delay using HPC Strategies in Parameters Space

Leonardo Fagundes-Junior[1], **Michael Canesche**[2], **Ricardo Ferreira**[1], **Alexandre Brandão**[1]

[1]Universidade Federal de Viçosa (UFV)
Avenida Peter Henry Rolfs – 36570-900 – Viçosa – MG – Brazil

[2]Dept. de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos – 6627 – 31270-901 – Belo Horizonte – MG – Brazil

{leonardo.fagundes,ricardo,alexandre.brandao}@ufv.br, mcanesche@ufmg.br

Abstract. *In practical applications, the presence of delays can deteriorate the performance of the control system or even cause plant instability. However, by properly controlling these delays, it is possible to improve the performance of the mechanism. The present work is based on a proposal to analyze the asymptotic stability and convergence of a quadrotor robot, an unmanned aerial vehicle (UAV), on the performance of a given task, under time delay in the data flow. The effects of the communication delay problem, as well as the response-signal behavior of the quadrotors in the accomplishment of positioning mission are presented and analyzed from the insertion of fixed time delay intervals in the UAVs' data collected by its sensors system. Due to the large search space in the set of parameter combinations and the high computational cost required to perform such an analysis by sequentially executing thousands of simulations, this work proposes an open source GPU-based implementation to simulate the robot behavior. Experimental results show a speedup up to* ***4900***$\times$ *in comparison to MATLAB® implementation. The implement is available in Colab Google platform.*

1. Introduction

System modeling is a crucial step in solving engineering problems, such as system analysis and control. Seeking to reproduce patterns of behavior observed in practice, a mathematical model consists of a set of differential (continuous time) or difference equations (discrete time) that describe the temporal and/or spatial variation of the variables of interest. The survey and mathematical formulation of all the phenomena that affect the behavior of a robot is an extremely complex task. In this sense, a model never exactly reproduces the behavior of the practical system, however, it can be estimated and approximated using techniques as parameters estimation for transfer functions and state space systems [Ji and Kang 2020]. In fact, the simplifications are valid and relevant in most cases. Despite such uncertainties, controller design techniques are able to approximate, with a certain accuracy, the model output to a reference value, restricting its operation to a set of constraints.

The problem of robot control is associated with stability and performance in accomplishing the task for which the robot was developed. With the discretization of controllers and the need for communication, either between robots or between robot and a ground control station (GCS), there is some delay in sending and receiving information. In particular, in aerial robotics, the presence of delay can cause deterioration in performance or even stability loss of the control system. The main difficult is the complex dynamics of unmanned aerial vehicles (UAVs) due to the inherent nonlinearities, uncertainties, time-varying parameters, high dynamic coupling, and potentially uncertain time delays in processing and/or communication. These undesired nonlinearities affect the flight stability and performance of the controlled systems. Some strategies take into account the robot's dynamics, reducing the control problem to a model analysis and stabilization of its posture over time [Han et al. 2017]. Due to the ability to generate high performance tracking in the presence of uncertainties, Adaptive (AC) and Robust Controls techniques (RC) are candidates to resolve this issues [Koksal et al. 2020].

Communication delay can arise from different reasons due to the characteristics of the different system components such as sensors, actuators, acquisition systems, physical components, and others. This work proposes to study the effects of delay in sending and receiving information from a quadrotor without paying attention to its sources. In summary, the objective is to observe the robot behavior under the condition of communication delay and to propose a method for evaluation and parameter tuning of conventional controllers in consideration of the presence of time delay while executing positioning missions. Due to the difficult and long time spent in analytical control tuning in parameters space (a few hours in MATLAB simulation), this paper proposes a high-performance computing (HPC) strategy for evaluating large parameter combination sets. In the approach, each GPU kernel runs a complete simulation using the controller with a possible parameter combination and analyze the robot behavior in terms of performance metrics. Furthermore, all experiments could be easily reproducible using others robot models and controllers, are distributed as an open-source, and it was implemented in C++ using the GPU made available in Google Colab[1]. In addition, other functionalities of the Jupyter notebook and Python are used to visually exhibit the simulation results and important graphics, such as UAVs' path, position and control signals evolution during the simulations.

The paper is organized as follows. Section 2 provides a brief description of the UAV modeling and control. In Section 3, the control tuning evaluation in parameters space, under time-delayed communication, is presented. Section 4 presents the proposed HPC control tuning and analysis method highlighting the adopted task parallelism. Section 5 shows our results and provide a few examples and comparing the performance analysis using MATLAB, C/C++ and GPU simulator versions. Finally, in section 6, we discuss our findings, concluding the paper, and addressing directions for future work.

2. UAV Dynamic and Kinematic Modeling

As for the UAV adopted, it is an *ArDrone 2.0* quadrotor, from Parrot Drones SAS, wich consists of a set of four engines positioned in the shape of a cross, which are independently driven. This is a quadrotor manufactured with a ARM Cortex A8 processor, with 1 GHz, and Linux operating system, in addition to on-board sensors that make it possible to know the robot's posture, according to ArDrone Parrot Software Development Kit (SDK) [Piskorski and Brulez 2012]. The collective variation in propulsion forces, resul-

[1]`https://colab.research.com/UAV_DelayedControl`

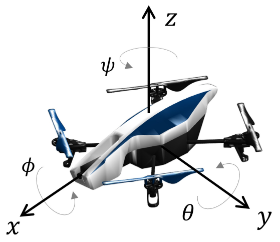


Figura 1. ArDrone 2.0 description and pose variables.

ting from the angular velocity of the engines, governs the three-dimensional navigation of the aircraft. Two opposite engines rotate clockwise, while the other two rotate counterclockwise, a configuration that eliminates the anti-torque effect on the fuselage caused by the rotation of the blades by the motors. There is also an internal controller responsible for takeoff, hovering and landing [Santana et al. 2015, Piskorski and Brulez 2012].

The quadrotor has six degrees of freedom (DOF) as shown in Figure 1, where the robots' gravitational center position is described by robot reference $(x,\ y,\ z)$, and the yaw (ϕ), roll (θ) and pitch (ψ) angles are related to the drone's rotations around its axes x, y and z, respectively. Its control is performed in an underactuated way, since it has a fewer number of actuators than the number of DOF [Brandao et al. 2013], using four control signals which are responsible for guiding the robot according to its current state and the mission definition.

The command signals sent to the robot, according to ArDrone Parrot Software Development Kit (SDK) [Piskorski and Brulez 2012], are

$$\mathbf{u} = \begin{bmatrix} u_\phi & u_\theta & \dot{u}_z & \dot{u}_\psi \end{bmatrix}^\intercal \in [-1,\ 1] \tag{1}$$

where:

- u_ϕ controls the roll angle, responsible for the left-right movement;
- u_θ controls the pitch angle, which results in forward and backward movement;
- $\dot{u}_z$ controls the vertical velocity;
- $\dot{u}_\psi$ is responsible for the yaw rate, which rotates the robot about the z-axis.

The study of control techniques applied to UAVs has been a widely explored topic in academia, with significant results already published. Some controller strategies (linear and nonlinear) can be found in [Brandao et al. 2013, Lv et al. 2020, Allahverdy et al. 2019]. In these works, the robot's kinematic and dynamic models are described for controller design. Since the robot has 6DOF and is controlled from the propulsion forces generated in the four engines coupled in the UAV, we need a underactuated model to describe the drones' dynamics. From the kinematic model of the aerial robot, already described in the literature by [Santana et al. 2016, Rabelo et al. 2018], it is possible to identify the UAV behavior and propose appropriate control strategies. A simple model, assuming that the pitch and roll angles are sufficiently small, i.e., with the UAV near-hovering, can be found in the aforementioned works. This models were used to perform simulations and validate the proposed approach.

At this point, it is important to mention that this vehicle is one of the most complex stabilization and control test platforms [Tang and Li 2015, Tang et al. 2017]. For this reason, when proposing the navigation model of a vehicle, if only its kinematic model is considered, one must ensure that it performs linear displacements at low speeds subject to the minimum action of external disturbances, in such a way that the dynamic effects can be neglected. Otherwise, since quadcopters have an inherently unstable and highly coupled nonlinear dynamic model, their dynamics must be considered when designing controllers [Brandao et al. 2013].

A possible way to represent the dynamic model of a UAV is through the Euler-Lagrange formulation. The dynamic model and controller for the quadcopter considered in this work will be the similar one developed by the authors in [Brandao et al. 2013].

3. Robot Control Under Time Delay

Since the focus of this research is not to propose a specialized robotic mechanism control technique to mitigate the effects of delayed communication, one of the controllers already developed by one of the authors will be implemented. In [Brandao et al. 2013] a nonlinear controller, based on the ArDrone high-level dynamic model, is described, given by

$$\mathbf{u}_d = \ddot{\mathbf{x}}_d + \mathbf{K}_1 \tanh\left(\mathbf{K}_2 \dot{\tilde{\mathbf{x}}}\right) + \mathbf{K}_3 \tanh\left(\mathbf{K}_4 \tilde{\mathbf{x}}\right) \tag{2}$$

in which, $\mathbf{K}_i \in \mathbb{R}^{3\times3}$, $i = 1, 2, 3, 4$, are positive diagonal gain matrices and $\tilde{\mathbf{x}} = \mathbf{x}_d - \mathbf{x}$ represents position error given by the difference between the desired and the current position. Analogously, we define the instantaneous velocity error as $\dot{\tilde{\mathbf{x}}} = \dot{\mathbf{x}}_d - \dot{\mathbf{x}}$, considering $\mathbf{x}$ as the variables referring to the robot position $[x,\ y,\ z]^\intercal$.

From *Lyapunov Stability Theory*, it is possible to show that the strategy guarantees stability and convergence of the reference calculated by the robot with respect to the position (or trajectory) specified for performing the task. In other words, for $t \to \infty$ the state errors tend to zero [Brandao et al. 2013].

3.1. Control Evaluation in Parameters Space

The quadrotor controls the rotation velocity of the blades. If the front two are accelerated, the frontal part of the drone rises and it will move backwards. If the back two are accelerated, the back goes up and the quadrotor moves forward. Therefore, it can be inferred from this analysis that the control of the propulsion motors (low level) is performed by the quadrotor in its internal mesh. The drone's momentum is conserved due to the characteristic that two propellers are turning clockwise and the other two are turning counterclockwise.

The interest here is to guide the movement relative to the global referential $x{-}y{-}z$ (high-level). This can be achieved in a few ways, the one adopted in this work is to control from a conversion model by sending control signals at accelerations in x, y, z, represented by $\ddot{\mathbf{x}} = [\ddot{x}\ \ \ddot{y}\ \ \ddot{z}]^\intercal$. In this concept, the controller will be given by

$$\mathbf{u}_{1,\ 2,\ 3} = \mathbf{u}_d = \ddot{\mathbf{x}}_d + \mathbf{K}_d \tanh\left(\dot{\tilde{\mathbf{x}}}\right) + \mathbf{K}_p \tanh\left(\tilde{\mathbf{x}}\right) \tag{3}$$

as shown in Figure 2. The intern gains ($\mathbf{K}_2$ and $\mathbf{K}_4$) are equal to 1. $\mathbf{K}_d$ and $\mathbf{K}_p$ will be exploit by the simulation scenarios to tune the parameters space as details in Section 3.2. We set the $\dot{u}_\psi$ control signal to zero ($\mathbf{u}_4 = 0$).

In this scenario, the robot communicate with a GCS using a wireless link, to send and receive information (see Figure 2). The control signal output is a reference acceleration, which will be executed by the robot in simulation ($\ddot{\mathbf{x}} = \mathbf{u}_d$). From the control

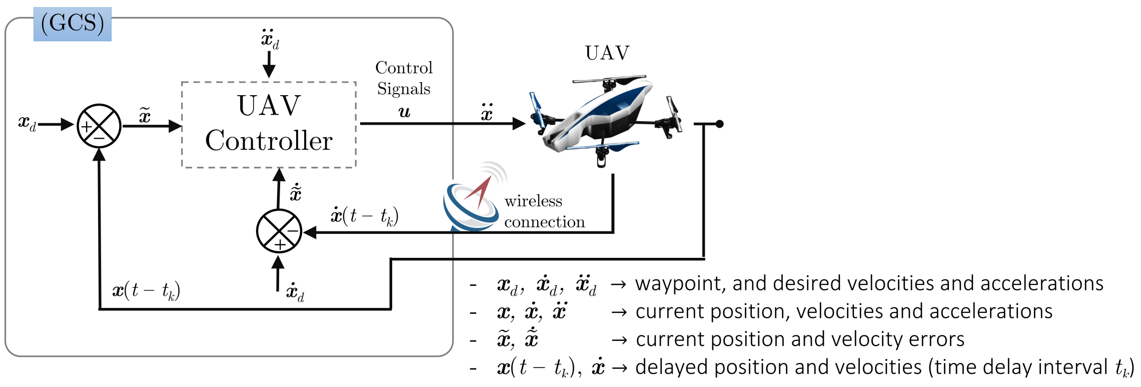

(GCS)
$\ddot{x}_d$
UAV Controller
Control Signals u
$\ddot{x}$
UAV
x_d
$\tilde{x}$
$\dot{\tilde{x}}$
$\dot{x}(t-t_k)$
wireless connection
$x(t-t_k)$
$\dot{x}_d$
- x_d, $\dot{x}_d$, $\ddot{x}_d$ → waypoint, and desired velocities and accelerations
- x, $\dot{x}$, $\ddot{x}$ → current position, velocities and accelerations
- $\tilde{x}$, $\dot{\tilde{x}}$ → current position and velocity errors
- $x(t-t_k)$, $\dot{x}$ → delayed position and velocities (time delay interval t_k)

Figura 2. Controller Model send/receive data to/from the Drone after delay t_k.

commands, the drone's velocity and position data will be calculated from numerical integration, considering $t_s = 1/30$ [s] as the robot sample time, and t_k the observed time delay. To ensure generalizability of the results, delay intervals proportional to the sampling period of the simulated air vehicle (*ArDrone 2.0 Parrot*) were selected. According to the system discretization, this delay in receiving and sending information will be represented by n (or "$idDelay$" in the Algorithm 1), a fixed integer number of samples stored in a data table, $t_k = nt_s$. The quadrotor *delay* is internal with a delayed input in the controller equation, and can be induced by the data table containing the UAV's states over time.

The Figure 2 illustrates the control loop used, with the reference position $\mathbf{x}_d$ (constant for the positioning task, and time-varying for the trajectory tracking task), $\dot{\mathbf{x}}$ and $\ddot{\mathbf{x}}$ representing the drone's velocity and acceleration. The insertion of time delay is done from the knowledge of previous data from the robot. These will be used in the error calculation (controller input). Resulting then in a delayed controller. Note that the controller structure is obtained to maintain a critically damped response [Brandao et al. 2013].

Algorithm 1: Time Delay Insertion in Data Acquisition

```
Data: $idDelay \in \mathbb{N}$, Hist ;                          /* data matrix */
      $k$ ;                                               /* Initialize Counter */
Result: Returns the delayed data from the Quadrotor as controller input
while $t < t_{max}$ do
    if $k > idDelay$ then
        $\mathbf{x} \leftarrow$ Hist.Position$(k - idDelay)$;
        $\dot{\mathbf{x}} \leftarrow$ Hist.Velocity$(k - idDelay)$;
    end
    $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_d - \mathbf{x}$, $\dot{\tilde{\mathbf{x}}} \leftarrow \dot{\mathbf{x}}_d - \dot{\mathbf{x}}$;
    Inserts the delayed input into the controller $(\mathbf{u}_d)$;
    $\ddot{\mathbf{x}} \leftarrow$ Control Signal;
    Numerical Integration to obtain the Reference State;
    UAV Dynamic Model;
    Calculation of the *Performance Indices*;
    Hist $\cup$ Current Drone Data;
    $k \leftarrow k + 1$;
end
```

The Algorithm 1 has been implemented to get the delayed data from the quadrotor. Note that given a *idDelay*, representing an index of the data history matrix, the quadrotor controller will receive delayed information only when the elapsed time is greater than the delay, otherwise the robot will receive updated information. Initially, the gains of the controllers were adjusted to maintain a statistical difference of less than 5%. Next, the controllers were compared through the ITAE, whose objective was to analyze the convergence time to the desired position and the energy expenditure for mission accomplishment.

It is important to note that the control selection did not aim at mitigating or overlapping the effects caused by time delay. The choice of the control structure was due to its wide application and ease of manipulation, as well as the fast response in practical application and simple understanding.

3.2. Controller Tuning in Parameters Space

Control design from parameter space is a widely used approach in robust control applications. The technique consists in designing a control system to meet some stability and/or performance specifications, performing a study on how the controlled plant will behave with respect to controller parameter variations [Zhu and *et al* 2018, Ma et al. 2020]. The goal is to find a controller that behaves satisfactorily even if anomalous system variations occur. In robust control applications, both control and plant parameters are taken into account, which typically includes controller and model variations, quantization effects, and sensor faults.

The objective here differs slightly from the objectives of robust control design; however, a similar procedure is applied. The goal is to analyze the nonlinear control parameter space used for a set of predefined convergence and performance requirements and to verify that the currently used control gains are within or near the intersection of the specifications when applied to the studied quadrotor model.

In order to perform this analysis, a grid search was performed in a region around the currently used gains. Multiple combinations of proportional and derivative gains were tested to see if the resulting closed loop system would meet the specifications for given time delays. The requirements chosen are:

1. The performance indices IAE and ITAE must be up to 30 and 50 times lower, than those obtained in the case without delay;
2. The maximum obtained *overshoot* must be smaller than the initial error.

Such choices were made to prevent selecting gains that generate initial errors large enough to make the task unfeasible, so that this is limited. Also, ignore cases where the robot does not reach the point, or has very slow motion, or reaches the desired point and oscillates with undesirable amplitudes. The gain variation intervals were chosen based on the behavior of the analyzed system and, mainly, the controller responses in order to avoid saturation of the control signals.

3.3. Analytical Convergence and Stability Check

Stability will be verified by observing the convergence of the quadrotor's state when performing the task, graphically. The situations in which the drone presents oscillatory behavior of constant and divergent amplitude will be considered unstable. On the other hand, the responses with underdamped and overdamped behavior will be considered stable, regardless of the time it takes for the drone to reach the permanent regime.

3.4. Performance Analysis

The performance of the proposed controllers is measured according to some performance indices widely known in the literature. This section presents the analysis as a function of the indices IAE (*Integral Absolute Error*) and ITAE (*Integral Time-weighted Absolute Error*), defined by

$$\text{IAE} = \int_0^{t_f} ||\tilde{\mathbf{x}}||\,\mathrm{d}t \tag{4}$$

and

$$\text{ITAE} = \int_0^{t_f} t||\tilde{\mathbf{x}}||\,\mathrm{d}t, \tag{5}$$

these indices indicate how the error has accumulated over time. The first provides information about how fast the robot reached the desired point. The second, evaluates the response on a permanent regime. If the robot oscillates when it reaches the point, a higher rate will be observed, compared to robots that converge with higher damping. This error multiplied by the time *t* will be computed and ITAE will get higher and higher for larger oscillations (permanent regime errors). Both are calculated considering the vector containing the robot's position errors for positioning tasks [Neto et al. 2019].

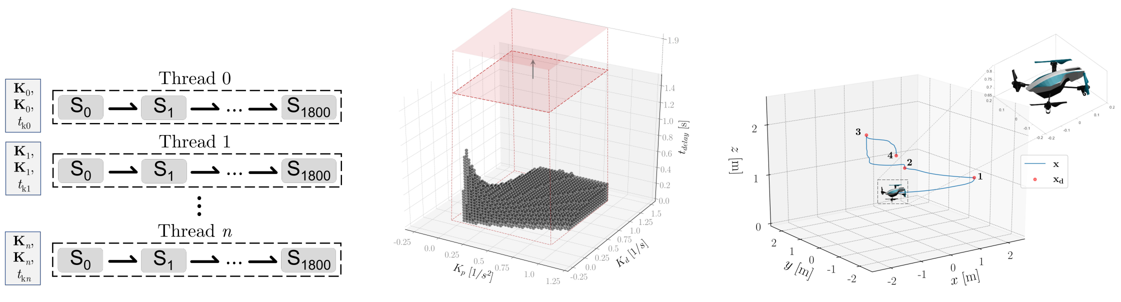
To conclude on the performance of the controllers, the IAE and ITAE metrics for the non-delayed drone performing the same task were taken as a basis.

4. High-performance GPU Implementation

We first implement the robot model and the controller in MATLAB©. Although the high-level MATLAB programming model simplifies the tasks, the long computing time and a proprietary tool are the main drawbacks of this approach. Generic GPU toolboxes [Zhang et al. 2011] have shown performance improvements. A MATLAB© GPU toolbox for CBCT image reconstruction was presented in [Biguri et al. 2016], where there is a performance slowdown due the GPU encapsulation inside MATLAB©, and the authors suggest to write in C++/CUDA directly to improve the computation time. We have adopted this strategy to maximize the performance gains. In addition, our implementation is available in Google Colab format. Therefore, we provide documentation, an open source approach, graphical output (to visually exhibit the simulation results and important graphics), and it makes running much easier and faster while sharing reproducible results.

In proposed notebook, we implement the well know AuRoRA simulator[2] using C/C++ and CUDA languages. AuRoRa framework is able to simulate aerial and ground robots based on its dynamic and kinematic models, whose movements occur according an appropriate control law. All attributes of the robots can be easily modified if desired or even if another type of mobile robot with specific characteristics is used. The simulator allows to implement and evaluate strategies of the most different natures, including heterogeneous and homogeneous robot cooperation, computer vision, obstacle avoidance,

[2]https://github.com/NERO-UFV/AuRoRA



(a) (b) (c)

Figura 3. (a) Task parallelism, one task per thread; (b) Parameters space visualization; (c) Route performing by the ArDrone.

and load transportation. It can be done in the simulation platform and also run in real environments with the physical robots in the laboratory.

To implement the time-delayed control in parameters space, our approach consider that each GPU thread executes an entire simulation scenario as shown in Figure 3(a), where we explore the task parallelism since each thread has a different parameter $\mathbf{K}_d$, $\mathbf{K}_p$, and t_{delay}, as details in Section 3.2, that makes up the kernel code on the GPU. The gains that are currently being adopted and the range where the grid search was performed are: $\mathbf{K}_p \in (0.05, 0.75)$ [s^{-2}], $\mathbf{K}_d \in (0.05, 1)$ [s^{-1}], and $t_{delay}(\equiv t_k) \in [0, 1920]$ [ms]. The GPU kernel requires 158 registers per thread (which includes the number of variables and constants, per simulation) and the code size has around 40 thousand instructions (or 640KiB), and it should be read for L_2 cache which slowdown the execution time. However, the current implementation reports impressive speedups. Figure 3(b) shows the simulation result visualization generated inside the proposed Google Colab notebook. All $\mathbf{K}_d$ and $\mathbf{K}_p$ values in the solid surface are valid parameters as a function of t_{delay}, which can provide a safe and smooth navigation to complete the given mission. Finally, Figure 3(c) depicts the 3-D displacement performing by the ArDrone for a given value of $\mathbf{K}_d$, $\mathbf{K}_p$, and t_{delay}. In this sense, the robot is required to perform a positioning task with four points, named as **1**, **2**, **3** and **4**, where the robot needs to complete the entire mission in 60 [s]. The continuous blue curve represents the path performed by the robot during the simulation time and the red points are the waypoints that the UAV must to reach.

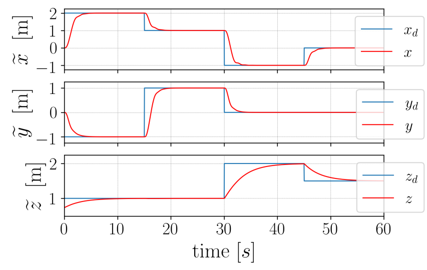
For more details of our CPU and GPU implementation and their visualization, please refer to public available Colab notebook[3].

5. Experiments and Simulated Results

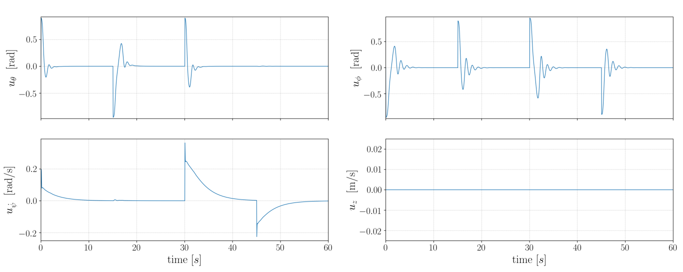
In order to display the tool's functionality, the drone's first mission is to reach a sequence of four points, defined in Figure 3(c) as $\mathbf{x}_i$, with $i = 1, 2, 3, 4$. In this figure is presented the route performed by the UAV in the task accomplishment. It is possible to verify that the robot reaches the desired points in the defined order, completing the mission. Other features of the tool are shown in Figure 4(a), in which the temporal variation of the UAV position in the three axis $[x,\ y,\ z]^\intercal$ (in blue), relative to the references provided to the control system $[x_d,\ y_d,\ z_d]^\intercal$ (in red), is shown. We can observe that the robot reach

[3]`https://colab.research.com/UAV_DelayedControl`

the desired point $\mathbf{x}_{d_i}$ rapidly and stay there waiting for the next task, the next reference point $\mathbf{x}_{d_{i+1}}$. Similarly, the control signals sent by GCS to the robot ($\mathbf{u}_d$) show the smooth stability of the system and asymptotic convergence of the commands that trends to zero as the position error becomes small enough or close to null (see Figure 4(b)).



(a)



(b)

Figura 4. Positioning task (a) temporal variation of the robots' position, and (b) control signals sent to the UAV.

5.1. Time-Delayed Control Analysis in Parameters Space

Firstly, the task given in the simulations requires the drone to reach the waypoint $\mathbf{x}_d = [2 \;\; -1 \;\; 1]^\intercal$ [m]. The robot will start from its initial position ($\mathbf{x} = [0 \;\; 0 \;\; 0.75]^\intercal$ [m]). The controller will at first receive a large error, and the control signal should be high. As the robot gets closer to the desired point, the control signal will get smaller, tending toward zero, where the quadrotor should stabilize until the position and velocity errors are zero. Based on the insertion of fixed delay intervals in the mission accomplishment, simulations were developed to estimate the acceptable delay time limit, for operation considered stable. The analysis was performed from performance indices (IAE and ITAE) of the delayed controller, as well as the position errors of the robot, comparing them to the responses of a drone without time delay.

The results obtained in this analysis can be seen in Figure. 3(b), in which the black balls represents the $[\mathbf{K}_p,\; \mathbf{K}_d,\; t_{delay}]^\intercal$ combination that satisfy the conditions proposed in Section 3.2. In the generated volume we can guarantee the asymptotic and smooth robot convergence for a positioning task.

5.2. Implementation Strategies Evaluation

Now, we are interested in analyze the proposed implement methods in comparison with the basis implementation, which spend much computational time to execute the entire analysis. We evaluate the performance of the proposed implementation in six platforms. First, as baseline, we measure the execution time in MATLAB© for 65536 simulation scenarios for different values of $[\mathbf{K}_p,\; \mathbf{K}_d,\; t_{delay}]^\intercal$. This implementation executes on Windows 10 in an AMD Ryzen 7 1700 Eight-Core Processor with a clock of 3.0 GHz, 64 GB of RAM (DDR4), and 16 MB L3 cache memory. The execution time is greater than 9 hours as shown in Table 1. Then, we evaluate a C++ implementation on the same machine under Ubuntu Linux LTS version 20.04, and compiled by using the optimization flag -O3. The C++ implementation is 262× faster than MATLAB©. Next, we evaluate the Google Colab processor, which is an Intel Xeon CPU @ 2.30GHz with a Two-Core processor with 12.6 GB of RAM (DDR4), and 46 MB L3 cache memory. The execution

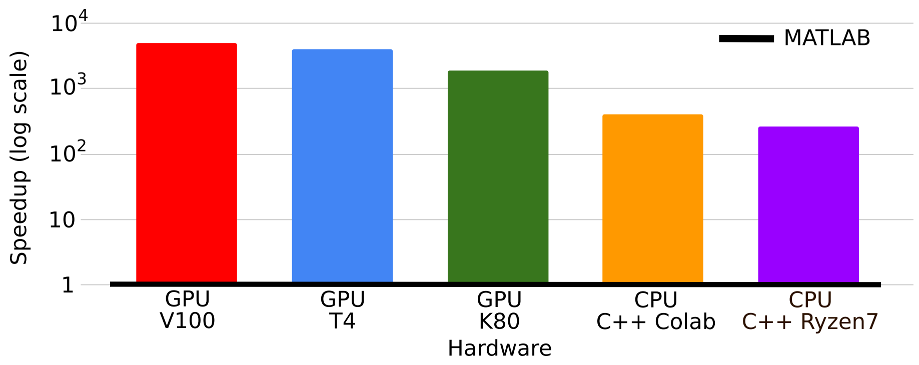


Figura 5. Speedup factor in Execution Time in Comparison to MATLAB©.

time is reduced to 1 minute and 20 seconds, which is 406× faster than MATLAB©. Finally, we evaluate our GPU implementation in three devices. First, we examine the two GPUs from Google Colab: T4 and K80, which reaches a speedup factor of 4070× and 1907×. In comparison to the C++ CPU implementation, the GPU T4 implementation is 15.5× faster. In addition, we also run on a V100 GPU by using the instance P3 AWS Amazon.

Tabela 1. Execution Time considering 65536 simulation scenarios for 6 implementations: GPUs T4, K80 and V100, CPU Xeon, CPU Ryzen, and MATLAB.

	Google Colab					**AMD Ryzen 7**		**Amazon**	
	T4 GPU		K80 GPU		Xeon	CPU		V100 GPU	
	Kern	Total	Kern	Total	C++	C++	MATLAB	Kern	Total
Time [s]	5.9	8.2	16.2	17.5	82.1	127.2	9h 16m 11s	5.2	6.8
					Execution Time per Instance				
Time [ms]	0.09	0.12	0.24	0.26	1.2	1.9	509.2	0.07	0.10

Table 1 summarizes the execution time results. Columns **kern** and **total** depicts the execution time for the GPU Kernel, and the total execution time including the data transfer beteween the CPU/GPU. The first line shows the total execution time in seconds for 65536 simulation scenarios. The GPU implementations fires 512 blocks of 128 threads each. The second line shows the execution time in milliseconds for one single scenario for a given value of $[\mathbf{K}_p,\ \mathbf{K}_d,\ t_{delay}]^\intercal$. Finally, Figure 5 depicts the reached speedup in comparison to the baseline implementation on MATLAB© in log scale. The GPU improves the execution time in more than three orders of magnitude.

6. Conclusion

This work introduces a unified, scalable and replicable approach to make implementation of the autonomous system on a new vehicle faster while preserving its autonomous navigation performance under time delayed communication effects. The simulations were performed in order to verify the response of the delayed nonlinear controller, taking into account the actual execution sequence of the closed-loop process, as well as other aspects such as the data acquisition and actuation system (inertia, *hardware* and *software* limitations, sampling time). The estimation of the acceptable time-delay limit is performed from the analysis of performance indices (IAE e ITAE).The data are compared in relation to the actuated drone, without the presence of time delay.

From the analysis of the effects of inserting input time delay and varying two of the control parameters, in a three-dimensional search space by analyzing the stable convergence of the robot states, the interest here is to estimate the acceptable delay limit for a quadrotor. In this sense, the UAV's navigation system will be aware of delayed data. This information will be sent to the controller and then take a delayed control action. Finally, we present the calibration strategy for the control parameters given time delay interval.

The exploration of the analyses presented in this paper is of utmost importance to try to predict the behavior of the robot and the control system due to the time delay effects inherent in practice. However, the analysis can be costly and slow if performed sequentially, especially for more complex models and with a larger number of parameters in the control system. The proposal presented here is to transpose the code in MATLAB$^{©}$ to its "handmade" version and then parallelize the code to run on GPU as a high performance strategy.

The performance gain shown offers new possibilities and perspectives for the scientific community. Despite not being a code optimized for GPU execution, it has the advantage of being in Colab with documentation and free access, open source, without proprietary software. The framework can also be used in remote classes in pandemic for being a versatile, interactive and fast tool with the possibility of displaying examples in real time, evaluating more than 60,000 simulations in a few seconds.

As suggestions for future works, it can be proposed an optimization of the codes for the GPU, seeking the maximum use of the potential of the different architectures presented here and the evaluation of changing the two others control parameters ($\mathbf{K}_2$ and $\mathbf{K}_4$). Furthermore, the analysis can be extended to other tasks, such as, for example, trajectory tracking and path following, besides being possible to study other types of robots and controller models.

Acknowledgment

This work was carried out with the support of FAPEMIG, CNPq, and the Coordernação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Referências

Allahverdy, D., Fakharian, A., and Menhaj, M. B. (2019). Back-stepping integral sliding mode control with iterative learning control algorithm for quadrotor UAVs. *Journal of Electrical Engineering & Technology*, 14(6):2539–2547.

Biguri, A., Dosanjh, M., Hancock, S., and Soleimani, M. (2016). TIGRE: a MATLAB-GPU toolbox for CBCT image reconstruction. *Biomedical Physics & Engineering Express*, 2(5):055010.

Brandao, A. S., Filho, M. S., and Carelli, R. (2013). High-level underactuated nonlinear control for rotorcraft machines. In *2013 IEEE International Conference on Mechatronics (ICM)*. IEEE.

Han, L., Dong, X., Li, Q., and Ren, Z. (2017). Formation tracking control for time-delayed multi-agent systems with second-order dynamics. *Chinese Journal of Aeronautics*, 30(1):348–357.

Ji, Y. and Kang, Z. (2020). Three-stage forgetting factor stochastic gradient parameter estimation methods for a class of nonlinear systems. *International Journal of Robust and Nonlinear Control*.

Koksal, N., An, H., and Fidan, B. (2020). Backstepping-based adaptive control of a quadrotor UAV with guaranteed tracking performance. *ISA Transactions*, 105:98–110.

Lv, F., He, W., and Zhao, L. (2020). A multivariate optimal control strategy for the attitude tracking of a parafoil-UAV system. *IEEE Access*, 8:43736–43751.

Ma, F., Wang, J., Yu, Y., Wu, L., Liu, Z., Aksun-Guvenc, B., and Guvenc, L. (2020). Parameter-space-based robust control of event-triggered heterogene-ous platoon. *IET Intelligent Transport Systems*, 15(1):61–73.

Neto, V. E., Sarcinelli-Filho, M., and Brandao, A. S. (2019). Trajectory-tracking of a heterogeneous formation using null space-based control. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE.

Piskorski, S. and Brulez, N. (2012). Ar. drone developer guide parrot. sdk version 2.0. *tech. rep.*

Rabelo, M. F. S., Brandao, A. S., and Sarcinelli-Filho, M. (2018). Centralized control for an heterogeneous line formation using virtual structure approach. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. IEEE.

Santana, L. V., Brandao, A. S., and Sarcinelli-Filho, M. (2015). Outdoor waypoint navigation with the AR.drone quadrotor. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE.

Santana, L. V., Brandão, A. S., and Sarcinelli-Filho, M. (2016). Navigation and cooperative control using the AR.drone quadrotor. *Journal of Intelligent & Robotic Systems*, 84(1-4):327–350.

Tang, Y.-R. and Li, Y. (2015). Dynamic modeling for high-performance controller design of a UAV quadrotor. In *2015 IEEE International Conference on Information and Automation*. IEEE.

Tang, Y.-R., Xiao, X., and Li, Y. (2017). Nonlinear dynamic modeling and hybrid control design with dynamic compensator for a small-scale UAV quadrotor. *Measurement*, 109:51–64.

Zhang, B., Xu, S., Zhang, F., Bi, Y., and Huang, L. (2011). Accelerating MatLab code using GPU: A review of tools and strategies. In *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*. IEEE.

Zhu, S. and *et al* (2018). Parameter space and model regulation based robust, scalable and replicable lateral control design for autonomous vehicles. In *2018 IEEE Conference on Decision and Control (CDC)*. IEEE.