LEONARDO ALVES FAGUNDES JUNIOR

CLOUD4AURORA: AN OPEN AND INTERACTIVE FRAMEWORK FOR UAV SIMULATION

Dissertation submitted to the Computer Science Graduate Program of the Universidade Federal de Viçosa in partial fulfillment of the requirements for the degree of *Magister Scientiae*.

Adviser: Alexandre Santos Brandão

Co-adviser: Vitor Barbosa Carlos de Souza

VIÇOSA - MINAS GERAIS
2022

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
F156c
2022

Fagundes Junior, Leonardo Alves, 1997-
Cloud4AuRoRA: uma estrutura aberta e interativa para simulação de UAV / Leonardo Alves Fagundes Junior. – Viçosa, MG, 2022.
1 dissertação eletrônica (93 f.): il. (algumas color.).

Texto em inglês.
Orientador: Alexandre Santos Brandão.
Dissertação (mestrado) - Universidade Federal de Viçosa, Departamento de Informática, 2022.
Referências bibliográficas: f. 85-93.
DOI: https://doi.org/10.47328/ufvbbt.2022.538
Modo de acesso: World Wide Web.

1. Computação em nuvem - Métodos de simulação. 2. Robótica. 3. Drone. 4. Google Colaboratory. 5. Software - Desenvolvimento. 6. Aprendizado do computador. I. Brandão, Alexandre Santos, 1982-. II. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 22. ed. 004.36

Bibliotecário(a) responsável: Alice Regina Pinto Pires CRB-6/2523

LEONARDO ALVES FAGUNDES JUNIOR

CLOUD4AURORA: AN OPEN AND INTERACTIVE FRAMEWORK FOR UAV SIMULATION

Dissertation submitted to the Computer Science Graduate Program of the Universidade Federal de Viçosa in partial fulfillment of the requirements for the degree of *Magister Scientiae*.

APPROVED: August 26, 2022.

Assent:

Leonardo Alves Fagundes Junior
Author

Alexandre Santos Brandão
Adviser

Este trabalho é dedicado a todas as pessoas que fizeram dos meus dias, únicos.
Em especial, à minha família, aquela com a qual compartilho laços sanguinos e de amor, e aquela que surgiu pelos mais singelos laços de amizade. Amo vocês!

Agradecimentos

Todos os dias este agradecimento se expressa em minhas palavras e ações, mas, essencialmente na escrita desta dissertação, não posso deixar de glorificar a Deus, principalmente pelo dom da vida, pela dádiva de poder acordar e respirar. Agradeço por tudo que me conduziu a este momento, por todas as pessoas maravilhosas que Ele colocou em minha vida e por toda a força que me ajuda a encontrar todos os dias.

À minha família, deixo o meu mais sincero "muito obrigado por tudo!". Aos Fagundes, aos Carvalho e aos Alvim-Santos, deixo aqui registrado que sem vocês nunca teria chegado onde cheguei. Vocês todos são o impulso dos meus voos mais altos. De modo especial, agradeço a meus pais, Leonardo, meu *amigão*, e Adwania, minha *rainha*, meus heróis e guias. Obrigado, mãe, por todo apoio, carinho, preocupação, e por insistir em mim (mesmo quando nem eu mesmo acreditava). Obrigado, pai, por todas as conversas, todos os sermões, por me ensinar a ouvir, por todo apoio e confiança. Ao meu irmão, Leandro, meu *companheiro* de (absolutamente) tudo, agradeço pela força, por estar sempre junto, por toda motivação e colaboração em inúmeros projetos da UFV (sem você, eu não teria feito metade do que fiz!). Te motivar é a missão que mais me motiva durante esta jornada. À minha *princesa*, Amanda, companheira e amiga, por todo apoio, "puxão de orelha" e sorrisos que só você me faz dar. Obrigado por me ensinar a sonhar a vida. Meu maior sonho sempre foi ser um orgulho para cada um de vocês, saber que estou no caminho certo na realização deste desejo me faz sentir agraciado.

Aos professores que marcaram presença em minha vida. Vocês fizeram da minha formação um momento único de aprendizado e crescimento. Todas as oportunidades que me proporcionaram foram muito importantes para a construção profissional e pessoal que adquiri neste tempo. Em especial, agradeço ao Alexandre Brandão, meu orientador - e *inspiração* -, pela atenção e paciência ao ensinar e guiar, pelos momentos de risadas e descontração, por toda motivação, conhecimento e tempo dedicados, pela amizade desenvolvida ao longo desse tempo. Pela incansável determinação de fazer história no Brasil e no mundo, conte sempre comigo!

Aos meus coorientadores Ricardo Ferreira e Vitor Barbosa, por toda confiança, suporte e profissionalismo, por sempre estarem dispostos a me conduzir, mesmo na correria das dezenas de projetos, disciplinas e artigos. Agradeço por dedicarem ao menos uma hora toda semana para conversarmos e colocarmos as atividades e os resultados em dia. Ainda temos muito o que fazer (e publicar), mas espero que este tempo que passamos juntos tenha sido confortável e produtivo para vocês, com certeza tem sido muito gratificante para mim. Agradeço a vocês por me orientarem não somente na pesquisa, mas também na vida pessoal, pela generosidade e empatia.

Agradeço aos bons amigos que fiz em toda esta jornada de aprendizagem técnica e científica, de crescimento contínuo. Aos times do Robótica na UFV: ao Núcleo de Especialização em Robótica (NERo), ao Believe, Do and Play (BDP) e ao AstroBot, ambientes de acolhimento, compartilhamento e muita disposição para colaborar. Em especial, ao Celso, *monitor da natação*, pela parceria mais determinada a "quebrar a marreta" e fazer engenharia de verdade em todos os detalhes, do projeto mais simples ao mais avançado. Momentos difíceis e complicados, madrugadas em claro, experimentos dando errado, robôs quebrando e batendo na parede, no teto, horas olhando para a tela de um computador para achar um erro de sinal (ou fazer aquela figura de respeito).

Ao Daniel Dourado, pela amizade e auxílio no crescimento profissional, acadêmico e pessoal, por estar sempre disposto a discutir as matemáticas e os detalhes dos robôs (não fugindo da marreta, está tudo bem!). Apesar de me "abandonar" para ajudar as "equipes inimigas" nas competições do NERo, você sempre "botou fé" em mim, e nunca mediu esforços pra me explicar os detalhes que ninguém sabe. Ao Vinícius "Pacheco", o ROS Master dos meus projetos, sempre disposto a me ensinar cada detalhe do que sabe, além de sentar comigo por horas para fazer códigos que eu, sozinho, levaria semanas para escrever. Sem a ajuda de vocês, muitos dos meus trabalhos não seriam executados, muito obrigado por tudo. Aprendi muito com vocês, e espero continuar aprendendo. O apoio de vocês foi essencial para que eu chega-se até aqui.

Às minhas amigas do Laboratório de Análises Biomecânicas (LAB). À Amanda Silvatti, minha mais nova coorientadora, por me receber no LAB com todo amor, carinho e toda paciência. Por confiar e acreditar em meu potencial mesmo me conhecendo há tão pouco tempo. Daqui para frente, ninguém nos segura. "Ao infinito e além!". À Nara, Ana, Luíza, Isabella, obrigado pelo apoio, pelas risadas e conversas.

Finalmente, agradeço aos professores do departamento e da UFV, por todo conhecimento e suporte. Agradeço também aos demais funcionários, da UFV e terceirizados, por toda estrutura oferecida, por cada sorriso e conversa, pelo ambiente limpo e agradável, pela comida feita com amor e carinho, e pelos jardins mais floridos de toda a cidade. A dedicação de vocês é essencial para este e todos os trabalhos realizados na universidade.

Agradeço ao CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, e à FAPEMIG - Fundação de Amparo à Pesquisa do Estado de Minas Gerais, pelo apoio dado a esta pesquisa. Agradeço também à CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, pela bolsa que me foi concedida, permitindo dedicação exclusiva aos estudos de Mestrado.

E todos que não citei, afinal são muitos nomes, serei sempre grato.

Leonardo Alves Fagundes Júnior

"Part of the Journey is the End."
(Tony Stark)

Abstract

FAGUNDES, L. A., Jr., M.Sc., Universidade Federal de Viçosa, August, 2022. **Cloud4AuRoRA: an Open and Interactive Framework for UAV Simulation**. Adviser: Alexandre Santos Brandão. Co-adviser: Vitor Barbosa Carlos de Souza.

To verify solutions before implementing them in a robot, robots and developers need an experimentation platform that accurately reproduces the real-world environment as well as the robot's physical interactions with that environment. Through simulation, they can evaluate the robot's performance in terms of localization, planning, and motion control. Over the past ten years, most of the progress in robotics developed in academia has been made using well-known software such as MATLAB/Simulink, Robot Operating System (ROS), Copelia Sim, and Gazebo. However, learning to use and program these platforms can be challenging for students. Nonetheless, the current body of literature does not provide comprehensive frameworks to adequately address this issue in conjunction with learning about modeling (kinematics and dynamics) and controlling the unmanned aerial vehicle (UAV) to perform different types of missions. This dissertation presents a UAV simulation tool for learning robotics on Google Cloud Platform for developing control strategies, parameter tuning, and evaluation on different types of tasks such as positioning, trajectory planning, trajectory tracking, and robot cooperation. The proposed framework, called cloud4AuRoRA, can also contribute to advances in scientific research in aerial robotics, specifically UAV control and navigation, and machine learning applications. Following the framework of the AuRoRA platform, developed in MATLAB, the approach proposed here merges robot modeling and control using C/C++, and data storage and display using Python. The proposed framework is intended to facilitate the development and validation of studies in aerial robotics for students and researchers by improving results and proof-of-concept development time, aided by the high computational performance provided by Google Collaboratory. Cloud4AuRoRA showed a 70 times performance improvement compared to MATLAB when running the same routine, through direct conversion/translation between programming languages. Finally, it is worth mentioning that could4AuRoRA is adaptable for implementation in a GPU environment through modifications to the core code of the platform.

Keywords: Aerial Robotics. Simulation on the Cloud. Google Colaboratory. Modeling and Control. Learning Technology. Student Assessment.

Resumo

FAGUNDES, L. A., Jr., M.Sc., Universidade Federal de Viçosa, agosto de 2022. **Cloud4AuRoRA: uma Ferramenta Aberta e Interativa para a Simulação de UAV**. Orientador: Alexandre Santos Brandão. Coorientador: Vitor Barbosa Carlos de Souza.

Para verificar soluções antes de implementá-las em um robô, roboticistas e desenvolvedores precisam de uma plataforma de experimentação que reproduza com precisão o ambiente do mundo real, bem como as interações físicas do robô com esse ambiente. Através de simulação, eles podem avaliar o desempenho do robô em termos de localização, planejamento e controle de movimento. Nos últimos dez anos, a maior parte dos progressos em robótica desenvolvidos na academia foi feita usando softwares bem conhecidos, tais como MATLAB/Simulink, Robot Operating System (ROS), Copelia Sim e Gazebo. Entretanto, aprender a usar e programar estas plataformas pode ser um desafio para os estudantes. Contudo, o corpo atual de literatura não fornece estruturas abrangentes que permitam abordar adequadamente esta questão em conjunto com o aprendizado sobre modelagem (cinemática e dinâmica) e controle do veículo aéreo não-tripulado (VANT) para realizar diferentes tipos de missões. Esta dissertação apresenta uma ferramenta de simulação de VANTs para aprendizagem de robótica na plataforma Google Cloud, para o desenvolvimento de estratégias de controle, ajuste de parâmetros e avaliação em diferentes tipos de tarefas, tais como posicionamento, planejamento de caminhos, rastreamento de trajetórias e cooperação entre robôs. A estrutura proposta, denominada cloud4AuRoRA, também pode contribuir com avanços na pesquisa científica em robótica aérea, especificamente em controle e navegação de UAV, e aplicações de aprendizagem de máquinas. Seguindo a estrutura da plataforma AuRoRA, desenvolvida em MATLAB, a abordagem aqui proposta mescla modelagem e controle de robôs usando C/C++, e armazenamento e exibição dos dados usando Python. A estrutura proposta destina-se a facilitar o desenvolvimento e validação de estudos em robótica aérea para estudantes e pesquisadores, melhorando os resultados e o tempo de desenvolvimento da prova de conceito, assistida pelo alto desempenho computacional fornecido pelo Google Colaboratory. A cloud4AuRoRA apresentou melhoria de desempenho em 70 vezes, comparado ao MATLAB ao executar a mesma rotina. Por fim, vale ressaltar que a could4AuRoRA é adaptável para implementação em ambiente de GPU, mediante modificações no código principal da plataforma.

Palavras-chave: Robótica Aérea. Simulação na Nuvem. Google Colaboratory. Modelagem e Controle. Tecnologia de Aprendizagem. Avaliação do Aprendizado.

List of Figures

List of Tables

List of abbreviations and acronyms

AuRoRA	Autonomous Robots for Research and Application
Colab	Google Colaboratory notebook
DLR	Deutsches Zentrum für Luft- und Raumfahrt (German Aerospace Center)
ESA	European Space Agency
GPU	Graphics processing units
HMI	human machine interface
HPC	high performance computing
HRI	human robot interface
ISS	International Space Station
NERo	Núcleo de Especialização em Robótica (Robotics Specialization Center)
CUDA	NVIDIA Compute Unified Device Architecture
O2	Ocean One
UFV	Universidade Federal de Viçosa
UAV	Unmanned aerial vehicles
UGV	Unmanned ground vehicles
VANT	Veículo aéreo não-tripulado (Unmanned aerial vehicle)

Contents

1 Introduction

Robots have become an indispensable tool for general industrial, due its capability of efficiently and accurately performing a range of repetitive and often unsafe tasks. As a consequence, the rapid progress in the implementation of robots to solve common day-to-day problems is becoming more and more evident, showing its potential as well as the various benefits associated with the insertion of such technology in society. The rapid technological advance and the growing interest of worldwide researchers and financiers in the various fields of robotics also contribute to such improvements.

Currently, mobile robotics (classified as aquatic, aerial or terrestrial) and industrial robotics (consisting of manipulators/arms) stand out for advanced applications in sectors such as: the medical/hospital sector (LOPEZ et al., 2022; BERNHARD et al., 2022; SEIBOLD et al., 2019; MILLAN et al., 2021), especially in patient care and/or assisting doctors and nurses during surgical procedures; the precision farming industry (LYTRIDIS et al., 2021; KIM et al., 2019; QUAGLIA et al., 2020), in the application of agro-chemicals, in the mapping, handling and planting of various cultures such as corn and soy, and in the inspection of the product in an autonomous way; human-robot interaction (HENSCHEL; HORTENSIUS; CROSS, 2020; CHOI et al., 2020; HE et al., 2020), for hotel reception and service, entertainment, and education; in search and rescue incidents (DREW, 2021; QUERALTA et al., 2020), assisting rescue teams during natural disasters and accidents by quickly locating human life; the sector of conservation and marine and archaeological exploration (KAPETANOVIĆ et al., 2020; JOHNSTON, 2019), with robots capable of reaching depths at pressures up to ten times normal pressure; space exploration sector (KALITA; THANGAVELAUTHAM, 2020; HAMBUCHEN; MARQUEZ; FONG, 2021; SCHMAUS et al., 2018; KALITA; THANGAVELAUTHAM, 2020), with robots on Mars collecting soil samples and conducting space research, capable of basic tasks such as recharging their batteries, cleaning equipment, positioning and orienting solar energy plates, among other abilities; and the sector for the preservation of the environment and animal life (ROSS et al., 2021; KOCER et al., 2021), with animal monitoring and forest preservation applications. Some of these applications, already consolidated in the market and advanced research in private and government agencies, are presented in Figure 1.

From these examples, it is clear the advance of robotics and the possible benefits to be achieved with its use in day-to-day life: improvement in the quality of life; higher quality agricultural products and production in increasingly larger scales; greater precision and success rates in hospital surgeries; greater speed and chance of success in search and rescue occurrences; discovery of habitable planets and initiation of cultivation experiments, soil quality, among others.

From manipulators to mobile robots, each architecture has a set of specific attributes carefully designed to perform certain tasks. Manipulator robots used for inspection or cargo transport have characteristics such as: high motion speed; precision and repeatability, able to reach the target point several times; high torque, able to exert forces up to kilo

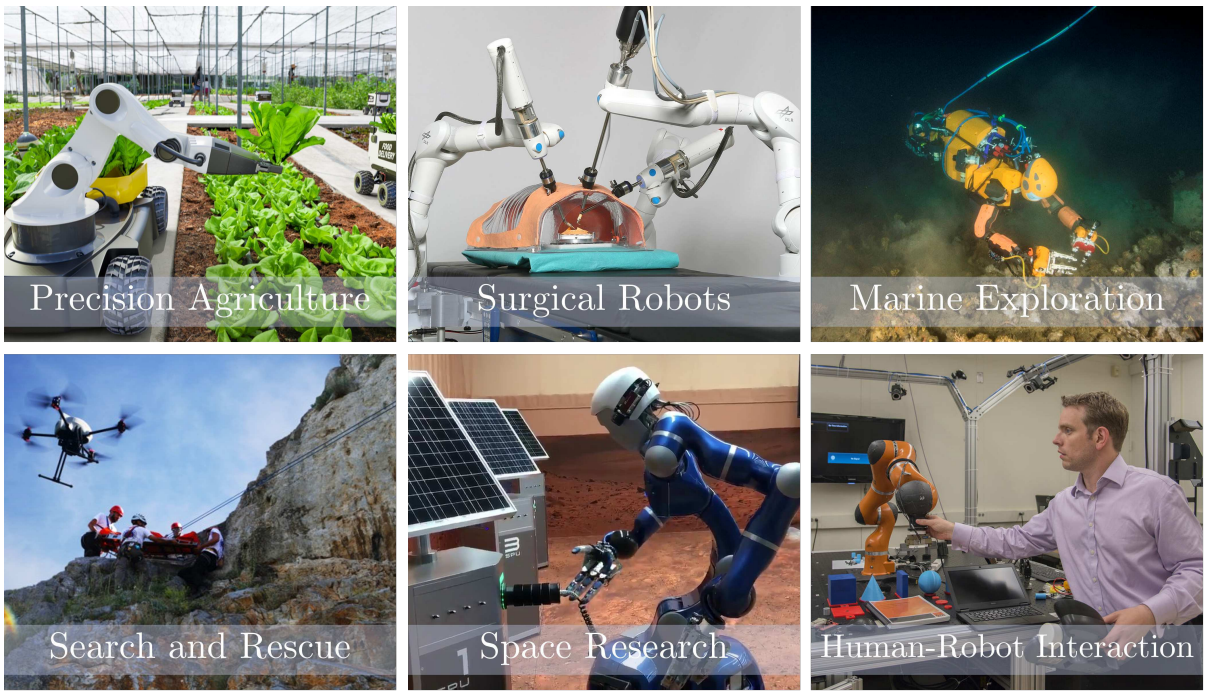


Figure 1 – Robot applications in research and industry. On the **left top**: a mobile manipulator designed for agricultural industry, collecting products, inspecting and maintaining sustainable developments in precision farming; on the **center top**: MiroSurge[a], the DLR's surgical robot operating a chest prototype for healthcare robotics (TOBERGTE et al., 2011; HAGN et al., 2010; KONIETSCHKE et al., 2009); on the **right top**: the Ocean One (O2)[b], from Stanford Robotics Lab, a humanoid underwater robot dexterous enough to replace human divers in carrying out deep or dangerous ocean research (submited to ten times normal pressure) (BRANTNER; KHATIB, 2021; KHATIB et al., 2016; BRANTNER; KHATIB, 2018); on the **bottom left**: the OnyxStar XENA-8F UAV[c] assiting a rescue squad in mountain rescue operatio by providing real-time visual information; on the **bottom center**: the Rollin' Justin[d], a mobile manipulator from DLR's institute of Robotics and Mechatronics, teleoperated/commanded from space in a representative environment for Mars exploration missions, designed for household work and assisting astronauts in space (LEIDNER et al., 2014; SCHMAUS et al., 2019; SCHMAUS et al., 2018; HULIN et al., 2021); on the **bottom right**: human-robot safaly collaboration on NIST[e].

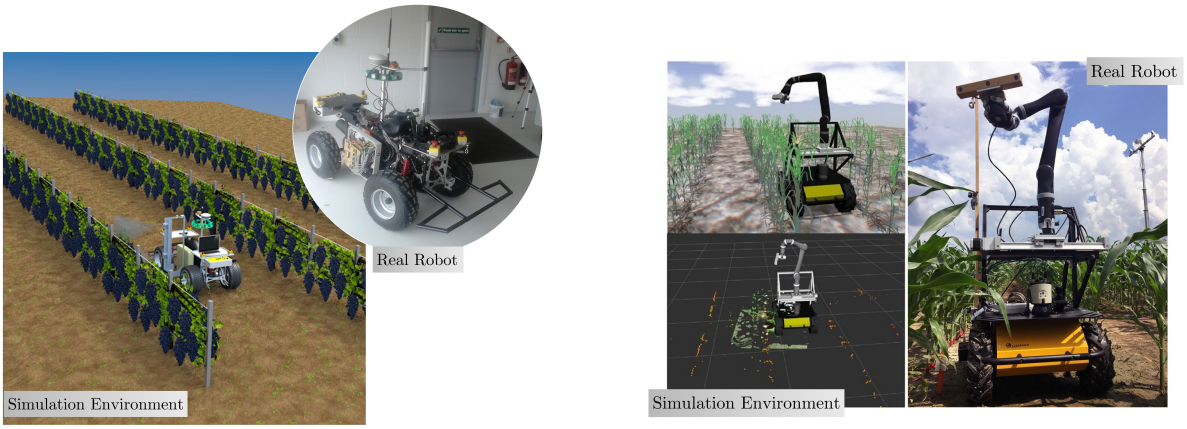
[a] https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3795/16616_read-40529/
[b] https://khatib.stanford.edu/ocean-one.html
[c] https://www.onyxstar.net/search-and-rescue-by-drone/
[d] https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-11427/gallery/35411
[e] https://www.nist.gov/industry-impacts/human-robot-collaboration

Newtons; large working space with a considerable reach; dexterity of motion, allowing the robot to reach a position within its working space with more than one possible orientation; compliance, the robot must be compliant with external forces in order to avoid or minimize damage to the environment (BIEZE et al., 2020); among other capabilities. On the other hand, an aerial robot is characterized by: being adaptable according to the application, able to navigate in a reactive way recalculating the safest and most optimized route according to the task; real-time remote sensing, able to collect large volumes of data in a short period of time; among other attributes (DELAVARPOUR et al., 2021). In this thesis, the focus is on unmanned aerial vehicles (UAVs).

In all mentioned applications, the development of a system capable of being taken to the field is an arduous process until it becomes a marketable product, because it usually involves mathematical and mechanical modeling, motion control, and numerous tests. Mainly aiming to maintain the integrity of the robots and the human beings responsible for operating such devices, usually many of the applications/tests are done through simulators before their experimentation in the real world. Especially in the world of research and education, where robots are used for developing and testing control and cooperation strategies. In fact, despite being an emerging and increasingly explored technology, robots can be expensive technologies for educational institutions, such as universities, and students/researchers sometimes need preliminary training in order to know how to use the device correctly, so the use of 3D simulations to mimic the robot's behavior is widely proposed in the literature to solve these problems (LORETO-GÓMEZ et al., 2019).

Not only in robotics at the teaching/learning or research initiation level, but also advanced researchers are dedicated to implementing their strategies in simulators before application on the real robot, or even before its manufacturing. During the design and modeling phase, a range of simulations are performed. Examples of this type of situation are found in Figure 2. In precision farming, the iRTA program (*intelligent robotic high precision treatment application in rough terrain vineyards*) has a project for developing agricultural robot for precision spraying[1], shown in Figure 2(a), in which a four-wheeled robot was developed with sensory apparatus, for navigation in agricultural plantations, and actuator, for pesticide and fertilizer application. In such project, the simulation environment was crucial for the development and improvement of the robot (equiped with a high-precision, low-drift spraying component) and of strategies to optimize the robot navigation for product application in the plantation (such as autonomous localisation, navigation and obstacle avoidance, in order to enhance its traversability and ensure its safe operation in rough environments). Another project developed for application in the agricultural environment is the mobile manipulator robot, composed of two robotic platforms: Vinobot, presented in Figure 2(b), an autonomous ground vehicle with a manipulator; and Vinoculer, a mobile

[1] https://www.agenso.gr/projects/irta/



(a) Robotic platform for precision spraying in steep slope vineyards.

(b) Robotic platforms for high-throughput field phenotyping.

Figure 2 – Simulation and real robot of research projects for developing agricultural robot.

observation tower. The prototype was designed for autonomous plant phenotyping. The ground vehicle collects data from individual plants, while the observation tower oversees an entire field, identifying specific plants for further inspection by the ground vehicle. Indeed, while real robotic platforms for field phenotyping can only be deployed during the planting season, simulated platforms can help the improvement of the various algorithms throughout the year. In order to do that, the simulation must be designed to mimic not only the robots, but also the field with all its uncertainties, noises and other unexpected circumstances that could lead to errors in those algorithms under real conditions (SHAFIEKHANI et al., 2017; SHAFIEKHANI; FRITSCHI; DESOUZA, 2018). In the aforementioned work, it is described how the target navigation algorithms are being tested and it provides the first insights on the functionality of the simulation and its usefulness for testing those same robotic platforms, as shown in Figure 2(b).

Robots and humans must be able to work together safely and reliably. Besides the development of robots for ground operation, the fields of study in marine and space exploration have shown outstanding results in applications focused on marine archaeology and Mars research, respectively, from simulation to real-world implementations, where the robot is required to interacting with a human operator. In the METERON SUPVIS Justin experiment, a collaboration between the German Aerospace Center (DLR) and the European Space Agency (ESA), teleoperation was demonstrated in order to command intelligent robots on earth from planetary surfaces, in specific from the International Space Station (ISS), by astronauts in Earth's orbit using different human-machine interface (HMI) devices. Rollin' Justin at the DLR Institute of Robotics and Mechatronics in Oberpfaffenhofen receives its instructions from a tablet computer on the ISS. Through a command from the operator in orbit, Justin must act autonomously and use its local and artificial intelligence to decide on how individual workings steps can be carried out, which

can be maintenance, construction, cleaning, object collection and reconnaissance activities (SCHMAUS et al., 2019). The experiments demonstrate the usability of the technology and provide solutions for commanding autonomous robots during space missions. Figure 3 presents one of the applications tested during the experiment. On it, Justin is asked to clean a planar glass surface: a window and a solar panel. In simulations, the task was performed in a virtual environment emulating the tool and the panel. The robot identifies and collects the tool to be used in the process (1), reaches the surface to be cleaned (2), and calculates the optimal route for efficiently cleaning the panel (3). The same strategy is applied to the real robot, which has already been tested and validated in simulation, allowing it to generalize the task to panels of different shapes and orientations, but now by means of a command sent by a control base outside the planet.

Research and development of robots for exploring venues that are inaccessible to humans, or simply inhospitable, has been a longstanding ambition of scientists, engineers, and explorers across numerous fields. The deep sea exemplifies an environment that is

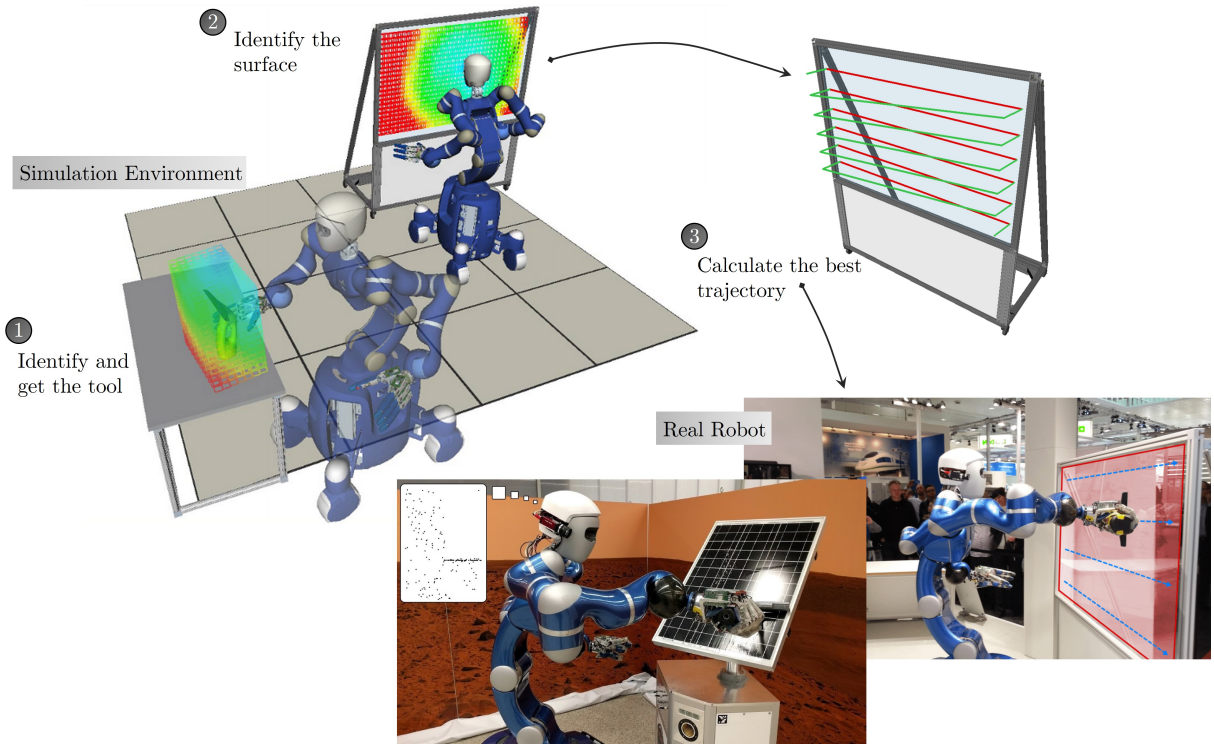


Figure 3 – Simulation and experiment: Rollin' Justin teleoperated from the Earth's orbit during the experiment "SUPVIS Justin" between ISS and Oberpfaffenhofen. The robot is required to clean the electric solar plate, emulating a real space exploration situation on Mars, where recurrent dust storms cause the accumulation of particles on the photovoltaic panel, reducing the direct incidence of ultraviolet rays on the panel cells, making it difficult and even impossible to generate and store electrical energy from this source. (LEIDNER et al., 2014; SCHMAUS et al., 2019; HULIN et al., 2021).

largely uncharted and denies human presence, mainly due to the high pressure found in this unstructured environment. The Ocean One (O2) robot, a humanoid unmanned underwater vehicle (UUV), was developed by Stanford AI/Robotics Laboratory, from Stanford University, as a solution designed specifically for underwater manipulation (BRANTNER; KHATIB, 2021; KHATIB et al., 2016). Ocean One's control architecture enables the deployment of dexterous robotic manipulation to the deep sea, through a collaboration between this humanoid robot and a human pilot. The robot, its control strategies, and teleoperation have been tested in simulation and validated in experiments in real environments: into the Mediterranean Sea, Ocean One explored the Lune, a French naval vessel (see Figure 4) Khatib et al. (2016); and assisting human divers in investigating underwater volcanic structures at Santorini, Greece. Brantner e Khatib (2021) utilizes simulation to clearly isolate and demonstrate the advantages of whole-body control in both free-space and contact situations. In the simulations, the work implements the Ocean One's kinematic and dynamic models, as well as the posture controllers. While interacting with the aquatic environment and the objects, a docking maneuver is performed by using unified motion and force control combined with posture control.

Robots are being deployed into dynamic environments in which the robot's tasks may change frequently or involve human collaborators. The level of complexity in advanced robotics programming and deploying and operating advanced robots, as well as the high

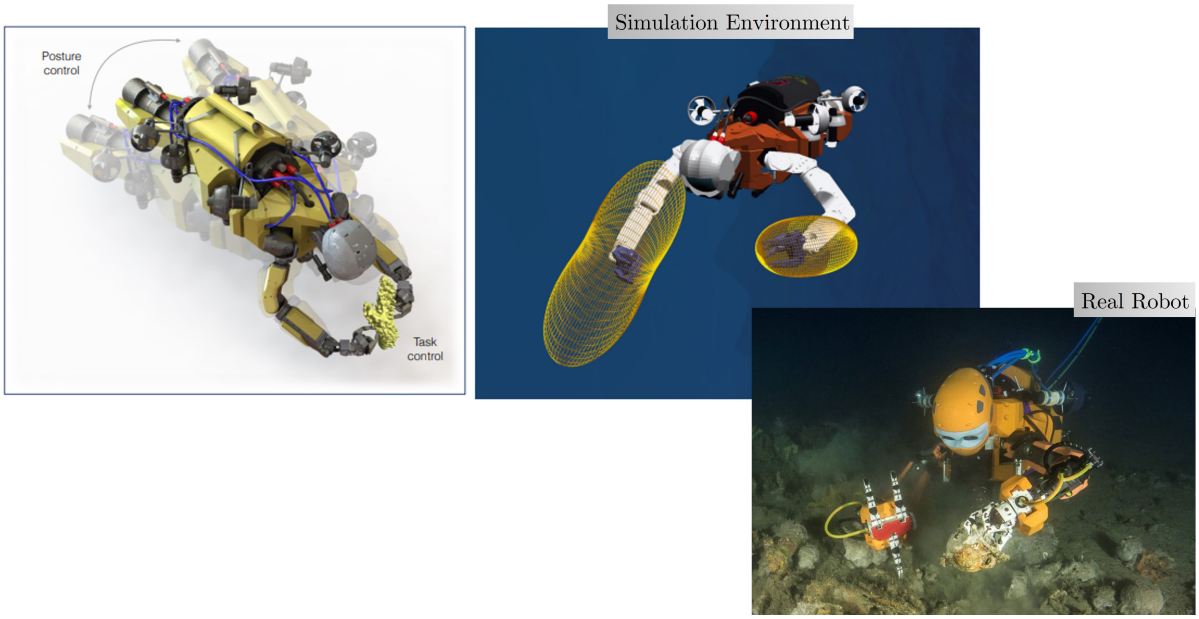


Figure 4 – Simulation and experiment: Ocean One control of posture. The deployment in the experimental field of Ocean One: its maiden mission to the Lune (Louis XIV's two-decked, 54-gun flagship that sunk in 1664 off the coast of Toulon, France), where it inspected the ship remnants and recovered an archeological artifact (BRANTNER; KHATIB, 2021; KHATIB et al., 2016).

cost that would be incurred to debug a robotics system on the field, make advanced robotics simulation a critical component of manufacturing engineering. Simulators are tools used to create application for a physical robot without depending on the actual machine, thus saving cost and time. Thus, the application can be transfered onto the physical robot (or rebuilt) with a minimum of modifications (CAVALCANTI et al., 2019; ZAGAL; RUIZ-DEL-SOLAR, 2007). Simulators use digital representation - a digital twin - to enable dynamic (or just kinematic) interaction with robot's models in a virtual environment, with the purpose of developing real systems much faster and integrating the robot into real-world applications with less errors than conventional engineering, resulting in safer operations (ŽLAJPAH, 2008).

The use of a robotics simulator for development of a control program is highly recommended regardless of whether an actual robot is available or not. The simulator allows for robotics programs to be conveniently written and debugged off-line with the final version of the program tested on an actual robot (CHOI et al., 2021; MARTINEZ-GONZALEZ et al., 2020). The major differences between the interaction of the robot with the environment in real and simulated scenarios are related to obtaining the position/orientation data of the objects in the scene and obtaining the sensory data. In the real world, the interaction of the robot with the environment is easily achieved; however its posture may not be as accurate due to the limitations of the robot's on-board sensors. On the other hand, in simulations, the exact knowledge of the robot's location is not a complex task, but a sensor-based robot actions are much more difficult to simulate and/or to program off-line. Nevertheless, there are well-established simulators that implement such sensors leaving only a high-level interaction with the tool to the user, such as the Gazebo Sim.

According to Choi et al. (2021), Martinez-Gonzalez et al. (2020), Guyot et al. (2011), teaching/learning robotics simulators have several advantages compared with real robots, for example:

a) Accelerate the engineering design cycle and significantly reduce platform costs;

b) Provide access to each student to his/her own robot, during the class;

c) Avoid costly mistakes and accidents, guarding against collisions;

d) Provide an accelerated, safe, and fully controlled virtual testing environment;

e) Facilitate the development of more intelligent robots.

The role of simulation in the development, research and teaching of robotics is well known. Simulation is not a goal in itself but a means to an end. The motivation of this current work is the previous one carried out at *Núcleo de Especialização em Robótica* (NERo), at UFV called Autonomous Robots for Research and Application (AuRoRA[2]

[2] Online repository: https://github.com/neroUFV

platform) (PIZETTA; BRANDAO; SARCINELLI-FILHO, 2016; PIZETTA; BRANDAO; FILHO, 2014; PIZETTA; BRANDAO; SARCINELLI-FILHO, 2014). It is a computational system used both for practical experimentation and numerical simulations. AuRoRA makes possible to model robots, sensors, and the environment, or even implement individual or cooperative control strategies for homogeneous and heterogeneous formations. Thus, in such a context, this present work proposes the development of an aerial robot simulator platform, based on AuRoRA, with programming in the cloud, without the need to install proprietary software on the user's computer.

Different from AuRoRA, which is hosted in MATLAB©, the proposed framework runs in Google Colaboratory (Colab). It is a Jupyter Notebook on the web, increasing the performance of application execution since it is written in the C++ programming language. Thus, the framework has been named **cloud4AuRoRA**, to elucidate cloud programming and debugging. In summary, the developed tool brings together code with textual description of mathematical models and control laws, which enables its use in teaching since the greatest difficulty for students is to translate what is learned in robotics theory to computational practice, and in research, as a live paper presenting methods, code, graphical results and discussion.

For validation and better presentation/explanation, this work conducts a case study on the implementation of a practical problem on graphics processing units (GPUs), in NVIDIA CUDA (Compute Unified Device Architecture) language, parallelizing the application code to run thousands of simulations in a few seconds, on architectures available for free into Colab: Tesla K80, T4, P4 and P100.

1.1 Aims and Scope

This thesis aims to develop a mobile robot simulation platform hosted on Google Colab in order to combine the theorical description and mathematical modeling required for robot motion control and the simulation code, graphically demonstrating the robot's performance in accomplishing the assigned task, supporting the discussion and analysis of the results. In addition, this tool should enable the implementation of high performance computing (HPC) strategies for code parallelization and GPU programming to solve problems that require large amounts of simulations. To achieve this purpose, the following specific objectives are proposed:

1. Translate AuRoRA simulation from MATLAB to C++ language, in an optimized way;

2. Implement the cloud4AuRoRA in Colab, using C++ for the main code, and develop functions that generate the plots of interest, in Python;

3. Develop the framework as a "live-book", with mathematical description and discussion about the robot model and the implemented control strategy;

4. Implement cloud4AuRoRA on GPU for control tuning in parameters' space;

To complement the concepts presented so far, the sequence of this chapter brings a brief description about the AuRoRA platform, the Jupyter notebook and Colab, and the UAVs and their navigation and control strategies used to accomplish the proposed missions.

1.2 Open Source Robotic Simulators

Simulators play an important role in robotics research as tools for testing the efficiency, safety, and robustness of new algorithms, or studying the robustness of the performance of a robotic design by modifying only specific model/controler parameters or environmental conditions is one of the benefits of simulation softwares. Around the world there are several commercial and open-source simulators, the most advanced ones present dynamic interaction with the environment, proprioceptive and exteroceptive sensor models, human-robot interaction, as well as an extensive library of objects, robots and scenarios that can be adapted for each application. Most of them usually has 3D CAD models available (STARANOWICZ; MARIOTTINI, 2011). Code portability from a simulated to a real platform is another feature required for implement the strategies and take the robot for validation experiments. Examples of advanced simulators are: Gazebo[3]; FlightGear[4]; CopeliaSim[5]; MIT Drake[6]; Unit Robotics Simulator[7]; PyBullet[8]; RaiSim[9]; (CRAIGHEAD et al., 2007; KOENIG; HOWARD, 2004; CASTILLO-PIZARRO; ARREDONDO; TORRES-TORRITI, 2010; COLLINS et al., 2021). Modern simulators tend to provide the following features:

- Fast robot prototyping;
- Physics engines for realistic movements, as shown in PyBullet or ODE[10].
- Realistic 3D rendering to build the environments, using standard 3D CAD modeling tools such as Unit or Blender[11];

[3] https://gazebosim.org/
[4] https://www.flightgear.org/
[5] http://www.coppeliarobotics.com
[6] https://drake.mit.edu/
[7] https://unity.com/solutions/automotive-transportation-manufacturing/robotics
[8] https://pybullet.org/wordpress/
[9] https://raisim.com/
[10] https://bitbucket.org/odedevs/ode/src/master/
[11] https://www.blender.org/

- Dynamic robot bodies with scripting, as done in Gazebe using C++ and Python languages (KOENIG; HOWARD, 2004).

The use of simulators in robotics research is widespread, underpinning the majority of recent advances in the field. There are also platforms dedicated specifically to teaching, helping students to successfully assimilate abstract knowledge and skills (TSELEGKARIDIS; SAPOUNIDIS, 2021). Plasencia et al. (2019) presents two platforms for teaching and benchmarking deep reinforcement learning algorithms: the OpenAI Gym for robotics[12], which provides a collection of scenarios/problems designed to support the machine learning community to testing and developing reinforcement learning algorithms; and CoppeliaSim, a commercial software with educational version for free, providing sensors, solids, mechanisms and robot models, as well as algorithms for obstacle avoidance and navigation planning. In robotics competitions, such as robot soccer, simulators are also used extensively. In the RoboCup Soccer Simulation League, the teams use SimSpark[13], a multi-robot simulator for soccer simulations using NAO robot. The simulator presents realistic features such as real motor models and heterogeneous robots (XU; VATANKHAH, 2013).

All these tools are powerful, which makes them extremely complex because they manage many packages and libraries that must interact with each other. This makes the learning curve of such simulators high, making them difficult to use; because they require a lot of time from students and researchers to learn how to use the tool, and not necessarily learn robotics or develop an impactful application. In this sense, AuRoRA platform simplifies and abstracts all the low-level language by transforming the programming into a simple block diagram containing the model of the robots, the sensors and the environment, in classes and object-oriented programming in MATLAB language. It is worth stressing once again that the same platform is used for both robot simulation and experimental validation.

1.3 AuRoRA Platform

AuRoRA platform is a real-time mobile robot simulation and experimentation framework, hosted on MATLAB®. The tool is structured according to the *software-in-the-loop* (SIL) paradigm, in which all components (sensors, actuators, kinematic and dynamic UAV modeling, control, etc.) are simulated using descriptive mathematical models. The framework provides the tools to simulate and experiment UAVs with several types and characteristics, and different control strategies involving one or more robots simultaneously (PIZETTA; BRANDAO; SARCINELLI-FILHO, 2016; PIZETTA;

[12] https://openai.com/
[13] http://simspark.sourceforge.net/

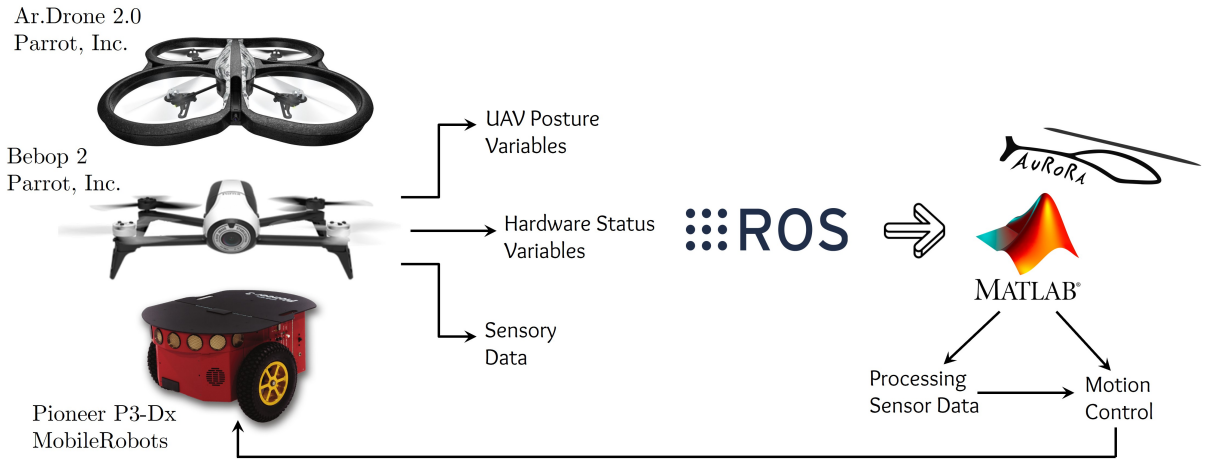


Figure 5 – AuRoRA platform operation overview and robots used in NERo.

BRANDAO; FILHO, 2014). As shown in Figure 5, the platform supports commercial robots (Bebop 2, ArDrone 2.0, both from Parrot Inc., and Pioneer P3-Dx, from Mobile Robotics) and their communication with *Robot Operating System* (ROS[14]), using a predefined set of libraries and tools that assist in managing application-specific messages.

Algorithm 1 presents the structure of the AuRoRA Platform. Note that all system actions require a permission to be executed. This permission to execute is given only when the time interval referring to a sampling period of each simulated robot is reached. In other words, for different robots simulated simultaneously, where the sampling periods are different, the data reading and sending to each vehicle is performed independently,

[14] https://www.ros.org/

Algorithm 1 Structure of the AuRoRA platform.

```
Initialization of Environment
Initialization of the Robot
Initialization of External Sensors
while t < t_max do
    if Execution Permission then
        Read Sensors
        Task Definition
        Calculate the Desired Position
        if Joystick Connected then
            Read Commands
        else
            Calculate the Control Signals
        end if
        Send the Commands to the Robot
        Store the State Variables
        if Permission to Plot in Real-Time then
            Display simulation graphics
        end if
    end if
end while
```

respecting such time intervals. Such an approach minimizes computational effort by avoiding the execution of a series of instructions at each iteration, and avoids sending multiple control signals to the vehicles in the same sampling period.

The platform is divided into several modules, which can be enabled or disabled. For example, the graphics generation can be enabled or disabled, a joystick can be enabled for safety (since its operation overrides the automatic controller), and some sensory system (GPS, vision system) can be disregarded, all without compromising the task execution.

For the purpose of describing the platform's operation, consider the Parrot ArDrone 2.0 quadcopter, which has a set of on-board sensors and is capable of generating a wireless communication link with the main computer of the proposed system, where the AuRoRA platform is running. By establishing a bidirectional communication, the platform starts the process of collecting sensory data and sending control signals to the drone. However, this task is only accomplished if the execute permission flag is set. Otherwise, no action is performed (i.e. received data is ignored and calculated commands are discarded). Therefore, in case of activation of the execution permission, the data is captured, the navigation reference at the time instant is given, and the navigation errors are then calculated. Next, based on these errors the control signals required to accomplish the task are determined, which are finally transmitted to the vehicle.

Considering a real UAV, the control signals are transmitted to the vehicle's onboard system, which applies them directly to the actuators. After reacting to the controller's actions and interacting with the environment, the onboard sensors are excited and register the values of the new flight condition. These values are transmitted to the platform and the control cycle restarts.

When the UAV is simulated, the control signals are sent to the dynamic model of the drone, which includes the actuator model and the rigid body model, and also considers the existence of uncertainties and disturbances added to the flight maneuvers. After acting on the model, the future posture of the drone is determined by numerical integration, therefore, updating the state variables (thus filling in the data concerning the sensory part). At this point, this data becomes available for a new permission to execute the control system. Figure 6 illustrates this simulation scenario. Different tools are used for the analysis of kinematics and dynamic modeling of the robots, for off-line programming (during simulation), to design and test different control algorithms. Notice that the model of Ar.Drone 2.0 and the AuRoRA Platform have already been validated in (BRANDÃO; FILHO; CARELLI, 2013) and (PIZETTA; BRANDÃO; SARCINELLI-FILHO, 2015).

If the graphics part is enabled, in both cases, a display permission flag is enabled at each preset period, and thus an illustration of the current state of the drone, throughout the flight task, is done. It is worth mentioning that the display permission time can be chosen arbitrarily. Nevertheless, it must have a value greater than the biggest sampling

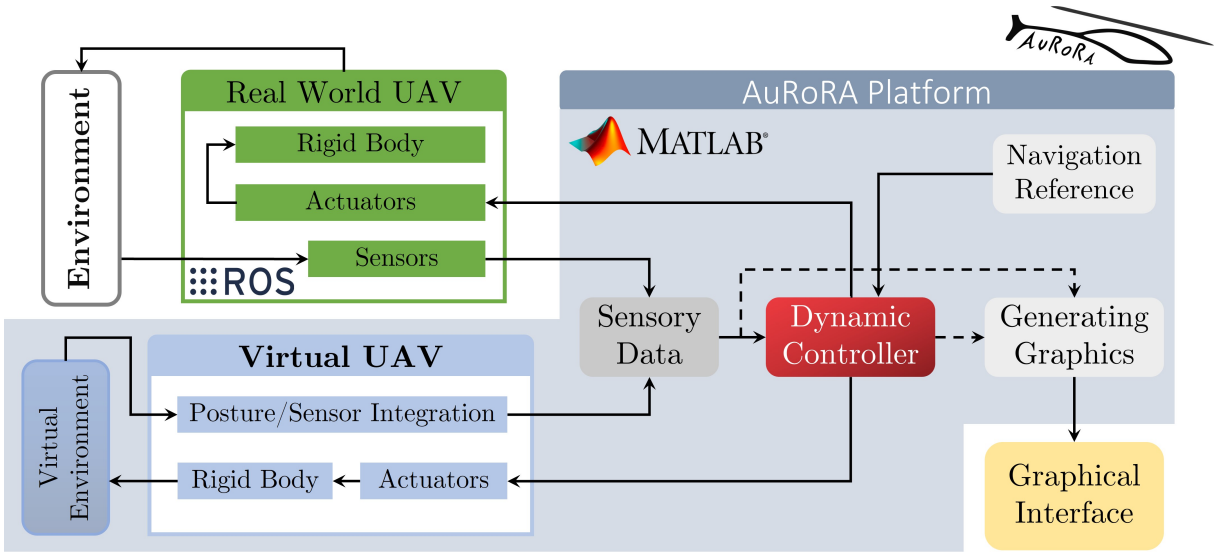


Figure 6 – Block diagram of the AuRoRA Platform's internal workflow.

period of the robots involved in the simulation or experiment, since the priority is to send stabilization commands to the robots before graphically displaying their navigation state. It is known that the display task has a high computational cost when compared to the operation cycles of the vehicles.

Finally, it is worth mentioning that at the end of the navigation routine all the data is stored, thus creating a record, which can be retrieved at any time for purposes of analyzing the mission's progress.

1.3.1 Unmanned Aerial Vehicle (UAV)

As for the UAV adopted, an *ArDrone 2.0* quadrotor, from Parrot Drones SAS, which consists of a set of four engines positioned in the shape of a cross, which are independently driven. The collective variation in propulsion forces, resulting 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. The quadrotor has six degrees of freedom (DOF) as shown in Figure 7. Its center-of-gravity is the robot reference ($x,\ y,\ z$), and the roll (ϕ), pitch (θ) and yaw (ψ) angles are related to its orientation around its axes x, y and z, respectively. Its control is performed in an under-actuated way, since it has a less number of actuators than the number of DOFs (BRANDÃO; FILHO; CARELLI, 2013). In summary, four control signals are responsible for guiding the robot according to its current state and the mission definition.

The command signals sent to the robot are $\mathbf{u} = \begin{bmatrix} u_\phi & u_\theta & \dot{u}_z & \dot{u}_\psi \end{bmatrix}^\intercal \in [-1,\ 1]$, in

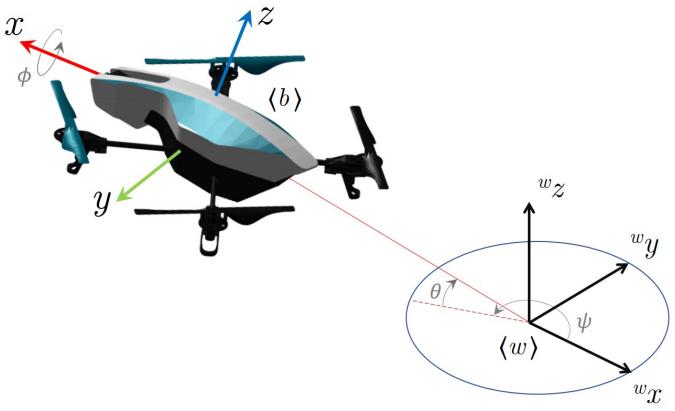


Figure 7 – ArDrone 2.0 description and pose variables, according to Tait-Bryan angles notation.

which 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.

At this point, it is important to mention that this vehicle is one of the most complex platforms for stabilization and control testing (TANG; XIAO; LI, 2017). For this reason, if only its kinematic model is considered, when proposing the navigation model of a vehicle, the designer must ensure that the UAV performs only linear displacements at low speeds, in the (near) absence of external disturbances. If this is respected, the dynamic effects of and on the UAV can be neglected. Otherwise, its dynamics must be considered when designing the controllers, since quadcopters have an inherently unstable and highly coupled nonlinear dynamic model (BRANDÃO; FILHO; CARELLI, 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 (BRANDÃO; FILHO; CARELLI, 2013; BRANDÃO et al., 2022).

1.3.2 Navigation Controllers

Since the focus of this research is not to propose a specialized control technique for UAV motion and stabilization, one of the controllers already described in (BRANDÃO; FILHO; CARELLI, 2013) is here implemented and discussed. A nonlinear controller, based on the ArDrone high-level dynamic model, is depicted. The four control signals, described in Section 1.3.1, are calculated by the control law and sent to the robot according to the block diagram presented in the Figure 8. It is beyond the scope of this research to present in detail the kinematic and dynamic model of the robot, and the implemented control law. The idea here is to present a generalist proposal that can be extended to other types of

robots and controllers by analyzing the effect of the communication time delay, as will be shown in the sequel. For this reason, such information is omitted, but the interested reader can access the available codes (both in the Google Colab platform and in the GitHub repository of AuRoRA).

The problem of motion control of mobile robots varies depending on the task to be performed, from regulating, to trajectory tracking, or path following. If the desired point is time-varying, i.e., has a time dependency, as illustrated in Figure 9(a), one has a trajectory and the control problem is defined as a *trajectory tracking control*. In turn, if a route is composed of a set of points and the robot has no time constraint to reach each point, one has a preset route, as illustrated by Figure 9(b). Each point on the path can be associated with a desired velocity at which the robot should reach the target. On it, $\mathbf{p}_k$ is the closest point to the robot and $\vec{\mathbf{p}}_k$ is the reference speed for it to reach the next point $\mathbf{p}_{k+1}$. In this case, the control problem is defind as *path-following control*.

In summary, from its current posture $\mathbf{x} = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^\top$ to a desired one $\mathbf{x}_d = \begin{bmatrix} x_d & y_d & z_d & \phi_d & \theta_d & \psi_d \end{bmatrix}^\top$, a control strategy guides the UAV to reduce its position error, given by $\tilde{\mathbf{x}} = \mathbf{x}_d - \mathbf{x}$, according to the control loop depicted in Figure 8. In order to make $\mathbf{x} \to \mathbf{x}_d$, a stable control loop guarantees $\tilde{\mathbf{x}} \to 0$, and whenever possible in a smooth and asymptotically way.

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 (BRANDÃO; FILHO; CARELLI, 2013; LV; HE; ZHAO, 2020; BRANDÃO et al., 2022). 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

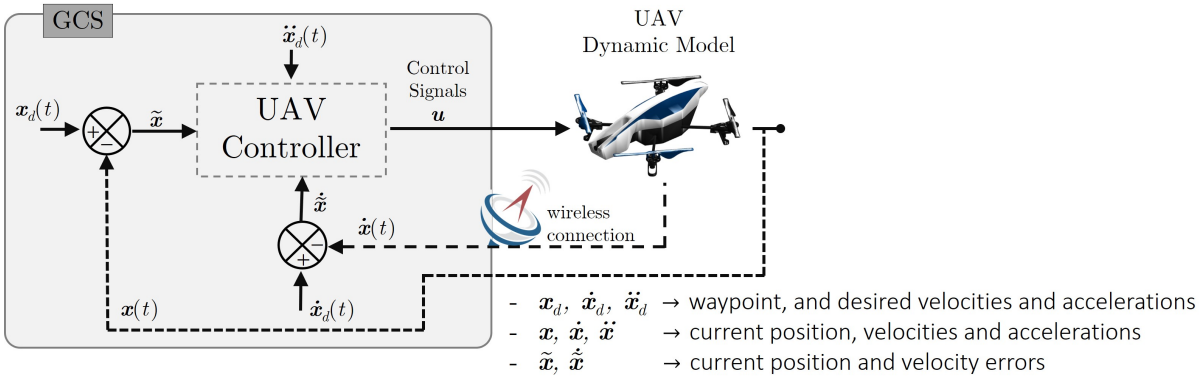
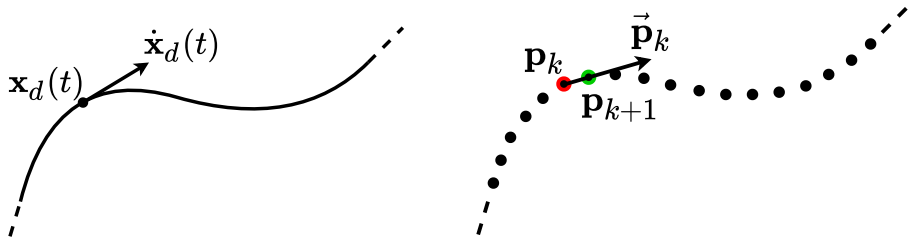


Figure 8 – Block diagram of the robot control system for experiments. For simplicity, the ROS was removed from the diagram. However, communication and message management is done by the ROS Master. If in simulation, the control signals is sent to the robot dynamic model and the posture is obtained by means of numerical integration. GCS is an acronym for Ground Control Station.



(a) Generic trajectory tracking. (b) Generic path-following.

Figure 9 – Motion Control Strategies (VASCONCELOS, 2021).

propulsion forces generated in the four engines coupled in the UAV, one needs an under-actuated model to describe the drones' dynamics. From the kinematic model of the aerial robot, already described in the literature by (SANTANA; BRANDÃO; SARCINELLI-FILHO, 2016; RABELO; BRANDAO; SARCINELLI-FILHO, 2018), it is possible to identify the UAV behavior and propose appropriate control strategies. A simple model is described, assuming that the pitch and roll angles are sufficiently small, i.e., the UAV operating in near-hovering.

1.4 Work Contributions

This Master's thesis presents contributions expressed in scientific production that has been published or is in the process of being published. Chapter 2 presents the development of cloud4AuRoRA, a mobile robot simulator that combines descriptive text, equations, and images, with code and examples with graphical analysis of the temporal variation of the simulated robot's state. Specifically, this work implements the dynamic model of ArDrone 2.0, as well as two control strategies in order to exemplify the use of the platform. Chapter 3 extends the version of the paper presented by Fagundes-Junior et al. (2021) whose goal was to investigate the problem of communication time delay in controlling UAVs in regulation and trajectory-tracking tasks, using parallel programming strategies, in order to optimize code execution time. This study evaluates what the effects are on the robot behavior when varying the controller parameters for an observed time delay in sending and receiving information. Analysis in controller parameter space provides a volume where motion stability and mission convergence can be guaranteed in a timely manner, according to the performance requirements of the task. Subsequently, a comparative analysis of the performance gain of the manually translated code from MATLAB to C++/CUDA is presented, using the contributions of Chapters 2 and 3, resulting in a 5000 times processing speed gain. This demonstrates the potential of the tool proposed in this work for applications that require massive amounts of simulations, such as machine learning algorithms, Monte Carlo simulations, or calibration of flight controller parameters.

In this first chapter, the motivation to use simulators in robotics becomes explicit. Considering this discussion for the specific context of NERo and the use of AuRoRA platform, which is linked to most of the work produced within the laboratory, the development of the cloud4AuRoRA framework extends the use of AuRoRA. Besides promoting the inclusion of other programming languages (Python, C++ and CUDA) in the simulator, it offers a tool that runs on the web browser without software acquisition fees, without the need to consume space and computing power on the user's machine; moreover, with optimized execution performance due to the programming languages used and access to powerful GPUs, often inaccessible to students and researchers in academia.

In addition to the contributions obtained directly from the subject of this dissertation, the author also contribute within the graduate program in Computer Science at UFV with the following works:

- A Nonlinear UAV Control Tuning Under Communication Delay using HPC Strategies in Parameters Space, *XXII Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)* (FAGUNDES-JUNIOR et al., 2021): 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. 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.

- Laser Sensing Based Navigation for a Leader-Follower Formation, *Simpósio Brasileiro de Automação Inteligente (SBAI)* (FAGUNDES-JÚNIOR et al., 2021): proposes a strategy for leader-follower formation of ground mobile robots considering an environment in which there are geometric patterns that can be confused with the pattern carried by the leader robot, corrupting the formation. In this work the laser sensing and detection of objects are simulated in the scene by intersecting line segments.

- Corridor Navigation for Mobile Robots using Laser Sensing, *14th IEEE International Conference on Industry Applications (INDUSCON)* (FREITAS et al., 2021): presents a control strategy for a unicycle type mobile robot that enables wall-following and corridor navigation, using the information obtained by the simulated laser sensor scan. In this work, control strategies considering obstacle avoidance were evaluated.

- On the Evaluation of Access-point Handovers for UAVs in Long-distance Missions, *International Conference on Unmanned Aircraft Systems (ICUAS)* (FAGUNDES; SOUZA; BRANDÃO, 2021): models and implements strategies for enhancing connectivity of a UAV performing long-distance navigation tasks, taking into account

applications' requirements. The proposed strategies aim at assessing the impact of frequent handovers between wireless access points in terms of packet loss and available bandwidth. The capabilities of the simulation tool designed for the work are illustrated through some case-study scenarios.

In addition, there are works under review and in the process of submission that collaborate with (and motivate) the development of this dissertation, namely:

- Decentralized Planning for UAV–UGV Cooperative Teams in Last-Mile-Delivery Missions, work submitted to the journal *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*: this work deals with the cooperation and control of multiple robots systems involving heterogeneous robot formation with sensing and actuation capabilities to perform load transportation tasks. All the work was developed using the AuRoRA platform. The video of the full experiment can be seen at the link: https://youtu.be/GDdqWQX0Hd8.

- Communication Delay in UAV Missions: A Controller Gain Analysis to Improve Flight Stability, work awaiting approval of the journal *IEEE LATIN AMERICA TRANSACTIONS*: This work analyzes the asymptotic convergence of a quadrotor, under time-delay in the communication with a ground control station. The the response-signal behavior of the quadrotors in the missions accomplishment are analyzed by numerical simulations.

- Decentralized Formation Control of Mobile Robots for Multiple Leaders and Multiple Followers, work accepted for publication at *XXIV Congresso Brasileiro de Automática*: describes a control scheme proposed for guiding a leader-follower formation of mobile robots. In our approach, there is no information sharing, emphasizing a decentralized approach. The proposed strategy can be applied in scenarios with several agents, in the possibility of having several leaders and also several followers.

- GPU-based Strategy for UAV Control Tuning Under Time-Delay, work submitted to the journal *Concurrency & Computation: Practice & Experience*: A GPU-accelerated flight route analysis for multi-UAV systems is proposed for the problem of control tuning in the parameters space considering the problem of delay in sending information to a ground control station (GCS). An extension of the previous work (FAGUNDES-JUNIOR et al., 2021).

- An UAV Path-Following Strategy for Crossing Moving Narrow Passages, work in process of submission to the journal *ISA Transactions*: aiming at the efficiency, stability, and security of an inspection task performed by unmanned aerial vehicles (UAVs) in unstructured and unpredictable environments, this work proposes an path-planning formulation that ensures safe passage through challenging moving locations while

respecting the robot's physical limitations and the narrow passage shape. The video of the full experiment can be seen at the link: https://youtu.be/KrNmOUjahy8.

- Machine Learning for Unmanned Aerial Vehicles Navigation: An Overview, submitted to the journal *Neural Computing & Applications*: presents a survey about the state-of-the-art in machine learning techniques for UAV navigation. This work was developed through a research conducted from the interest in working with the topic. In this sense, the proposal described in this dissertation is able to help in the training of autonomous navigation models, since training the model online can be dangerous, thus avoiding injuries and wear of the robots.

- Identification and Prevention of Intruders in Leader-Follower Formations, submitted to the journal *Robotics and Autonomous Systems*: in our previous work, a follower robot uses its laser sensing to localize and identify a geometric pattern on-board the leader. Considering that identification problems can occur in the presence of objects with dimensions similar to the pattern onboard the leader, this work proposes and implements a Leader-Follower formation strategy for mobile robots in dynamic environments with other intruders robots ("false leaders"). The proposed sensing strategy modifies the searching area, when seeking for the leader, according to the estimation the follower gets of its velocity. The video of the full simulation can be seen at the link: https://www.youtube.com/watch?v=87mtXPoM608.

- Heterogeneous Formation Control in a Collaborative Cargo Transport and Delivery Task Using Lifting Electromagnets, submitted to the journal *Applied Science*: a cargo transportation problem involving heterogeneous robot formation is discussed. Two unmanned ground vehicles (UGVs) working cooperatively with an unmanned aerial vehicles (UAV) are used to validate the proposal. The existence of a physical obstacle between the two UGVs makes it impossible for them to meet each other. Thus, the lifting, transport and delivery of the cargo from one UGV to the other are performed by a UAV with an electromagnet actuator.

1.5 Structure of the Thesis

The thesis has been structured to consist of self-contained chapters, where each one will have its own introduction and conclusion according to its purpose.

- Initially, Chapter 1 presents a general introduction and briefly discusses the impact of simulations in robotics. It also introduces the AuRoRA platform, on which the contribution of this thesis is based.

- Next, Chapter 2 briefly introduces Jupyter notebooks and IPython, the development base for the cloud4AuRoRA framework. Then, it describes the developed platform

detailing its organizational structure, which contains descriptive text and code implementing the model and controller of the studied aerial robot. Finally, some examples of the use of the tool are presented.

- In the sequel, Chapter 3 discusses a possible application of the cloud4AuRoRA platform via parallel programming in CUDA language. This chapter presents the controller calibration problem and the communication delay problem in aerial robot control. The proposal is comparatively validated against results obtained using AuRoRA, in MATLAB, showing a code execution performance gain of over 5000 times.

- Finally, Chapter 4 elucidates the concluding remarks and suggests different branches for the continuation of the developed work.

2 From AuRoRA to cloud4AuRoRA

Researchers around the world have been motivated to propose models and flight controllers capable of guiding Unmanned Aerial Vehicles (UAVs) for a wide variety of applications. However, because such aircraft are inherently unstable, nonlinear, strongly coupled, multi-variable with complex and highly coupled dynamics (often underactuated), conducting experiments with UAVs, regardless of their nature, is quite risky, not only for the equipment under development but also for the people nearby (TAKAOĞLU et al., 2022; HASSANI; MANSOURI; AHAITOUF, 2019). This is due to the fact that such vehicles can fly at high speeds and that their blades rotate at very high speeds. This issue motivates the creation of simulators of high complexity and approximation to reality, which are of extreme importance for the development of systems for autonomous navigation.

The rapid advances in aerial robotics are mainly due to the possibility of simulating many different types of UAVs and tasks, taking into account certain characteristics of the environments, as well as the dynamics and kinematics of the agents involved (ALJEHANI; INOUE, 2019). The use of flight simulation tools, for both research and education, reduce the schedule, risk, and required amount of flight testing for complex aerospace systems is a well-recognized benefit of these approaches. Due to the complexity and difficulty of performing experiments with these robots, simulation becomes a feasible and necessary alternative. Since UAVs are highly unstable devices, the design and validation of controllers is initially done through mathematical models and implementation in simulators. However, focusing either on the modeling of UAV operations and dynamics, available simulation tools are complex in some aspects and often requires locally installation of additional programs in personal computers (HENTATI et al., 2018; AL-MOUSA et al., 2019; EBEID et al., 2018). In certain cases the simulators requires high processing capacity and computing power as well as high quality 3D rendering representations which can decrease the control loop stability depending on the computer characteristics. Some of them are paid and computationally heavy, requiring the student/researcher to install software and packages on their own machines, such as MATLAB/Simulink.

In particular, the use of robotics simulators in general has proven to be a tool in education and engineering design. However, there is currently a lack of published work focusing on the use of simulators integrating all mathematical descriptions for modeling and controlling robots, and the mathematical implementation to code, elucidating how the process of translating theoretical concepts to real-world applications works. Especially for engineering students, this last step is the most difficult, and represents a rather slow learning curve.

In summary, the current robotic simulators are extremely complex tools. Commonly

they are generalists and need to manage several protocols and subsystems that describe different robots and environments, which must interact with each other. Besides this, they cannot present complex mathematical descriptions behind the implementations, which ends up leaving it up to the user to understand on his/her own and only then understand the platform's codes.

In such context, the proposed framework is designed to be used in an open and interactive platform which can run in the web browser, with no need to install any software locally (thus making it possible to use the platform on mobile devices such as cell phones and tablets). The implementation, which is based on C/C++ language, enabling its ease of use and implementation in advanced research and in the robotics apprenticeship. Once finished the simulation, it can generate figures for variables and robot behavior analyses, in Python.

To discuss the proposed framework, the rest of the chapter is organized as follows. Section 2.1 presents a brief introduction to IPython, Jupyter notebook and Google Colaboratory, where the framework proposed in this thesis was developed. Section Section 2.2 presents in detail the cloud4AuRoRA tool and its main attributions. Finally, Section Section 2.3 discusses the improvement from AuRoRA to cloud4AuRoRA.

2.1 A Brief Introduction to IPython and Jupyter Notebook

It is well known that Jupyter notebooks offer an efficient method for creating documentation which includes code/software tools and explanations into the same document (RULE et al., 2019). There are two main cell types. The "Markdown" cells presents text, images, equations, and other resources. The code cells allow the user to type and execute a code in a chosen programming language. Furthermore, the code/software inputs/outputs produce (text, graphical, images and animations) that appear immediately in the notebook. First, Subsection 2.1.1 presents the Ipython library, which is the basis of a Jupyter notebook. Following, Subsection 2.1.2 presents a brief discussion about Jupyter notebook and the relevant concepts. Finally, Subsection 2.1.3 gives an introduction to Colab, the tool used by this work to develop the cloud4AuRoRA framework.

2.1.1 IPython

In scientific computing, trial and error is the rule rather than the exception, and this requires an efficient interface that allows for interactive exploration of algorithms, data, and graphs. Pérez e Granger (2007) propose Ipython[1] project (the I in IPython stands for "interactive") to provide an enhanced interactive environment including support for data visualization and facilities for efficient computation by means of distributed and parallel

[1] https://ipython.org/

computation. IPython offers a set of control commands, also known as "magic commands", designed to improve Python's usability in an interactive context. Moreover, the user can extend it with new commands as desired. Therefore, the user could build custom interactive environments. Ipython also encapsulates the underlying operating system (OS), and the users could navigate the file system with Unix/linux commands, such as `ls` or `cd` by only adding a single '`!`' prefix character for direct execution by the underlying OS. Other important features to mention are: IPython offers access to Python's help system, the user is able to directly manipulate code and objects as they exist in the runtime environment, thus making it possible to complete any object's names and attributes with the Tab key, and a system to query an object for internal details, including source code, by typing the object's name and one or two '`?`' (two for extra details); also, IPython's '`%run`' magic command lets users run any `.py` file within the IPython session as if they had typed it interactively, so the user can further explore any quantity computed by the program, plot it, and so on. Typing `run?` provides full details about the run command.

From (NAGAR, 2017; ROSSANT, 2013), in summary, IPython offers the following features:

- An interactive shell for the Python interpreter;
- A browser-based notebook with support for code, text, figures, LaTeX mathematical expressions, Markdown interpretation, inline plots, and other media;
- Support for interactive data visualization and use of graphical user interface (GUI) toolboxes;
- Flexible, embeddable interpreters to load into a project;
- Tools for parallel computing.

These features are useful when developing code, exploring a problem, or using an unfamiliar library because direct experimentation with the system can help produce working code that the user can then copy into an editor as part of a larger program (PÉREZ; GRANGER, 2007). Therefore, a Jupyter notebook creates an overlay layer to interact to the file system and program facilities independent of the underlying operational system as a virtual machine, in addition to documentation and code presentation. It is possible to install third part softwares. Moreover, Colab provides a serve to free host cloud Jupyter notebooks with a high performance processor and hardware accelerators (GPU and TPU).

2.1.2 Jupyter Notebook

Reproducible publications are more easily reusable and therefore offer a significant opportunity to make research more impactful. However, reproducibility of computational

work is often hindered not only by the lack of data or meta-data, but also by the lack of details about the procedure and tools used, such as: the source code of the software used is not available; information about the computational environment, such as the hardware or operating system, is not disclosed; and, the exact procedure that led to the results reported in the publication is not shared (RULE; TABARD; HOLLAN, 2018a; BEG et al., 2021). All of this compromises the replication of research from one group by another.

Nowadays, making data and analyses understandable and public is crucial to advancing open science and enabling reproducibility while encouraging the exploratory process that analysts use to think with data. Computational notebooks are an important step towards these ends, but still in an early stage of development (RULE; TABARD; HOLLAN, 2018b). Computational notebooks combine code, visualizations, and text in a single document, addressing many fundamental challenges with performing, documenting, and sharing data analyses. They support incremental and iterative analyses, enabling users to edit, arrange, and execute small blocks of code in any order. They enable explanation of thought processes by allowing analysts to intersperse code with richly formatted textual explanations. They facilitate sharing by combining code, visualizations, and text in a single document that can be posted online or emailed. Some computational notebooks are truly remarkable in the way they elegantly explain complex analyses (KERY et al., 2018; ONO; FREIRE; SILVA, 2021; KANG et al., 2021). In order to contextualize, in 2014 the IPython developers decided to acknowledge the general-purpose nature of the IPython-Notebook by giving a new name to the project: Jupyter[2] (ROSSANT, 2013).

A Jupyter notebook is composed of cells, which can be of three types: code, markdown, and raw. A code cell contains executable code used to produce results. A markdown cell contains formatted text. Finally, a raw cell contains text that is neither code nor formatted text. Tools that convert notebooks into other formats use raw cells for configuration. Jupyter applications uses a kernel to execute code cells. When Jupyter sends a code cell for execution, it marks the cell as executing by assigning "*" to the cell execution counter. After the execution, the kernel allocates a number to the counter, which indicates the execution order. Users can execute the cells in any order, and a given cell can be executed multiple times (PIMENTEL et al., 2019).

Figure 10 shows an executed Jupyter notebook[3] which contains two markdown cells and two code cells. On the left of code cells, Jupyter displays an execution counter that indicates the order in which the cells were executed.

Below the code cells, Jupyter displays their outputs. Note that the first code cell displays an image, without returning a number on the left, and the third code cell returns a number, identified by [5]. This figure also presents two skips on the execution counters.

[2] https://jupyter.org/
[3] https://jupyter.org/try-jupyter/lab/

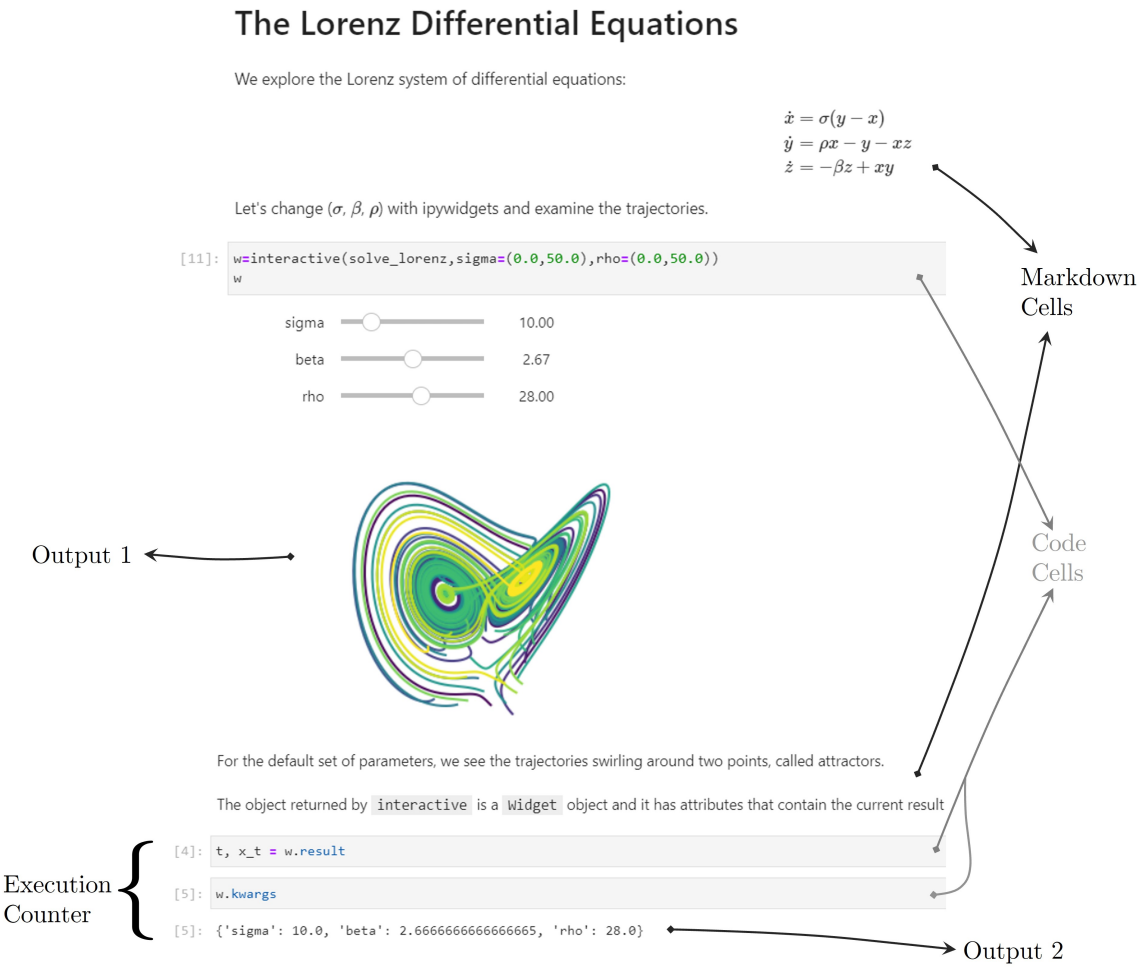


Figure 10 – A Jupyter notebook example with markdown, code, and interactive output with sliding button. This code is an example code from web browser Jupyter.

A skip represents cell executions that do not have explicit definitions in the notebooks. In this case, the four executions before the execution counter 5 represent one skip, and the four executions between 5 and 11 represent the other. The execution of a notebook does not require cleaning the outputs of previous executions. Thus, an executed notebook may contain retrospective data of multiple executions, which is the case of the first code cell, with the `[11]` index, which indicates that this cell was executed after the execution of 10 other cells.

The benefits of using the Jupyter environment for reproducible scientific workflows are numerous (BEG et al., 2021):

One study – one document: It is possible to carry out an entire study within a single notebook and provides a complete and executable record of the process, enriching the document with the interpretation of the results, immediately below the graphical, tabular or text-based output that needs to be described. Inserting all code, data,

and description and discussion within a single document can negatively affect the notebook's readability, making it appropriate to use parts of the code in libraries in repositories that will be imported in the notebook.

Easily shareable: The user can convert the Jupyter file (`.ipynb` file extension) to other file formats, such as HTML, LaTeX and PDF, facilitating sharing between collaborators without the need to install any additional software.

Interactive execution: The development of Jupyter notebook often involves interactively editing it, executing cells, inspecting computed outputs, modifying commands, and re-executing, while understanding the computational research question. Once a useful processing sequence has been found, the student/researcher may repeat that, often with different input data to test and validate the implementations.

Static and interactive software documentation: As the authors can type commands and explanations into the same document, and the outputs that the commands produce (text and images), it can save time during the documentation. Any change on the user interface or computational algorithms can be easily show where the documentation needs improvement by simply re-executing the documentation notebooks.

Documents in the cloud: Some tools offer customized computational environments in the cloud on-demand, such as Binder project, in which notebooks can be executed interactively, providing an environment for research, workshops or teaching purposes.

Reproducibility: Reviewers and editors of scientific and technical papers are increasingly (and justifiably) asking for details on how published results can be reproduced, or at least expect authors to provide that information if a reader requests it. It is often recommended that the text of an article contain descriptions of the experience of the authors in performing the work to prevent the reader from experiencing the same problems as the authors, thus reducing the development time of new work. However, it is difficult, and often impossible, to truly document an entire description of such parameters within a conventional manuscript submission. Combining data, code, and software environment the Jupyter-based research environment can help once it makes the process of publishing reproducible computational results easily achievable.

Remote access: The notebook can be shared and accessed by anyone with permission, making possible its use for classes assignments, for example. Jupyter notebooks offers the possibility to be accessed and executed on an Android device. A vital point of the user experience is that only a web browser is required to access the documents and to carry out computational work using these resources remotely.

Blending script and GUI-driven exploration methods: Notebooks provides selection menus, sliders, radio buttons, and other GUI-like graphical interaction widgets that can be used, for example, to select an answer in a questionnaire during classes, or to vary input parameters values and analysis the new obtained results.

Further, Notebook is not restricted to scientific computing, it can be used for academic courses, software documentation, or book writing thanks to conversion tools targeting Markdown, HTML, PDF, ODT, and many other formats (ROSSANT, 2013).

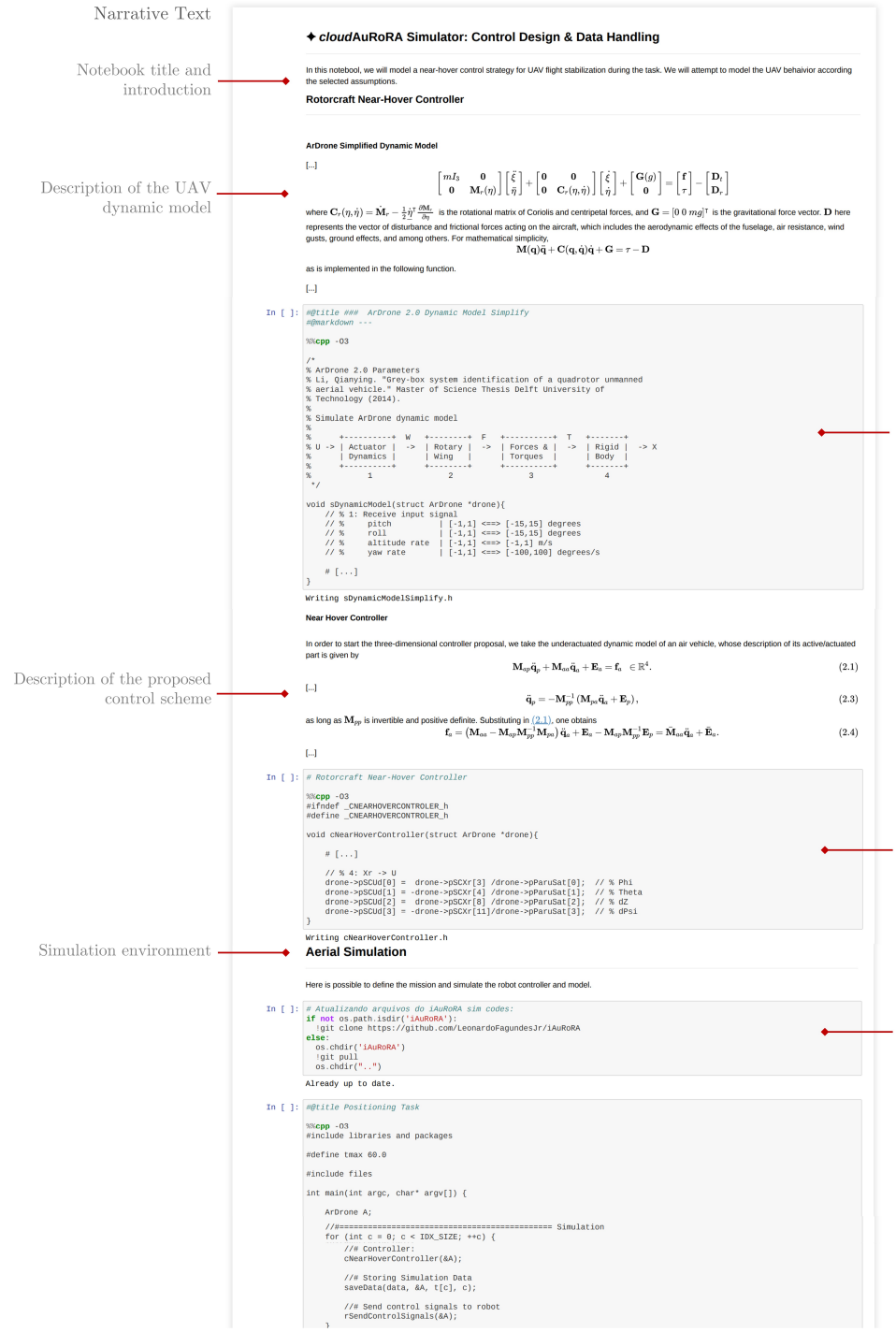
2.1.3 Google Colaboratory - Colab

Google Colaboratory[4] more commonly referred to as Google Colab, or simply Colab, is a very widespread machine learning education platform, is a free platform that provides a serverless Jupyter notebook environment for interactive development for the dissemination of research and education (PÉREZ; GRANGER, 2007; BISONG, 2019). Jupyter notebook includes documentation notes and code in a simple file. Colab provides support to run it on the cloud without the need for complex software setups. Colab integrates Jupyter notebooks to Google Drive and Github. Thus, they could be easily accessed and shared data and code.

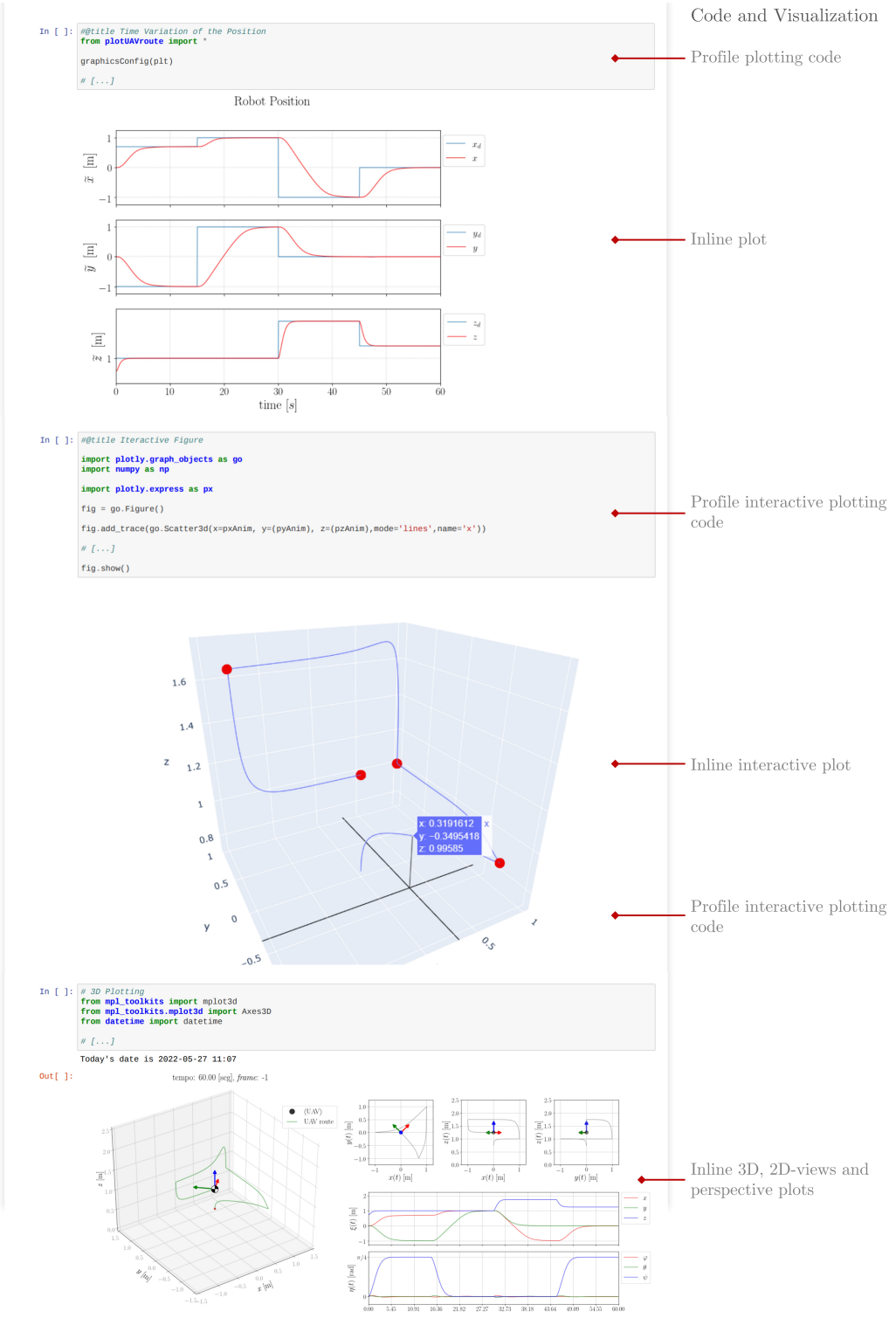
It has been found that the vast majority of Colab and Jupyter notebook examples focus on machine learning subjects (RULE et al., 2019; RULE; TABARD; HOLLAN, 2018b), which may indicate a lack of work to implement other computer science subjects as well as general science subjects in all fields. This subsection aims to promote ideas to pave the way to all subjects and that can be simply understood. There are plenty of simple ideas around Jupyter and Colab notebooks, and lots of them are used to. Here these platforms are show up as potential educational tool beyond machine learning research and teaching.

While Jupyter notebooks provide rich resources for presenting code notes and documentation in a single environment, there is a lack of explanation in most computational notebooks (RULE; TABARD; HOLLAN, 2018b). This can discourage users from promoting open science and reproducibility. Focusing on this gap, this thesis proposes new opportunities to design Colab notebooks for robotics teaching and research. One of the main challenges will be to develop simple Colab templates to facilitate the production of labs without much additional effort for the teacher (developer) and students (users). Colab runs in a browser and provides a unified web interface where code, text, mathematical equations, graphs, and interactive graphical controls can be combined into a single document. This is an ideal interface for scientific computing. Figure 11 illustrate a screenshot of a sample cloud4AuRoRA notebook, which is further explained in this chapter.

[4] https://colab.research.google.com/



(a) cloud4AuRoRA combining text, description, modeling, control and code for simulation and verification.



(b) Example of a graph of time variation of the robot's position and the sequence of waypoints given by the mission.

Figure 11 – A snapshot of the Colab notebook developed for the cloud4AuRoRA framework, which demonstrates the platform's implementations for modeling and controlling UAVs in simulation. The notebook combines code, visualizations, and text into a computational narrative. The "`[...]`" is equivalent to information removed from the text to increase space.

2.2 Cloud4AuRoRA Framework

The proposal developed in this work is configured as a "live book" where one can find both descriptive sections with illustrative images and the detailed development of the equations that describe the system as well as the controllers used, interspersing between the texts, a set of individual codes for each of the theoretical developments. For easy understanding of each part of the platform, it has been divided into chapters. Throughout the chapters the fundamental knowledge for the use of the tool is presented, as well as for the understanding of the kinematic and dynamic modeling of the ArDrone 2.0 and proposals for control based on inverse dynamics for a quadrotor with restricted motion and free flight, performing well-known tasks in the literature such as positioning and trajectory tracking.

The Google Collaboratory (Colab) notebook of this project is available at https://colab.research.com/cloudAuRoRA_TableOfContents. It provides the implementation and simulation of a quadrotor by its dynamic model and a control law suitable for performing positioning and trajectory tracking tasks. The robot adopted here is the Parrot ArDrone 2.0 from Parrot Inc., well known and used in academia and industry. However, any type of aerial robot could be easily implemented by changing the dynamic and control parameters, or simply by using a generic kinematic model.

The proposed framework can be used in an open, interactive platform that can be run in the web browser, in C/C++ and Python languages, allowing its ease of use and implementation in advanced research and robotics learning. After presenting the tool for the specific use of a positioning task, other control schemes and robotic cooperation with UAV are evaluated. In the proposed platform, it is possible to view the results of the robot motion after the simulation is finished. It is also possible to collect graphical data and save it as high-quality images that can be used in reports or specialized scientific papers or presentations.

At the end of each simulation it is possible to generate graphics and animations in order to analyze certain variables, such as position and orientation, control signals, robot velocities, among others. As a result, users only need a web browser and a Google account to work with cloud4AuRoRA framework.

In summary, the platform content was split into seven notebooks as explained below:

cloud4AuRoRA: An Interactive UAV Simulator is composed of text and image cells, and presents the platform and its division into chapters with a brief description about the content of each chapter. In Figure 12 one can see the basic structure of this Colab.

Ch 1: The Framework presents the simulation of the aerial robot used in this research. ArDrone 2.0 (Parrot) aerial robot model is implemented and tested in performing positioning and trajectory tracking tasks. This Colab also presents an introduction to the Aerial Robotics Simulator Pipeline.

This framework is based on the open source mobile robot emulator developed in MATLAB© (See AuRoRA github for more information). In this case, C/C++ language is used to rewrite the code and the simulator increasing the code execution performance, as shown in Figure 13.

Here, the robot is considered as a `struct` with assignments similar to the code in MATLAB. The first release of the tool presents only the simulator module of AuRoRA platform. Thus, a virtual entity representing the drone is initialized, and

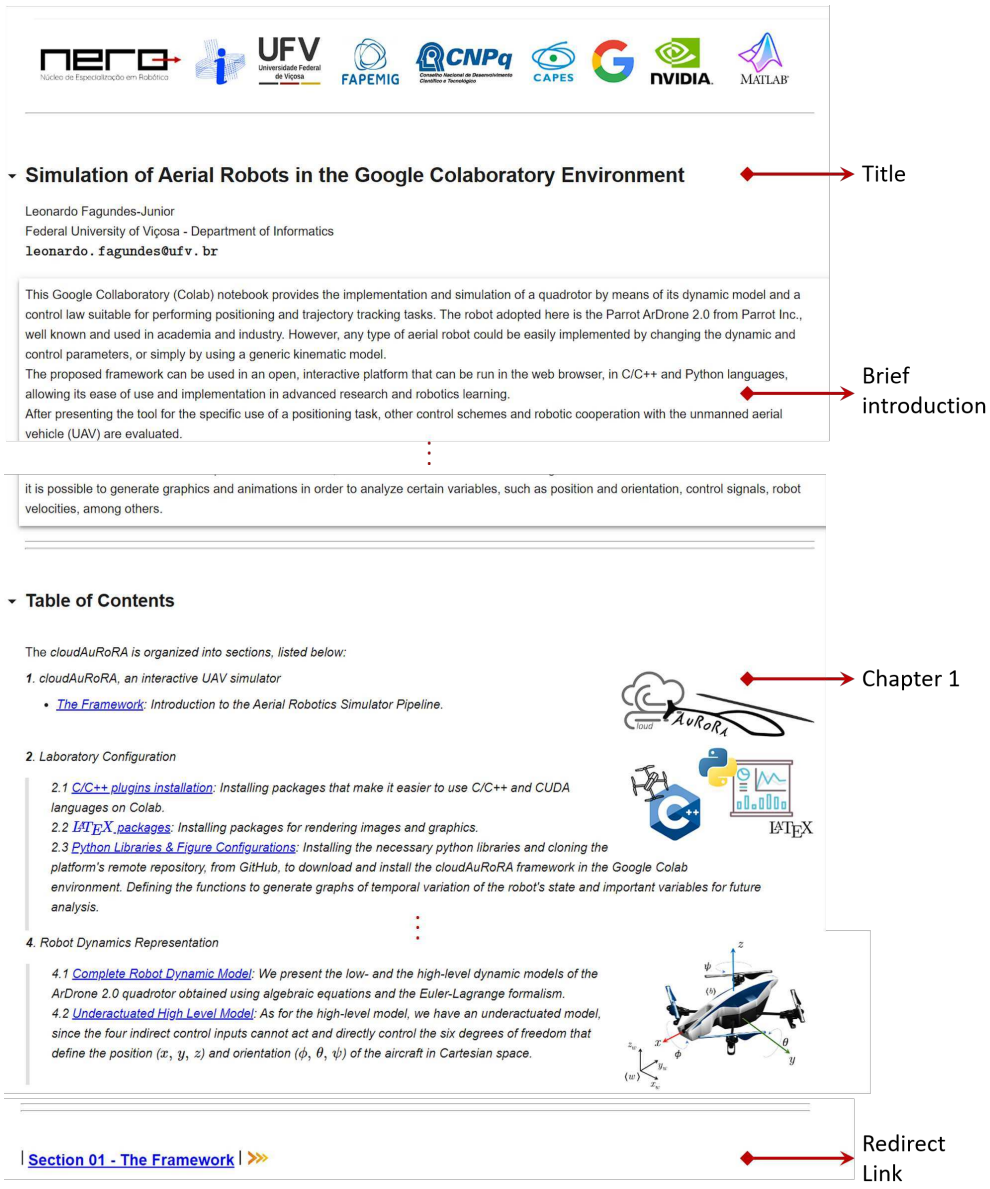


Figure 12 – Colab summary of cloud4AuRoRA platform.

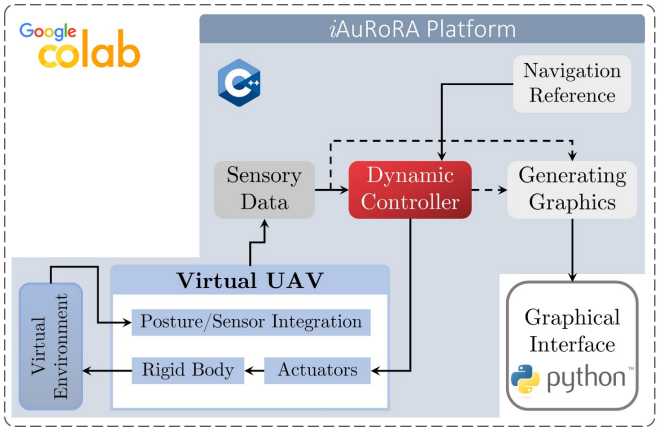


Figure 13 – Block diagram of the cloud4AuRoRA framework.

it will interact with the environment by collecting posture signals and simulated sensors, to be used in the generation of control signals and graph generation.

To optimize the code execution, the graphical simulation generation interface has been separated from the simulator. Since the graphical generation can negatively interfere in the control loop, if it takes longer than one sampling cycle of the UAV, so, it can be executed after finishing the simulation.

Algorithm 2 presents the structure of the platform, and stress the sequence and priority of the actions.

For graphical/plot generation and data analysis, IPython and Matlibplot functionalities are used. The users are able to find material on the internet about widely-used libraries such as NumPy (providing a practical array data structure), SciPy (scientific computing), matplotlib (graphical plotting), pandas (data analysis and statistics), scikit-learn (machine learning), SymPy (symbolic computing), and Jupyter/IPython (efficient interfaces for interactive computing). Python, along with this set of libraries, is sometimes referred to as the SciPy stack or PyData platform.

Algorithm 2 cloud4AuRoRA platform.

Require: C++, CUDA and LaTeX plugins
Robot Initialization
Environment Initialization
while $t < t_{max}$ **do**
    if Execution Permission **then**
        Read Sensors
        Task Definition
        Calculate the Desired Position
        Calculate the Control Signals
        Send the Commands to the Robot
        Store the State Variables
    end if
end while
Display simulation graphics

Ch 2: Colab Configuration is based on two main programming languages: C++ and Python. The simulator codes are implemented in C++, i.e. everything concerning the simulation execution (robot model, controller, main, etc.). The graphic analysis and animations are implemented in Python, using LaTeX text interpreter.

Ch 3: UAV Representation & Data Initialization: The robot is considered an entity defined as a `C/C++ struct`, where its data are stored in a single variable, called `drone`. The dynamic model consider the real robot and its physical parameters according to the model identification. In simulation the robot pose comes from the integration of sensory and posture data.

This Colab notebook summarizes all the variables associated with the robot such as its current or desired posture, its control signals, vector of disturbances acting on the aircraft during flight, among others. The list of all parameters and constants used in the robot model are also presented in this same notebook, as shown in Figure 14.

Ch 4: Robot Dynamics Representation: The low- and high-level dynamic models of a quadrotor obtained using algebraic equations and the Euler-Lagrange formalism are here represented. A preview of Colab is presented in Figure 15, in which it is possible to identify some mathematical descriptions about the model of the aerial robot, illustrative images that help in the understanding of the process described, as well as the code implementing all the mathematical description developed.

Ch 5: Control Design & Data Handling: Two flight controllers are presented to guide the aerial robot in its missions autonomously. The first controller considers a quasi-static motion constraint, while the second controller considers free motion in space. Both controllers are described mathematically and implemented in code in order to elucidate a possible codification of what was seen in theory, as well as in previous Colabs. Some excerpts from this notebook can be seen in Figure 16.

Ch 6: Simulation and Analysis: The latter Colab is a human-readable document detailing the computational steps taken. Thus, the user (researcher, teacher or student) is able to make it a highly productive environment for robotics in education and research, while facilitating reproducibility. Figure 17 presents a part of this notebook highlighting its main features.

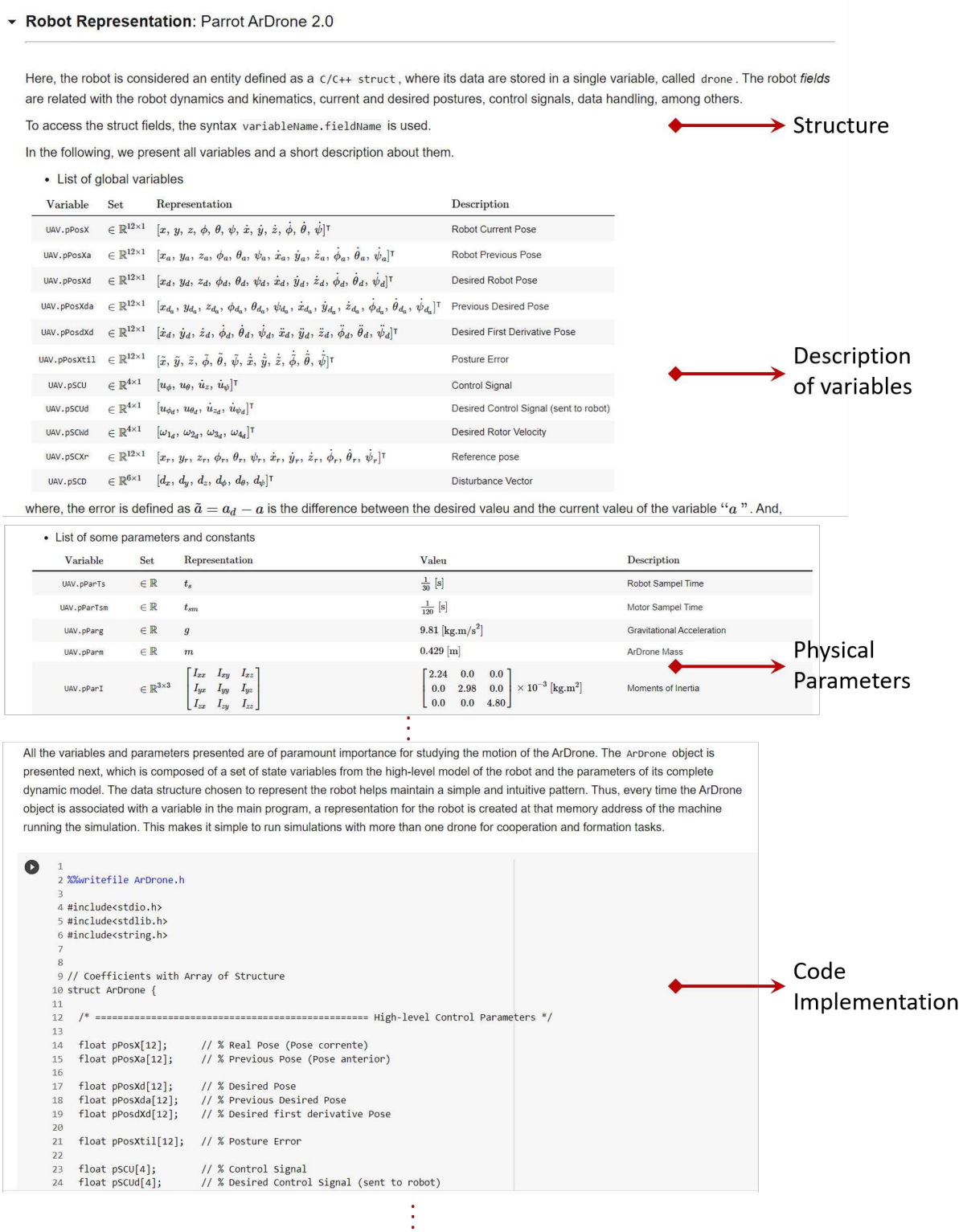


Figure 14 – Colab with the description of the robot representation and initialization of ArDrone model data and parameters.

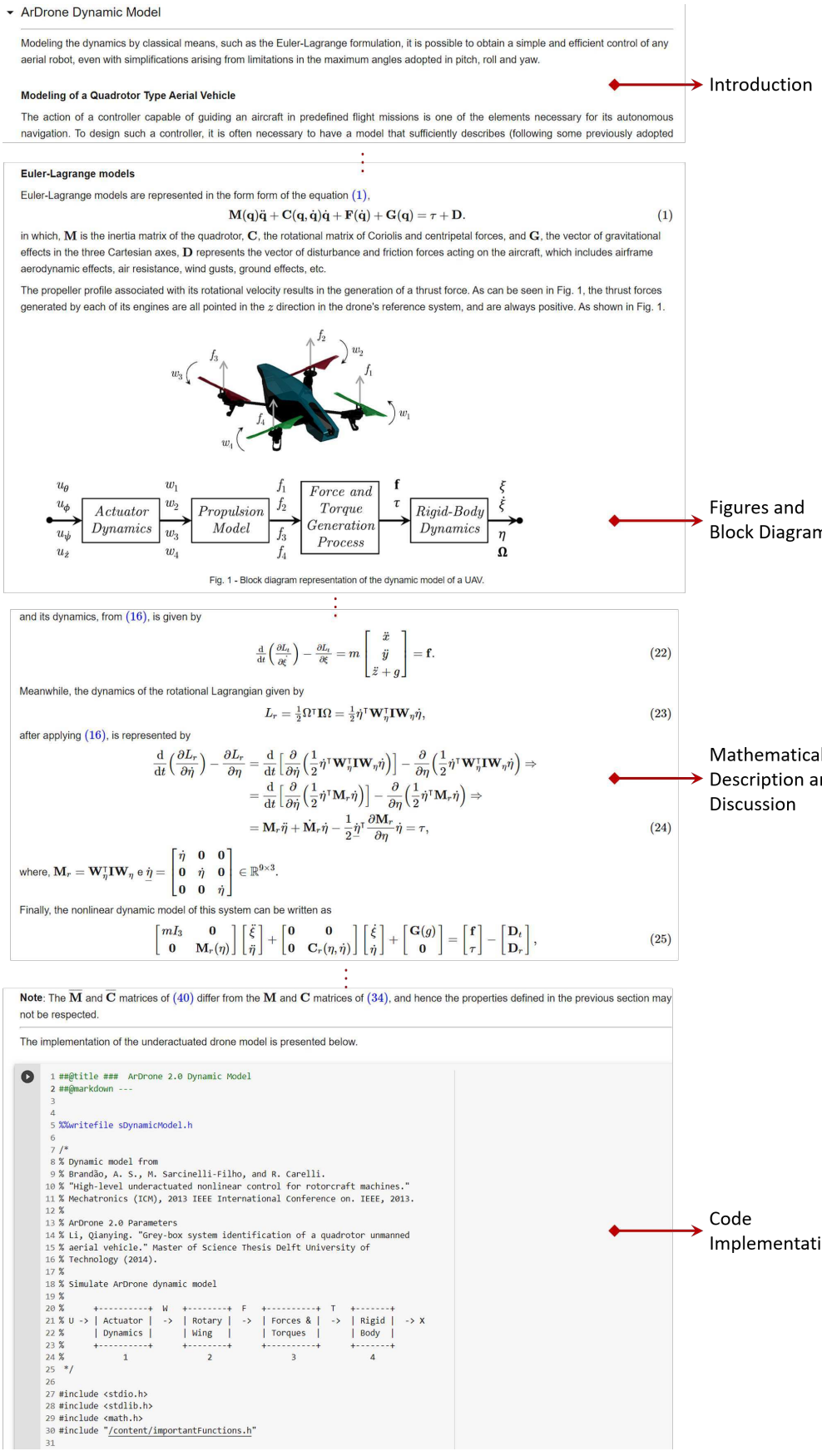


Figure 15 – Colab containing the mathematical description and implementation of the ArDrone 2.0 model.

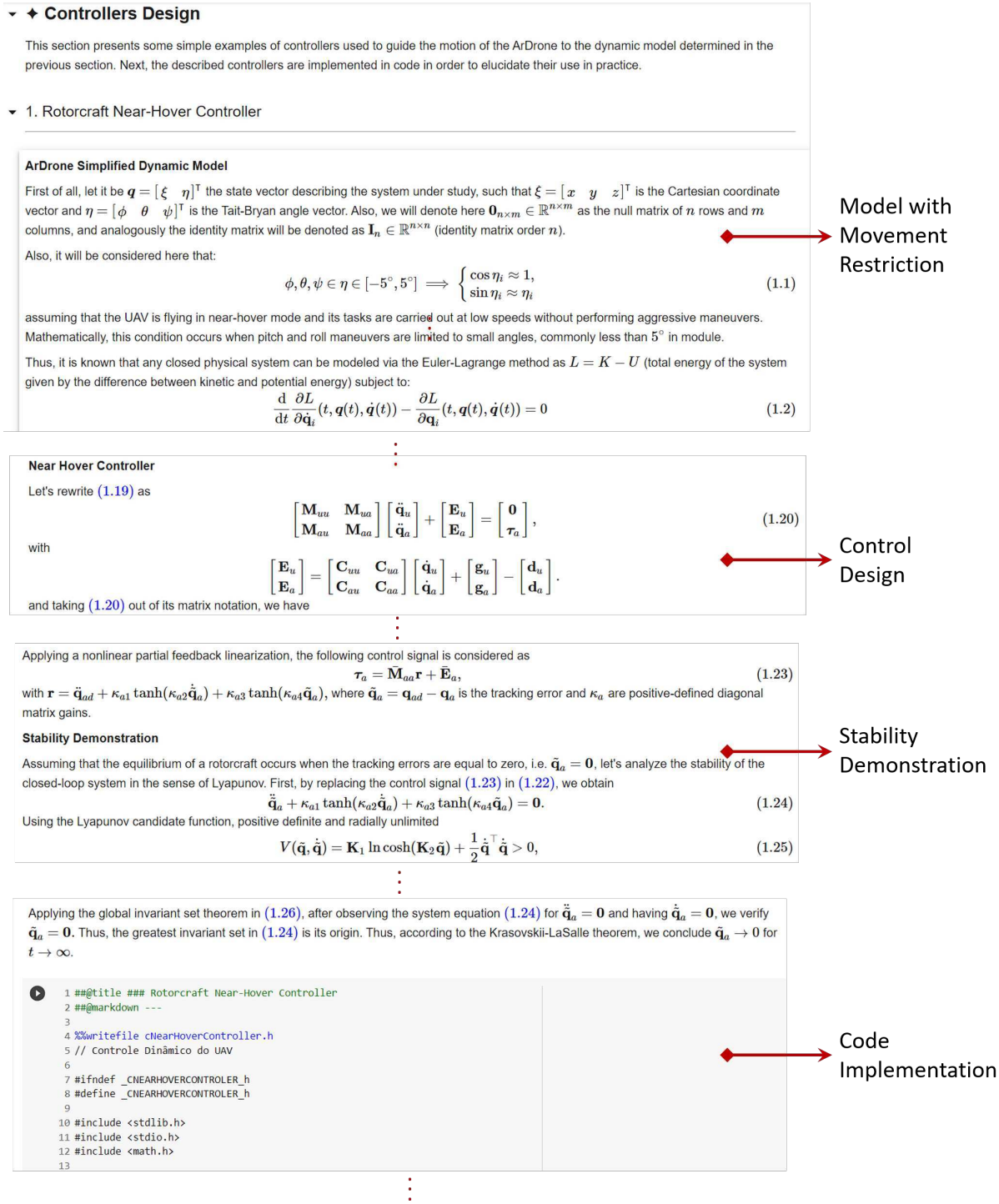


Figure 16 – Colab containing the design of a control system needed to guide the aerial robot in its flight missions.

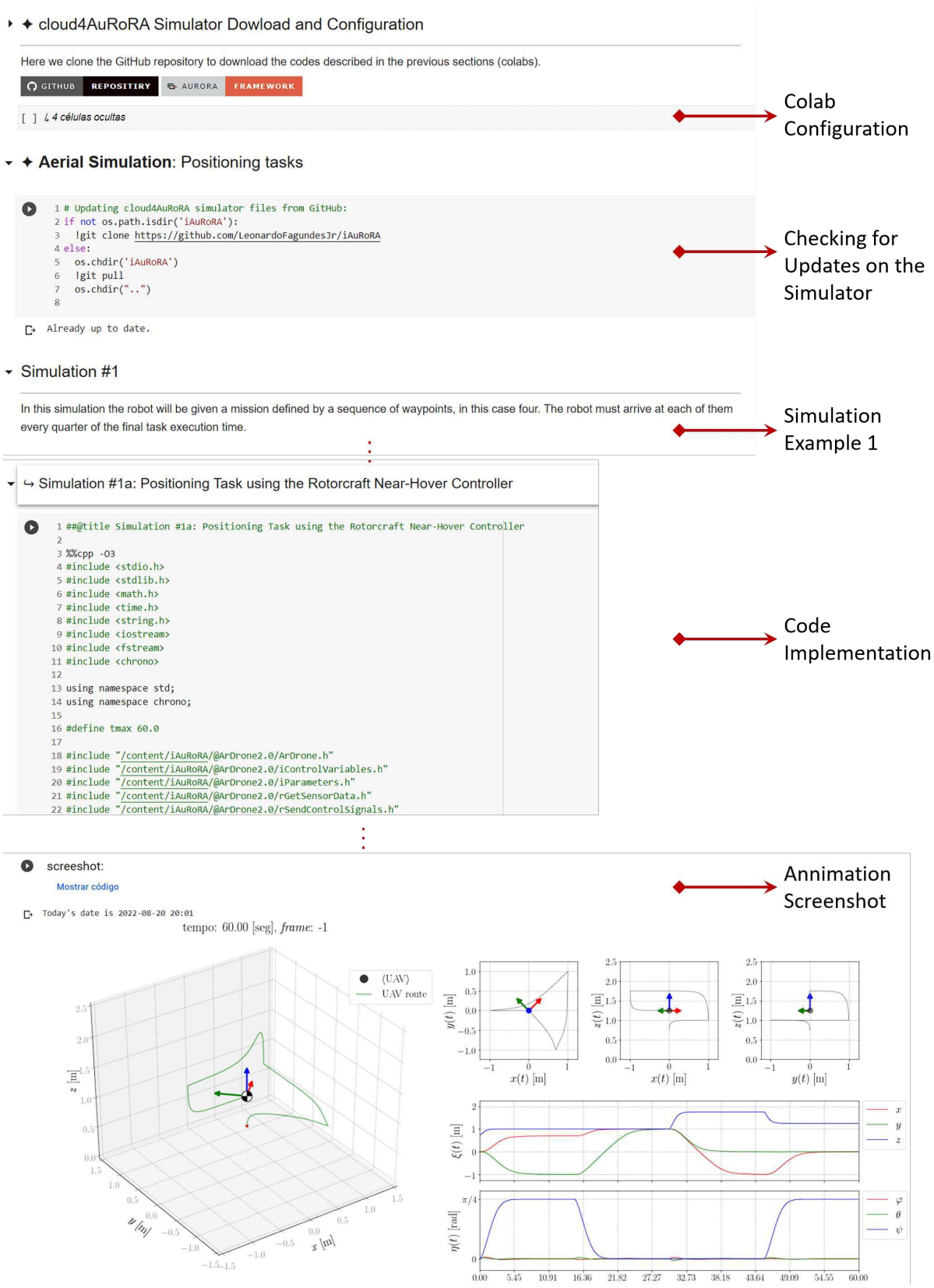


Figure 17 – Colab sections where the simulation and analysis of the obtained results is performed, generating static and animated graphics, as well as videos of the simulation, by means of already created functions that can be easily changed.

The use of the cloud4AuRoRA platform can be easily extended to the application and development of control strategies for path planning and tracking, robot cooperation, load transportation, and homogeneous and heterogeneous robot formation, as is done in AuRoRA in recent research involving the platform, as well as in Pizetta, Brandão e Sarcinelli-Filho (2019), Neto, Brandão e Sarcinelli-Filho (2020), Bacheti, Brandão e Sarcinelli-Filho (2020). Furthermore, with a specific focus on teaching, one can use the tool as a laboratory step in introductory robotics subjects where the student does not find the need to install additional applications since the tool runs in the browser, and can create a personal copy of Colab and alter it as they see fit to learn and explore the concepts learned in class.

The platform can be used as an evaluation system through the use of Gradio [5], an interactive environment linked to a section of Colab to collect student responses. In the case of cloud4AuRoRA, Gradio can be used as a tool for students to enter their controller code and observe the robot's response graphs during their evaluation. The general structure of the platform to support Gradio would make it possible to control and use it through mobile devices, as shown in Figure 18, and the teacher would have as feedback the overall result of the class in a simple way.

[5] https://gradio.app/quickstart/

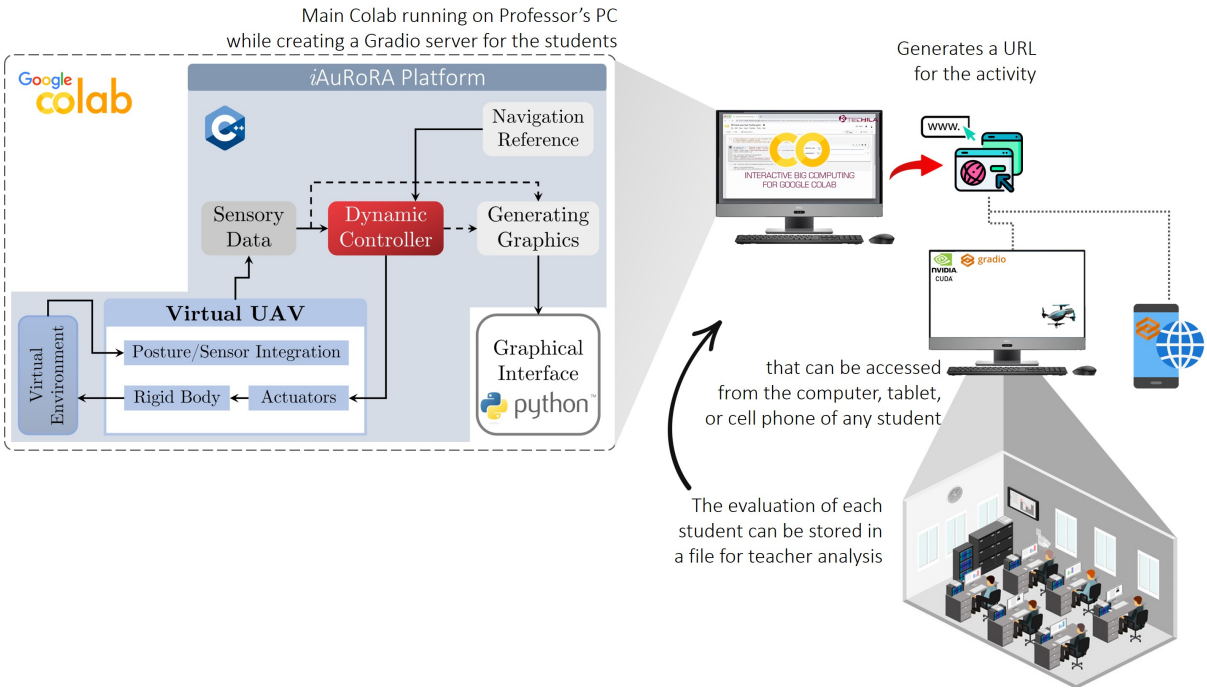


Figure 18 – Structure of cloud4AuRoRA platform using Gradio for education, running notebooks on the student's cell phone.

2.3 Comparing cloud4AuRoRA and AuRoRA Platforms

In this section the could4AuRoRA usability is discussed in advanced research and classroom when compared with AuRoRA, highlighting the possibility of extension the implementation of others common robots in a easy way. The interactive tool proposed works as a "live book", where some explanation about robot dynamic and kinematic models, task definition, sensory data modeling, and control design and evaluation, are provided.

As mentioned earlier, cloud4AuRoRA is based on AuRoRA platform, which is hosted in MATLAB and has a communication interface with simulated and real robots. Since both platforms have a simulation version, it can be implied that one replaces the other. However, the platforms have their individual differences and advantages that complement each other. Table 1 presents a comparative description of both platforms. Notice that they are not mutually exclusive, but rather reinforce each other for the advanced development of research, extension, and teaching.

Table 1 – A brief comparison between AuRoRA and cloud4AuRoRA.

	AuRoRA	cloud4AuRoRA
Support for experimentation on real robots	✓	
Communication with ROS Master	✓	
Implemented robots:		
ArDrone 2.0 Parrot	✓	✓
Tarot T-650	✓	
Bebop 2 Parrot	✓	
Anafi Parrot	✓	
Pioneer P3-Dx	✓	
Implemented sensors:		
Laser LMS 11	✓	
Joystick control	✓	
Automatic generation of figures	✓	✓
Graphical simulation environment	✓	✓
Programming in		
MATLAB	✓	
Python		✓
C/C++		✓
CUDA		✓
Automatic generation of the simulation video		✓
Combines narrative, code, and data analysis data in the same document		✓
Requires installation of proprietary software	✓	

As one can be observe in Table 1, the AuRoRA platform is able to communicate with real robots and create, within MATLAB, a ROS node to communicate directly with the ROS Master and with the robots that are connected to the server. Since the cloud4AuRoRA platform was developed for exclusive use in simulations in its first version, the ArDrone 2.0 aerial robot was chosen as a pilot model to validate the environment. Thus, the entire tool was built and validated using its UAV. For cloud4AuRoRA to be extended for communication with robots, one should check the possibility of creating a cloud server where ROS nodes will be able to publish and subscribe to topics that the ROS Master adds on this server. However, communication involving robots is a delicate matter and must be taken with great care. To use this tool to communicate with the real robot, one needs the internet, which can often be unstable, with loss of signal or problems sending and receiving information, and consequently causing accidents. Currently, the experiments performed by NERo use a wi-fi router specifically for communication with robots, mitigating data traffic problems due to the excess of users connected to the same network.

Once the platform is validated, one can easily extend its use by implementing the models of the other robots used in AuRoRA. This adaptation is simple due to the fact that the whole structure of AuRoRA has been designed as an object-oriented `class`. This means that robots, sensors and simulator environments have been implemented as classes, just like the ArDrone, so they will undergo a similar process during the conversion from MATLAB to C/C++ and Python code, with a few modifications particular to each case.

Joystick control is used in AuRoRA for flight demonstration and for emergency situations, where it is possible to recover control of the robot in case it exhibits undesired behavior. Since it is possible to interact with Colab using peripherals such as cameras, keyboard and mouse, it seems to be possible to use the joystick in the next version of the platform.

Despite the difference between the platforms, it is possible to verify that they are complementary. Because the tool is natively sequential, parallelizable applications will not gain as much in AuRoRA. This discussion is left as an application in the Chapter 3, but it should be noted that parallel programming in MATLAB requires the use of a specific toolbox, and depends on the settings of the machine that is running the program.

Finally, the main point highlighted in this thesis is the use of a platform capable of integrating narrative and context (text, figures, mathematical equations, discussion and analysis of results) with coding of the theory presented in an iterative way where the user is able to change parameters, rotate graphs, move cells around, etc., as well as running in the cloud with no membership fee. Although MATLAB provides a similar tool, the Live Editor, it is still limited by the need to install and purchase the product.

2.4 Conclusion

This chapter presented the first version of the cloud4AuRoRA platform as a complementary tool to AuRoRA, running in the cloud, as an open-source tool, free to access, and without the need to install any additional software because it runs in the web browser. cloud4AuRoRA is an iterative mobile robot simulator hosted on Colab, and can be accessed from various mobile devices.

Once validated, the platform can be extended for implementation of other robots, including robotic manipulators. From there, its use will contemplate engineering teaching activities, up to basic research developed in academia in the area of robotics.

3 Researching with cloud4AuRoRA

Unmanned aerial vehicles (UAVs) are considered as a valuable source of data, acting as a mobile sensor that can achieve unsafe places for human operators and collects data efficiently. However, due to the need for communication, during experiments on real-world robotic platforms, it can arise the problems of time-delay or even packet loss in communication between a robot and the ground control station (GCS). Specially when leading with UAVs in long-distance missions, these challenges can evaluate unsafe parameters that lead to safety-critical system failures and can deteriorate the robot performance or even destroy the system (CASAS; MITSCHELE-THIEL, 2018). In situations like these, it is important to know about how the robot will behave in order to anticipate and avoid dangerous displacements (TANG et al., 2019).

As a UAV is a complex system in which electromechanical dynamics is involved, the robust and reliable response of the controller system is an essential requirement. Selecting the right tuning parameters for controller algorithms is a prevalent problem in robotics system control that can significantly affect the performance of the robot when performing its missions. Designing effective low-level robot controllers often entail platform-specific implementations that require manual heuristic parameter tuning (SERRANO-PÉREZ et al., 2021), which takes long design times. By allowing a simulation to fully explore the cost space offline, certain states and actions can be constrained or isolated. Data is fit with simple models relating the desired commands, optimal control actions, and robot states to identify new parameters candidates for the controller gains. Automatic parameter optimization algorithms, such as online reinforcement learning (WEN et al., 2019) or Bayesian optimization (YUAN; CHATZINIKOLAIDIS; LI, 2019), have been used to automate this process. However, this approaches still requires large computational power and long development time, specially in reinforcement learning methods due to the training process. This type of analysis must be performed in simulation, mainly because the time delayed information can cause unpredictable and unsafe robot behavior during the experiments' execution. Predicting the time delay in the communication system can be considered one of the biggest challenges associated with autonomous navigation.

Combining the need for parameter calibration of the control system, as well as the parametric uncertainty in the modeling of the robot, and the time delay in the communication system, new challenges arise and the analyses may require more computational power with the increase in the amount of parameters to be studied. This analysis can be executed offline to evaluate the robot performance when executing a specific task (WADA; ARAUJO-ESTRADA; WINDSOR, 2021) and then improve the real-time mission execution by correctly tuning the control law parameters. This type

of analysis requires massive amounts of dynamic simulations to determine the possible controller gains that enable the robot to perform its task in a timely manner. So, it is necessary to implement the applications efficiently and using computational architectures that enable parallel programming.

GPU-accelerated computing delivers unprecedented performance for applications by moving certain processing-intensive parts of the code to the GPU (graphics processing unit), while the rest of the implementation continues to run on the CPU (central processing unit). From the user's perspective, applications simply run significantly faster due to the fact that the architectures rely on thousands of cores to efficiently process parallel workloads. A simple way to understand the difference between a CPU and a GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing, while a GPU consists of thousands of smaller, more efficient cores designed to handle multiple tasks simultaneously.

This chapter proposes to study the effects of delay in receiving information from a quadrotor without paying attention to its sources using GPU-accelerated simulations to improve the analysis performance. In summary, the objective is to observe the robot behavior under the condition of communication delay and to propose a method for state evaluation and parameter tuning of conventional controllers in consideration of the presence of time delay while executing positioning missions. The dynamical characteristics of a quadrotor can be analyzed to design a controller which aims to regulate the posture (position and orientation) of the quadrotor. Due to the difficult and long time spent in analytical control tuning in parameters space (a few hours in MATLAB), this chapter 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.

This chapter is an extended version of the previous work (FAGUNDES-JUNIOR et al., 2021). Considering a design space of two parameters, the execution time is reduced by refactoring the code to optimize the GPU register usage. The design exploration order, the MATLAB GPU support, and the MATLAB C code generator are also explored. Finally, the possibility to evaluate up to 12 parameters with negligible runtime overhead is demonstrated. The main contributions are summarized below:

- A GPU-accelerated UAV simulator built and running on the web, to facilitate rapid robot modeling and analysis of control strategies, as well as its use for education and research.
- Experiments on massively distributed simulations of thousands of UAV simulations on CPU and GPU settings.

- Improvements in the required time for analysis for various challenging locomotion tasks, control parameters tuning, and time delay in communication running task in less than 6 seconds on a single machine. Running on a cloud computing source available as free and open source.

It is worthy mentioning that the focus here is on the application of GPU-based physics engine to massive aerial robots simulation and control tuning in parameters space configuration, and not on comparisons and benchmarks of the physics engine itself.

This chapter is organized as follows. Section 3.1 discuss about the GPUs and possibility of accelerating applications, in special, MATLAB implementations, using HPC strategies. Section 3.2 provides a brief description of the UAV modeling and control. In Section 3.3, the control tuning evaluation in parameters space, under time-delayed communication, is presented. Section 3.4 presents the proposed HPC control tuning and analysis method highlighting the adopted task parallelism and its extension. Section 3.5 shows the results and provide a few examples and comparing the performance analysis using MATLAB, C/C++ and CUDA simulator versions. Finally, Section 3.6 discusses the findings, concluding the paper, and addressing directions for future work.

3.1 Background

Regarding hardware resources, performance, and computing limitations, complex implementations that require a higher level of computational rigor can become impractical in a timely manner. Those that require extensive amounts of matrix computation, are susceptible to acceleration and parallelization of code, which has been widely studied in recent years (HSU; TSENG, 2021; STONE et al., 2007). With the technological advance in computing platforms, the number of cores inside a compute node and the number of nodes across the whole system have increased dramatically.

3.1.1 Accelerating MATLAB/Simulink Implementations

The mathematical models describing the dynamical interactions used in the above systems can be simulated with the use of powerful software such as MATLAB, however, for large–scale simulations software begins to collapse (SOLEIMANI, 2021). MATLAB/Simulink is a widely used simulation tool for rapid prototyping and algorithm development. Many industries laboratories and research institutions face growing demands to run their MATLAB codes faster for computationally heavy projects after simple simulations. At this point it is important to optimize the implementation so that it runs in a short time. However, it is not always possible to reduce the execution time of the application as a tool with a high-level programming language. MATLAB is a very powerful tool for many applications, mainly because it specializes in working with matrices.

In practice, the software becomes especially useful in application development and in the proof-of-concept or validation of strategies and models, especially in robotics (eg, Reference (POPOV; SAYARKIN; ZHILENKOV, 2018)). However, due to the desirable system requirements for running applications on the software, the computational cost can be an impact factor, making unfeasible a number of real-time applications and implementations, or even massive testing or training of artificial intelligence models, which perform thousands of sequential calculations. Some function, as loops operating on matrices in MATLAB tends to be slow (REIS; BISPO; CARDOSO, 2020).

The great advantage of using MATLAB is the extensive libraries and toolbox available, and ready-to-use functions that by themselves greatly speed up the development and execution time of projects, especially the more complex ones. Once the application has been developed and validated in the MATLAB environment, it is possible to migrate the implementation to platforms that do not require as much computing power and to use high-performance computing strategies to improve the performance of code and application execution. One approach is to transpose the code to C/C++ languages, this change alone will bring great performance gains. However, since these are very low-level languages, where it is necessary to implement most of the functions that are ready in MATLAB, it becomes more complex to generate graphics and do visual analysis, for example. An alternative is to save the relevant data in a .CSV or .TXT files and generate the necessary graphics using other languages, for example MATLAB itself or Python. Another solution that increases the performance of the application would be to transpose the entire code to Python and retain the entire implementation in the same environment. However, the Python implementation can also be slow, compared to C/C++ (LEISERSON et al., 2016). When the application is parallelizable, the use of CUDA (the NVIDIA Compute Unified Device Architecture) and C/C++ languages for GPU implementation can boost execution performance.

Some operations in MATLAB are slow, specially when the data size or loop repetitions increases, even running in GPUs. However, it is possible to optimize the implementations using vectorization for parallel processing, prealocation for efficient memory management or nested loop for matrix column by column instead of line by line, for example. In addition, some optimization can be done using C-MEX files (MATLAB functions written in C/C++), the basic starting point for using CUDA and GPUs. These functions are dynamically loaded as a function during MATLAB sessions. Although the parallel computing toolbox from Mathworks and other third party CPU toolboxes provide the CUDA interface, many constraints and limitations hinder full utilization of CUDA and GPUs (SUH; KIM, 2013). Another way is the optimization through profiling to improve performance, examining the MATLAB built-in profiler to find the bottlenecks in m-files. From profiling, it is possible to find where the code consume more running time and identify the bottlenecks in the implementation. Since many big codes have multiple layers in

practice, it is not straightforward to find functions that, in turn, call other time-consuming functions. And, one can find this situation several layers down in the codes. Through the profiling process, one can efficiently determine which codes are responsible for such calls. This is an essential step in optimization planning.

Since MATLAB uses a vector/matrix representation of data, which is suitable for parallel processing, it can benefit a lot from GPU acceleration (SUH; KIM, 2013). There exists some ways to accelerate MATLAB codes using GPUs in the platform. One of the most simple ways is to configure the application to run with multiple CPU core and use built-in functions. One can easily parallelize tasks using a `parfor` instead of a for-loop (STRIPINIS et al., 2021), or one can data transfer from a CPU to GPU and vice versa using `gpuArray` and `gather` commands and run in a GPU (SHABANINEZHAD; AWAN; RAMAKRISHNA, 2021). So, one can easily take advantage of graphics processing units (GPUs) using the `gpuArray` command by using the MATLAB built-in Parallel Computing Toolbox, and directly call NVIDIA GPU in MATLAB. The MathWorks provides useful tools for parallel processing from the Parallel Computing Toolbox. The toolbox provides diverse methods for parallel processing, such as multiple computers working via a network, several cores in multicore machines, and cluster computing as well as GPU parallel processing. One of the advantages of the toolbox is that one can use GPUs without explicit CUDA or c-mex programming (eg, using `gpuArray` allocation). However, this comes with a heavy price tag for users to install the Parallel Computing Toolbox.

3.1.2 Accelerating Applications in GPU Environments and Computational Notebooks

In data science and data analysis research and educational fields, the computational notebooks have been widely used due to its versatility in present code implementation in different programming languages (CANESCHE et al., 2021) and documentation using text, math (LaTeX), images, videos, interactive links, forms, among others (VALLEJO; DÍAZ-URIBE; FAJARDO, 2022). These tools support incremental and iterative analyses, enabling users to edit and collaborate, arrange, and execute small blocks of code in any order, creating a specie of a "*living book*"(RULE et al., 2019), enabling explanation of through processes by allowing students and researchers to intersperse code with richly formatted textual explanation, enabling increasingly complex analyses in a single file.

Researchers around the world adopt computational notebooks with the aim to not only perform, but also document and share their analyses. It is possible to find various types of notebooks on the GitHub as open source code and data (RULE; TABARD; HOLLAN, 2018a), in different platforms such as Jupyter Notebook, RNotebooks, Google Colab or Mathematica. Indeed, computational notebooks were designed to support construction and sharing of computational narratives. The Google Colaboratory, or just "Colab", is

currently one of the most widely used platforms and the one with the most explicit support for narrative. It is a free platform provided by Google that uses the Jupyter notebook environment in intention to disseminate and explore applications on the Web Browser using the Google computational power, without the necessity of installing software or compilers on users' computers. It can be run completely on the cloud and load and store data and code on GitHub and Google Drive with some command lines to be easily accessed and shared (BEG et al., 2021). It can be seen as a vital tool for promoting open science, facilitating the cooperation, and greater engagement with data.

Recently, Google Colab has been used in different research areas. It provides a runtime fully configured for deep learning and free-of-charge access to a robust GPU. When initializing a session in Colab, the user has access to a dual-core processor, 12 GBytes of RAM and 40-50 Mbytes of L3 cache. In addition, the user can have access to a TPU (Tensor Processing Unit) or a GPU (Graphics Processing Unit). If opening a new session, the user must configure the execution environment if they are going to use the TPU or GPU as an accelerator. This feature is important for running the machine learning codes. The user also has access to a file system with 30 to 300 Gbytes of disk space and a Linux terminal. Colab provides four GPU types: Kepler K80, Turing T4, P4 and P100, which are high quality and high performance architectures. Currently, a session can be used for free for 12 consecutive hours (Google LLC, 2010). Such innovations, freely available online, can improve research and education. Thus, this service can be effectively exploited to accelerate not only deep learning, but also other classes of GPU-centric applications.

Due to the computing power available in Colab, the environment is used for high performance applications and parallelization of CUDA code. Especially, machine learning and deep learning techniques have increasingly exploited the tool (SHARIAR; HASAN, 2020). All this enables greater dissemination of the platform and faster development of applications open to the scientific community, accelerating scientific technological advance. Recently, applications with robotics have been widely studied and implemented with Google Colab (FAGUNDES-JUNIOR et al., 2021; ŠTEFEK et al., 2021).

Due to the complexity and difficulty of performing long distance experiments with robots, simulation becomes a viable and necessary alternative. Simulation is one of the most important and necessary steps in the design of engineering projects, making it possible to represent and study the dynamics of real systems by means of computers and their data processing and display capabilities. In certain cases, it is extremely expensive and impractical to go directly to the practical experiments or direct development of an application. Thus, simulations are able to provide a versatile and effective tool. Some robotics applications require simulations with massive repetitions for validation or determination of more complex analyses, for example, Monte Carlo simulation (PHANDEN; SHARMA; DUBEY, 2021), control design in parameters space (COELHO, 2017), machine learning, modeling parametric uncertainty analysis (SEIFI et al., 2020), just to name

some. Due to the high computational cost required by such applications, most of them are realized through HPC strategies, mainly with GPU programming for hardware acceleration using parallel computing.

Robotics simulation scenarios has been improved by GPU and cloud computing services. In terms of UAVs, the problem of trajectory and path planning needs to be performing in real-time. However, the calculation process can be slower as the complexity increases. It suggests using some additional mechanisms, such as parallel computing, instead of the traditional methods. The authors, in Reference (TURKER; YILMAZ; SAHINGOZ, 2016), solved the problem of path planning optimization by simulated annealing (SA) algorithms, paralleling simulating multiple UAVs using CUDA. The path planning problem commonly is stated as a data-parallel problem. Deep learning (DL) and reinforcement learning (RL) requiring many parameters' adjustment during the training process for learning complex tasks, this is made by matrix operations, basically, what is the GPU's specialty. The authors in Reference (LIANG; AL, 2018), proposed an GPU-accelerated RL simulations as an alternative to CPU ones, that were executable in serial manner. The adopted approach show promising speed-ups of learning various continuous-control strategies for locomotion tasks of legged robots. Other works using GPU for accelerating applications can be cited. In reference (OROZCO-ROSAS et al., 2021), the path planning problem is solved using the potential fields approach. This solution uses GPU implementation in open multi-processing (OpenMP) and compute unified device architecture (CUDA). The work in Reference (LIANG et al., 2020) proposes a GPU-accelerated parallel robot simulations and derivative-free, sample-based optimizers to track in-hand object poses with contact feedback during manipulation running in real-time on the GPU.

In this chapter, a parallel simulation strategy is proposed to solve the problem of robot control tuning in parameters space performing more than one hundred thousands simulations using CUDA programming and C++ to perform the simulations and Python to analysis the results graphically. The robot modeling, control, task definition, and evaluations are performed using a free cloud service based on Jupyter Notebooks, called Google Colab, as an alternative to AuRoRA, a MATLAB simulator for robotics. This implementation was done mainly because it makes it possible to run applications in the cloud without the need to install specific software on the user's personal computer.

3.2 UAV Dynamic and Kinematic Modeling

As for the UAV adopted, it is an *ArDrone 2.0* quadrotor, from Parrot Drones SAS, which consists of a set of four engines positioned in the shape of a cross, which are independently driven. The collective variation in propulsion forces, resulting 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. The quadrotor has six degrees of freedom (DOF) as shown in Figure 19, where the robots' gravitational center position is described by robot reference $(x,\ y,\ z)$, and the roll (ϕ), pitch (θ) and yaw (ψ) angles are related to the drone's rotations around its axes x, y and z, respectively. Its control is performed in an under-actuated way, since it has a fewer number of actuators than the number of DOF (BRANDÃO; FILHO; CARELLI, 2013), using four control signals which are responsible for guiding the robot according to its current state and the mission definition. There is also an internal controller responsible for takeoff, hovering and landing (SANTANA; BRANDAO; SARCINELLI-FILHO, 2015; PISKORSKI; BRULEZ, 2012).

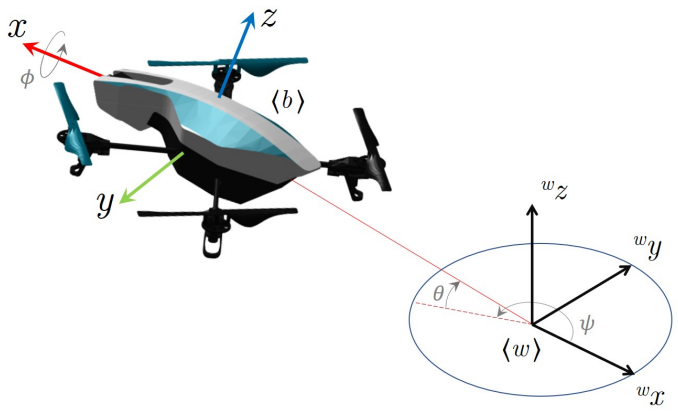


Figure 19 – ArDrone 2.0 description and pose variables, according to Tait-Bryan angles notation.

The command signals sent to the robot, are $\mathbf{u} = \begin{bmatrix} u_\phi & u_\theta & \dot{u}_z & \dot{u}_\psi \end{bmatrix}^\intercal \in [-1,\ 1]$, in which 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.

At this point, it is important to mention that this vehicle is one of the most complex stabilization and control test platforms (TANG; XIAO; LI, 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 (BRANDÃO; FILHO; CARELLI, 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 (BRANDÃO; FILHO;

CARELLI, 2013). Since in the modeling phase, a number of parameters appear during model acquisition, for exhibition purposes, the parameters used here also can be found in the aforementioned. This model was validated from a comparative analysis, of the input and output type performed between the real vehicle and the model, by the same work.

3.3 Robot Control Under Time Delay

3.3.1 Control System

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 (BRANDÃO; FILHO; CARELLI, 2013) a nonlinear controller, based on the ArDrone high-level dynamic model, is described. The four control signals, described in Section 3.2, are calculated by the control law and sent to the robot according to the block diagram presented in the Figure 20. It is beyond the scope of this research to present in detail the dynamic and kinematic model of the robot, and the implemented motion control law. The idea here is that the proposal is generalist, and can even be extended to other types of robots and controllers, analyzing the effect of time delay or not, as will be shown below. For this reason such information will be omitted, but the interested reader can access the available codes (both in the Google Colab platform and in the GitHub repository of AuRoRA, a mobile robot framework hosted in MATLAB), as well as find more information about the modeling of the UAV and the control used (in the Reference (BRANDÃO; FILHO; CARELLI, 2013)).

The interest here is to guide the movement relative to the global referential $x-y-z$, which that characterizes a control of 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 signals will be given by

$$\mathbf{u} = \ddot{\mathbf{x}}_d + \mathbf{K}_d \tanh(\dot{\tilde{\mathbf{x}}}) + \mathbf{K}_p \tanh(\tilde{\mathbf{x}}) \tag{3.1}$$

as shown in Figure 20 in the UAV controller block. The gains $\mathbf{K}_d$ and $\mathbf{K}_p$ will be exploit by the simulation scenarios to tune the parameters space as details in Section 3.3.3. If the position and velocity errors are null or very small (i.e., the drone is on the trajectory or at the target point), the drone's acceleration will be equal to the desired acceleration ($\ddot{\mathbf{x}}_d$), i.e., it will navigate with the desired acceleration to continue following the trajectory. While performing a trajectory tracking mission, if the desired acceleration is null, the drone will reach the trajectory but will not be able to anticipate where the trajectory is moving. So it will tend to stand still until the trajectory moves away from the drone, and it moves again to reach the time curve by *feedback* of velocity and position (an instantaneous

error will appear after all). The intention here is to provide a *feedforward* signal for the control to anticipate the quadrotor's behavior.

Note that the control system do not consider the time-varying in its design. Then, the study proposed here is important to highlight how the robot and the controller behave within the time delay in sending robot sensory information. In this scenario, the robot communicate with a GCS using a wireless link, to send and receive information (see Figure 20). The control signal output is a reference acceleration, which will be executed by the robot in simulation. From the control 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 generalization of the results, delay intervals proportional to the sampling period of the simulated air vehicle (*ArDrone 2.0 Parrot*) were selected.

The Figure 20 illustrates the control loop used, with the reference position $\mathbf{x}_d$ (the waypoint, 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 (BRANDÃO;

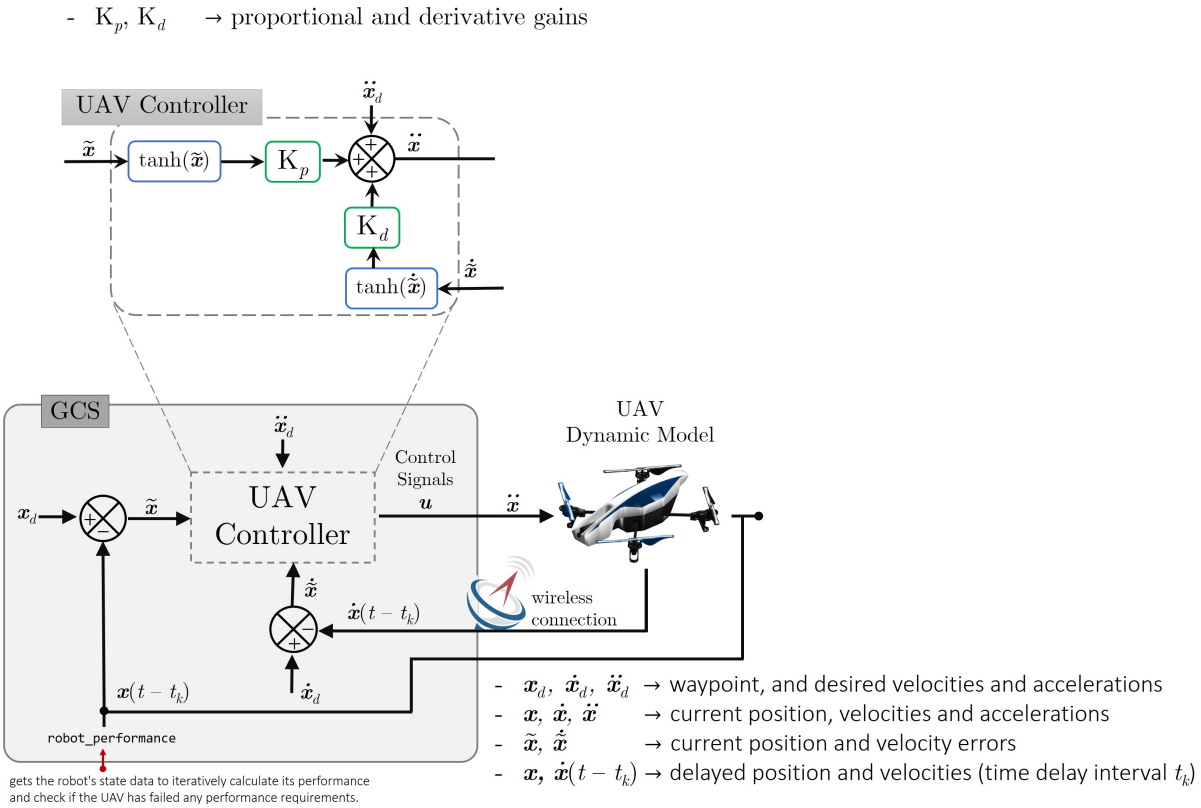


Figure 20 – Controller Model send/receive data to/from the Drone after delay t_k.

FILHO; CARELLI, 2013). In the GCS, one computes the control signals sent to the robot and analyze the robot performance computing some performance index defined to conclude about the asymptotic convergence and the robot's stability during the task execution, while evaluating if the UAV tends to not accomplish the mission in a stable and timely manner. It is worth mentioning the remarkable importance of simulation in the field of robotics and development, especially in these cases where you want to evaluate the robot's behavior under the effects of unmodeled and unpredictable disturbances. In this specific case of outdoor navigation in the presence of communication time delay, the behavior of the robot can generate dangerous situations both to the drone and to human life, potentially causing disasters.

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. Thus, it is worth mentioning that the strategy proposed here is generic and can be applied to any control where one wants to study the variation of its parameters, considering or not the communication time delay.

3.3.2 Solution Representation and Analysis

In this study, a table containing the parameters' combination that satisfy the required performance is used to represent solutions in the algorithm. Figure 21 illustrates this representation for one waypoint and three UAVs (or simulations running in different threads). Node **H** in the graph shows the **Home** which represents the single take-off and landing point for all UAVs in the system. Note that here is presented a single target point for the positioning task. However, the approach can be easily extended for another trajectory tracking or path following without lose of generality. To do this, more nodes are added and the correct control strategy is implemented for the analysis. It is, therefore, important to note that numbers 1, 2, and 3 used in the solution are reserved for 3 UAVs in the system simultaneously simulated, and each of them indicates the same take-off and landing point, **Home**. A solution of a simulation analysis is formed by the set of parameters if the robot accomplish the mission by complying with the established performance requirements. Individual routes for each UAV as it is illustrated by merging *UAV-1 route*, *UAV-2 route*, and *UAV-3 route* in Figure 21, so each element in the solution denotes a geographical position to be visited. The different robot behavior observed in the curves in blue, orange, and green are related to the control gains selected and a given time delay.

Representing what is happened when various robots with different control parameters and delay in communication are implemented, one can observe that, for UAV-1, its control gains can mitigate the effect of delay and the robot conclude the mission in a stable and timely manner. The UAV-2 perform a shorter path and execute its mission

during the time stipulated. Note that one cannot conclude yet that the UAV-1 spent much more time than UAV-2 or vice-versa. As seen and discussed latter, as the time delay increases, the controller gains tend to decrease so that the robot has a slow movement and compensates for the delayed information. In addition, the UAV-3 presents a completely unstable displacement, it occurs when the time delay is high and the controller gains are high too. So, the control system tends to have rapid responses with higher control signals due to the large initial positioning error, however, the GCS receives delayed data, it takes the robot to move fast while the GCS estimates that the robot is stopped. In resume, the error will maintain high, and the control signals has to give high values for velocity to make the robot achieve the goal.

During the simulation, the GCS store the robot data for future analysis (`robot_data`) and calculate the performance indexes (`robot_performance`) while the robot navigates. At the end of the simulation execution, the algorithm will get just the controller gains for a determined time-delay combinations and store it in a table. This is the solution of the proposed approach. This data is used for analysis what parameters combination can handle the delay well according to the mission's requirements.

In this section, the effects of changing the control gains are studied according to the time delay in sending information by the drone, however, it can be use just to calibrate or tuning a controller system, or even, to study the parametric uncertainty while modeling the robot dynamical behavior.

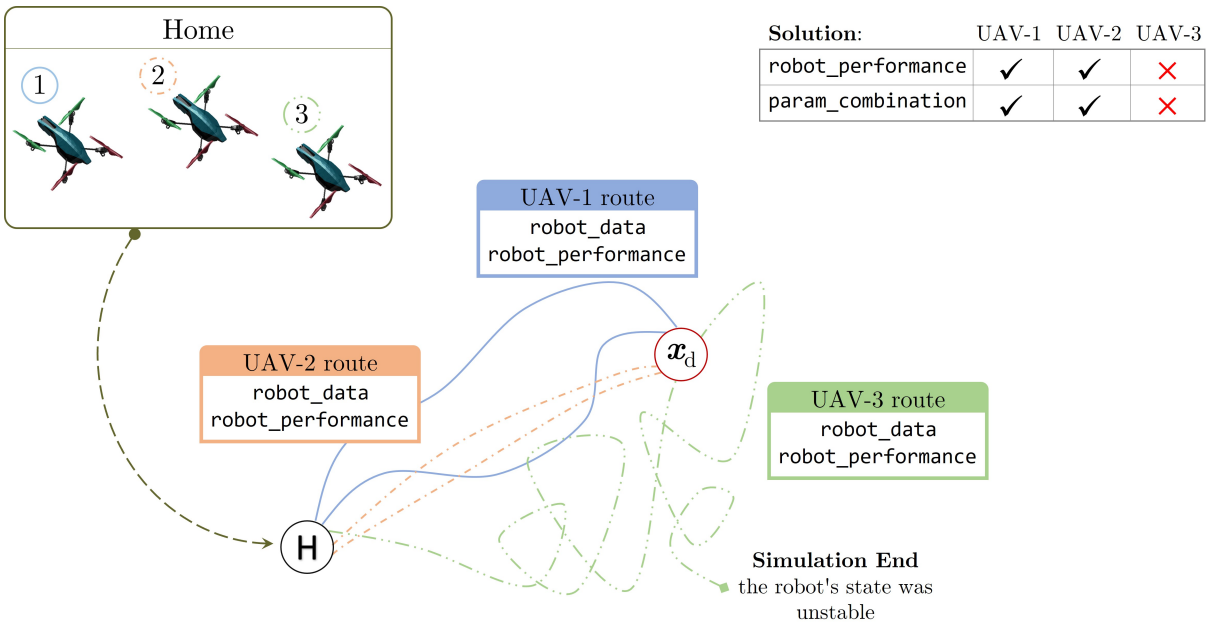


Figure 21 – Solution representation for a multi-UAV system with 3 UAVs and 1 waypoint for positioning missions.

3.3.3 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 (COELHO, 2017; ZHU; 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 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 subjected to time delay.

3.3.4 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.

3.3.5 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

$$\text{IAE} = \int_0^{t_f} \|\tilde{\mathbf{x}}\| \, \mathrm{d}t \tag{3.2}$$

and

$$\text{ITAE} = \int_0^{t_f} t\|\tilde{\mathbf{x}}\| \, \mathrm{d}t, \tag{3.3}$$

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; SARCINELLI-FILHO; BRANDÃO, 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.

3.4 High-performance GPU Implementation

The robot model and the controller are first implemented 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. This strategy was adopted to maximize the performance gains. In addition, the implementation is available in Google Colab format. Therefore, documentation, open source approach, graphical output (to visually exhibit the simulation results and important graphics) are provided, running much easier and faster while sharing reproducible results.

In the proposed notebook, the well-know AuRoRA simulator is implemented 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, and load transportation. It can be done in the simulation platform and also run in real environments with the physical robots in the laboratory.

3.4.1 GPU-Accelerated multi-UAV Simulation

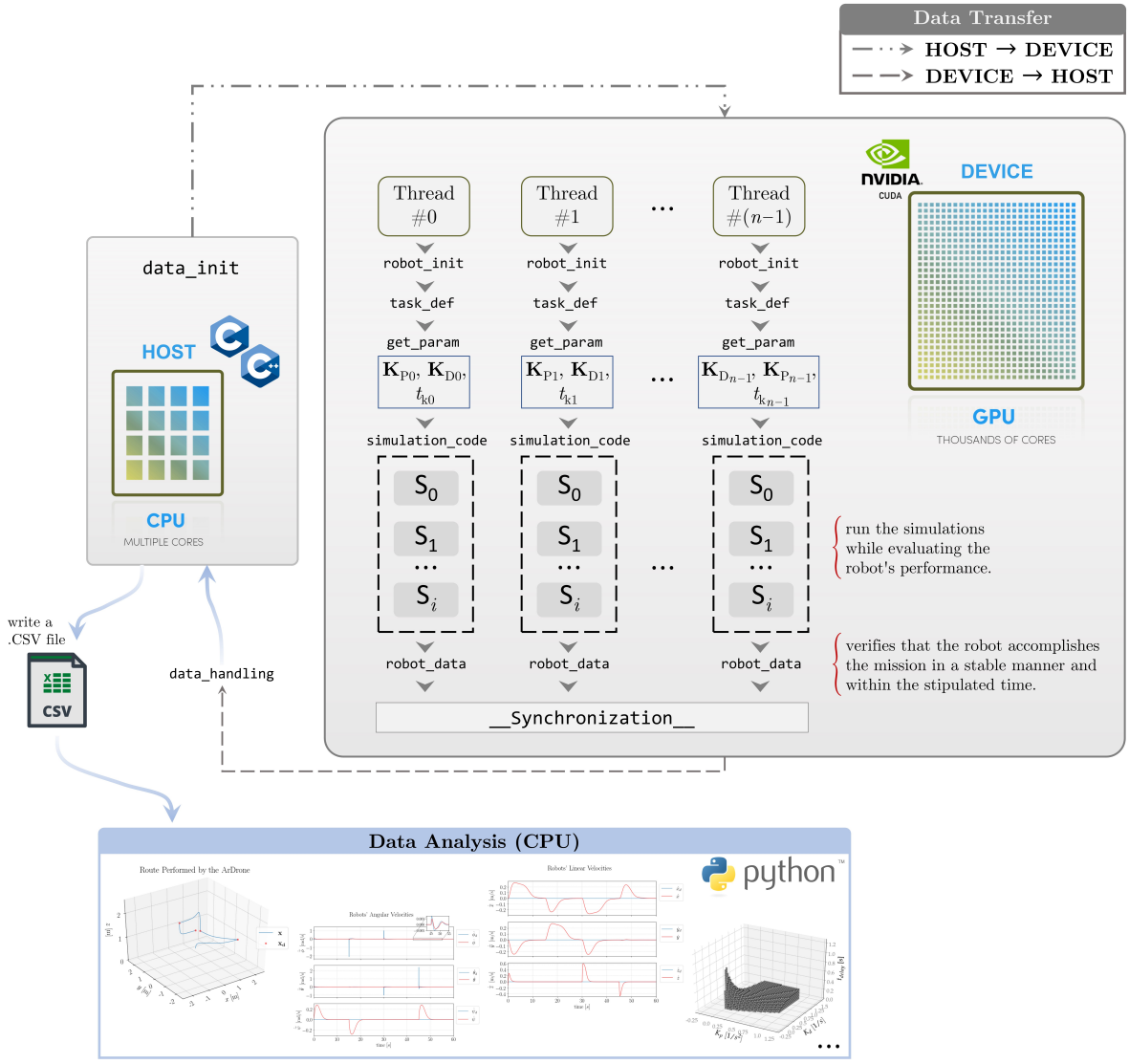
The implementation of parallel UAV navigation algorithm for control tuning consists of one major nested loop as it is presented in Figure 22(a). These are *Map Loop* which creates a table with all the possible combinations between the control parameters and the time-delay to be accessed by each simulation, and *Simulation Loop* which all threads in the GPU are responsible to execute, it as a CUDA kernel to simulate the robot navigation when executing its mission and analyses its state in each sample time, verifying if a certain parameters' combination is unstable, or it takes a long time for the robot to execute the mission. In the HOST, the code just initialized the parameters' combination data (`data_init`), preallocates memory spaces for the solution, transfer it to the DEVICE and get the solution from the GPU (`data_handling`) to save it on a .CSV file using C/C++ programming language. So, generate the figures for analysis and discussion.

In the DEVICE, the robot is initialized as a `struct` model, calling the robot's parameters and state (`robot_init`) and defining its task with the module of task definition (`task_def`). The controller parameters are accessed from the GPU global memory (`get_param`) and run the simulation (`simulation_code`) until the robot complete the mission or the times over, or even, if the robot could not achieve its goal and the code stops identifying that the robot performance could not meet the requirements' performance metrics (`robot_data`).

In addition to cooling scheduling, this loop is also responsible for selection of the solutions within the solutions found and returned by all CUDA threads. The solution is cloned into an array with size of number of threads in order to comply with data-parallel computation mechanism of CUDA. This is because all CUDA threads should perform their computation on their own data portion to exploit data parallelism. Then, this solution array, which is composed of the solutions for parameters combinations that are stable and converge the robot states to the desired states, and the time delay associated to this control gains.

In Figure 22, the blosk S_i represents the i-th iteration of the simulation, wich are executed in a serial manner. The robot sample-time is equal to $t_s = 1/30$[s], once the defined time for the robot conclude the task is 60 [s], it will have to perform a maximum of 1801 iterations during the simulation. If the robot accomplish the mission in a stable and timely manner, it will execute them all. In the other hand, it is need to be observed that if the robot performance diverges before the configured mission time, this thread will not execute 1801 simulations, it will stop when identify that the robot does not obeyed one of the performance requirements.

The UAV positioning simulation, as a CUDA kernel, is executed by an array of threads in parallel on the GPU. This kernel is invoked at the main function, as it is provided in Algorithm 3, (Line 7). The kernel is responsible for executing an entire simulation for a given combination of control parameters and time delay in control feedback information. As it is presented in Algorithm 4, the kernel parameters are preallocated in the CPU by the main function. The *solution array* mentioned above is represented as the parameter *parData* in Algorithm 3. Each thread gets its own solution `ctrl_parameters` from global memory to its local memory by using its unique id *idx* as the index of *map*, (Lines 2 and 5). Then, the algorithm starts initializating the robot representation and the target definition. At each simulation iteration, the robot state is obtained, and the control signals are computed and sent to the robot. The robot dynamical behavior is collected by its sensor's system, however, the control loop receive this data with a delay. During the simulation, the robot performance is evaluated. If the robot do not fail any of the performance requirements, it will continue to navigating until it accomplish its mission. However, if one of the performance requirements is not met in one iteration, the algorithm will assume that simulation to be unstable or not executable in a timely manner, and will



(a) Overview of parallel UAV control tuning under time delay communication.

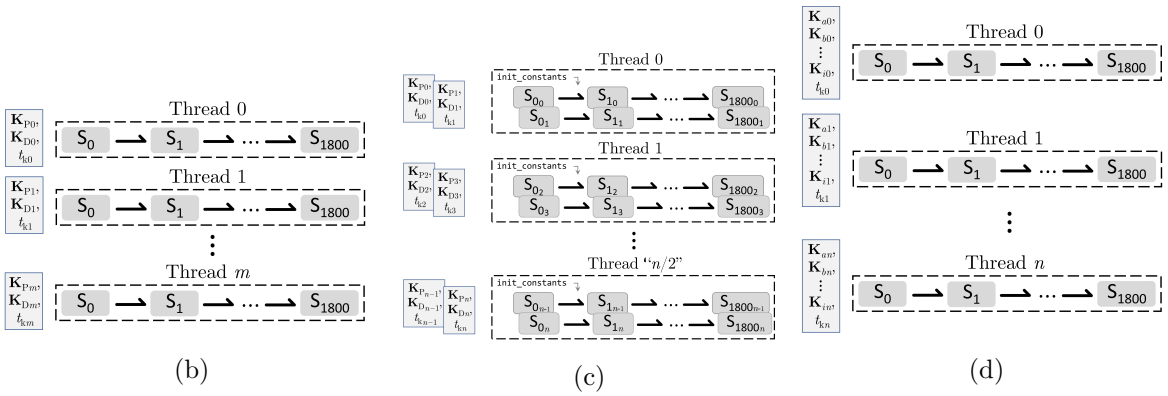


Figure 22 – Paralelization of the analysis; (b) Task parallelism, one task per thread; Ways to extend the framework: (c) two points for thread and use of constants and calculation, and (d) possibility to analyze multiple different parameters and controllers without loss of generality (which is unfeasible or unattainable in MATLAB).

terminate the simulation before the predefined time for execution. The thread will "kill" the kernel execution and is free to run another simulation with another combination of parameters.

These operations are repeatedly performed in the GPU by all threads until a predefined number of iteration, as referred to S_i, is reached or until the robot diverges. At the end of the simulation, each thread has its own candidate solution stored in the table (*parData*) for analysis on the host global memory as provided in Figure 21 and should wait for each other to synchronize for `deviceToHost` data transfer. After these candidate solutions generated at the end of the simulations by all threads transferred from GPU to CPU, the host continues its own operations in the sequence by saving this data in a .CSV file and selecting just the parameters and delay combination as the solution, and so on. This file is used to provide figures and graphs for analysis in Python.

Implementing the time-delayed control in parameters space, the approach consider that each GPU thread executes an entire simulation scenario as shown in Figure 22(b), where the task parallelism is explored since each thread has a different parameter $\mathbf{K}_d$, $\mathbf{K}_p$, and t_{delay}, as details in Section 3.3.3, 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 23(a) shows a simulation result visualization generated inside the proposed Google Colab notebook for the control system aforementioned. 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 23(b) 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

Algorithm 3 GPU-accelerated multi-UAV simulation and control tuning.

Require: C++ and CUDA plugins

function TIME_DELAY_ROBDATA(`N_BLOCKS`, `N_THREADS_PER_BLOCK`) ▷ Runs on CPU
  `NUMBER_DRONES` ← `N_BLOCKS`*`N_THREADS_PER_BLOCK` ▷ Number of simulations
  map ← all combinations of gains and time delay intervals
  n ← number of iterations per simulation
  data ← matrix of zeros $\in \mathbb{R}^{n\times(12*\texttt{NUMBER\_DRONES})}$ ▷ the robot state is a vector with 12 variables
  parData ← matrix of zeros to store the solution of the analysis
  kernel <<< `N_BLOCKS, N_THREADS_PER_BLOCK` >>> (*data*, n, *map*, *parData*); ▷ Kernel Call
  csvFile ← *parData* ▷ Data Handling for future analysis in Python
  Return *csvFile* ▷ The robot's response/state for all simulations
end function

Algorithm 4 UAV Navigation Simulation as a CUDA Kernel

function KERNEL(*data*, *n*, *map*, *parData*) ▷ Runs on GPU
  idx ← blockIdx.x * blockDim.x + threadIdx.x ▷ thread id
  if *idx* < `NUMBER_DRONES` **then**
    `robot_initialization`
    `ctrl_parameters` ← &(*map*[*idx*])
    `idDelay` ← *n*; k ← 0
    ▷ pragma unroll
    while $t < t_{max}$ **do** ▷ During the simulation time
      if $k >$ `idDelay` **then**
        $\mathbf{x} \leftarrow$ `historic_robData.Position`$(k -$ `idDelay`$)$
        $\dot{\mathbf{x}} \leftarrow$ `historic_robData.Velocity`$(k -$ `idDelay`$)$
      else
        $\mathbf{x} \leftarrow$ `historic_robData.Position`(k)
        $\dot{\mathbf{x}} \leftarrow$ `historic_robData.Velocity`(k)
      end if
      $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_d - \mathbf{x}$, $\dot{\tilde{\mathbf{x}}} \leftarrow \dot{\mathbf{x}}_d - \dot{\mathbf{x}}$
      Inserts the delayed input into the control signal
      $\mathbf{u}_d \leftarrow \ddot{x}(t - t_k)$
      $\ddot{\mathbf{x}} \leftarrow \mathbf{u}_d$
      Numerical Integration to obtain the Reference State
      UAV Dynamic Model
      Calculation of the Performance Indices
      `historic_robData` ∪ Current Drone Data
      $k \leftarrow k + 1$
      $t \leftarrow t + t_s$ ▷ t_s is the robot sample time
      if the robot failing any of the performance requirements **then**
        break
      end if
    end while
    if comply with all performance requirements **then**
      parData[*idx*] ← *map*[*idx*]
    else
      parData[*idx*] ← **0**
    end if
  end if
  Return *None* ▷ The robot response/state
end function

blue curve represents the path performed by the robot during the simulation time, and the red points are the waypoints that the UAV must reach. An example of the results obtained in this analysis can be seen in Figure. 23(a), in which the black balls represent the $[\mathbf{K}_p,\ \mathbf{K}_d,\ t_{delay}]^\intercal$ combination that satisfy the conditions proposed. In the generated volume, one can guarantee the asymptotic and smooth robot convergence for a positioning task. Figure 23(b) shows a simulation with four waypoints, where the robot performed its rout described by the curve in blue and the red points are the desired points.

Note that this is a problem that aims to answer the following question: "Under communication time delay effects, how will the robot behave when varying the control law parameters?" This analysis is done within the parameter space of the controller by inserting the time delay in the control loop. Thus, given a defined sample for each parameter to be studied, their respective values are varied, generating a sequence of unique combinations

of parameters and time delay, in this case, $[\mathbf{K}_p,\ \mathbf{K}_d,\ t_{delay}]^\intercal$. This analysis must travel through a volume in a $\mathbb{R}^{n+1}$ space for a controller that one wants to evaluate the variation of n parameters, in addition to the variation of the delay time (responsible for increasing the dimension of the search space by one, n"+ 1"). In this work, the search is performed on a volume in three-dimensional space, $\mathbb{R}^3$, specifically, a cubic volume formed by all possible combinations of $\mathbf{K}_p$, $\mathbf{K}_d$, and t_{delay} (region delimited by the red dashed lines on Figure 23(a)). To study this application, it is necessary to examine this entire volume, which leads to the execution of thousands (or even millions) of simulations. The result of this analysis generates a black sphere, with a white border, for each combination of parameters that satisfies the performance requirements stipulated for the execution of the given task.

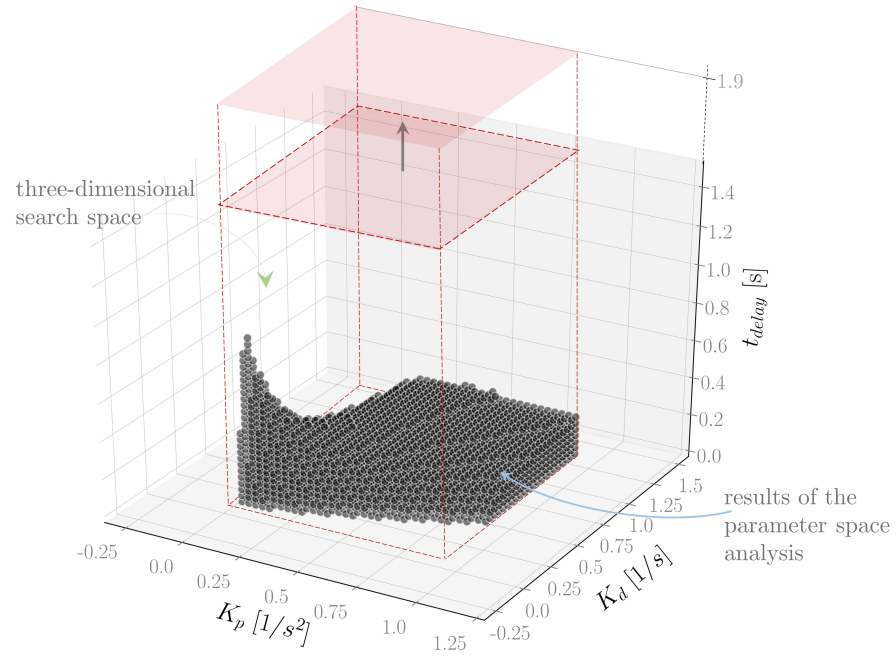
There are two main ways to perform this kind of analysis. These are: sequential execution and parallel execution of simulations. Since the search space and the number of simulations run can be as large as desired, the second approach is more suitable and can be easily performed with cloud4AuRoRA, as we will see below.

3.4.2 Analysis Expansion

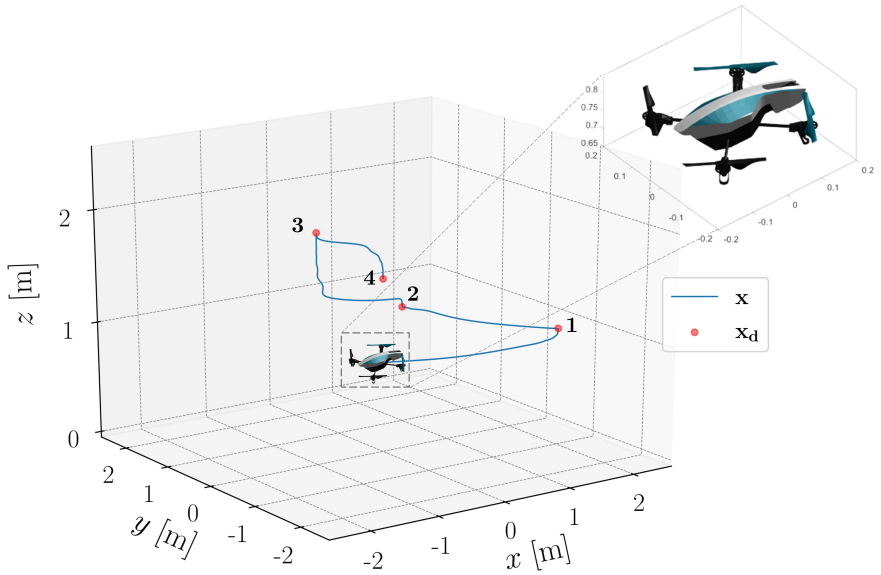
Because of the generalization of the proposal, it is possible to extend the approach in a few ways. The main ones are changing the robot model and the controller. In order to exemplify, two different case studies are considered.

The first is based on increasing the thread occupancy while performing more floating point operations. In this approach, each thread will execute two simulations simultaneously, but for unique combinations of input parameter for each simulation, as shown in Figure 22(c). In this way, the same analysis can be performed but with half of the total number of threads of the base case of Figure 22(b). The performance gain of this method is associated with the use of the constants and the physical parameters of the robot that are related to the constants initialized at the beginning of each simulation. That is, each thread will now initialize the drone only once and use a "virtual copy" of the robot to execute the same mission but with a different and unique combination of control parameters and time delay. In this case, each thread executes its own point and one subsequent point.

The second proposed expansion of the analysis method in this paper is to change the control system. The higher its order, the more parameters it must calibrate. Thus, Figure 22(d) shows a control system with i parameters in addition to the time delay. All the given combinations can be studied even if the number of parameters or the amount of calculation increases significantly. The result does not negatively impact the execution of the analysis so as to make it unfeasible, as would be the case when studying more complex controllers in MATLAB. In general terms, once the dynamic and kinematic models of the



(a)



(b)

Figure 23 – Control analysis in the parameters space: (a) Parameters space solution visualization. In the volume formed by the black spheres, one can guarantee that the robot is capable to accomplish its mission in a stable and timely manner; (b) Route performing by the UAV.

robot have been determined, the approach proposed here suggests that different types of analyses can be performed on the robot that are likely to be parallelizable.

To exemplify the ability to increase the amount of parameters, a new controller model, near-hover controller, based on inverse dynamics of the mobile robot was

implemented, which indicates that in this model there are more matrix operations and inverse matrix calculations. In this new model, there are 12 parameters in the controller, in addition to the time delay, making a total of 13 parameters to be studied in all possible combinations within the range of values delimited for each of the parameters.

As the complete simulation is by sequential nature, the amount of "live" variables needed to perform operations and calculate the control signals to be sent to the robot in each iteration increases as a result of the higher complexity of the code, increasing the lines of the .PTX file (file generated by the compiler when translating the code contained in the source file into microprocessor language). This will result in a consequent increase in the number of registers required for each thread. In the following sections it can be seen that this change increases the simulation time, but keeps it under 60 seconds, showing the potential of the tool compared to MATLAB, which by itself would make it practically impossible to perform all these analyses in a timely manner on any machine. The implementation in Colab brings several advantages, including the possibility of running heavy programs, on high-level and expensive GPUs, using minimal computation on the user's machine.

The mathematical formalism of this controller is not presented in the text because it is a long development with certain mathematical definitions that can cause confusion in the reader, which is out of scope of this manuscript. The interested reader is referred to the design of the control used here, which is described in the work of (BRANDÃO; FILHO; CARELLI, 2013).

3.5 Experiments and Simulated Results

3.5.1 Experimental Setup

UAV simulation algorithm was implemented both on CPU and GPU for performance comparisons. In addition, all code are implemented in Google Colab platform and in MATLAB. The hardware specifications are presented in Table 2.

Table 2 – The used hardware specifications.

	CPU Ryzen 7 1700	**Colab CPU Intel Xeon**	**GTX 1660**	**Colab Tesla K80**	**Colab Tesla T4**
Clock frequency (GHz)	3.0	2.2	1.7	1.25	1.59
Cores	8	2	1408	4992	2560
DRAM	64GB DDR4	12.7GB DDR4	6GB DDR5	24GB DDR5	16GB GDDR6

3.5.2 Implementation Strategies Evaluation

This study analyzes the proposed implement methods in comparison with the basis implementation in MATLAB, which spend much computational time to execute the entire analysis. The performance of the proposed implementation is evaluated in six platforms. First, as baseline, the execution time is measured 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 3. Then, a C++ implementation is evaluated 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, the code is evaluated on Google Colab processor, which is an Intel Xeon CPU @ 2.30 GHz with a Two-Core processor with 12.6 GB of RAM (DDR4), and 46 MB L3 cache memory. The execution time is reduced to 1 minute and 20 seconds, which is 406× faster than MATLAB©. Finally, the GPU implementation is evaluated in three devices. First, the two GPUs from Google Colab are analyzed: 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, a V100 GPU is evaluated by using the instance P3 AWS Amazon[1].

Table 3 – 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]	6.8	8.2	16.8	17.5	82.1	127.2	33371 (9h 16m 11s)	5.2	6.8
Speedup	4069.6		1906.9		406.5	262.4	1	4907.5	
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 3 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 24 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.

Based on the directly MATLAB to C++ and CUDA code translation, it can be proposed an optimization of the codes for the GPU, seeking the maximum use of the

[1] https://aws.amazon.com/pt/

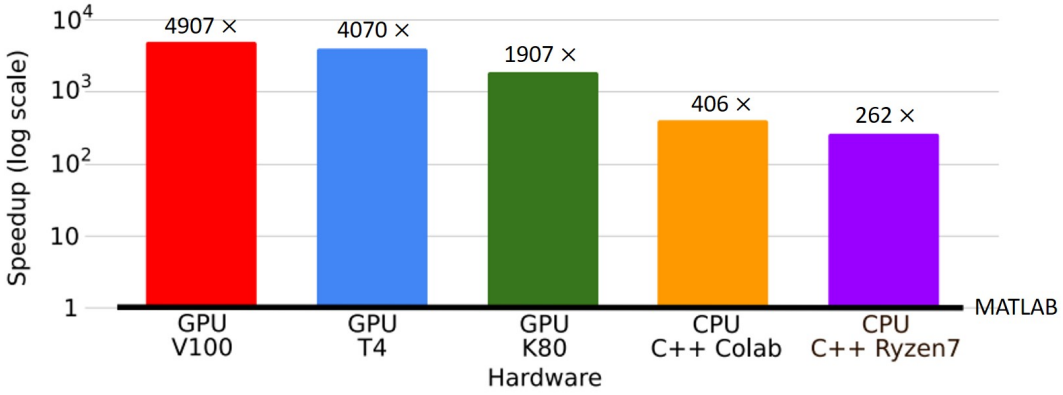


Figure 24 – Speedup factor in Execution Time in Comparison to MATLAB©.

potential of the different architectures presented here and the evaluation of changing more control parameters and different control strategies. For the following analysis, a handmade optimizations are performed. All the following implementations were made in the Google Colab environment and experimented for the two Tesla architecture GPUs available on the platform: K80 and T4; running 65536 ($= 2^{16}$) simulations. The first implementation is the direct MATLAB to C++ and CUDA conversion, which is named as "`dMAT2CUDA`". This version was presented in the previous work (FAGUNDES-JUNIOR et al., 2021). In this work, the identification of possible bottlenecks on GPU code are analyzed. In the first optimization, the manual code factoring is introduced. The original code performs a various quantity of multiplication by zero and uses some common trigonometric functions as "cossene", "hyperbolic tangent", and others, repeatedly. So, in this code version, more auxiliary variables are introduced to avoiding several repeated calculations that will result in the same value. Thus increasing the reuse of a single calculation to be used in several other operations throughout the code. In addition, all multiplications by zero implied were directly replaced by the real value, thus avoiding unnecessary calculations where the final result is already certain. This code version is called "`factMAT2CUDA`", due to the code factoring. The second change was made by reducing the use of generic functions by the direct calculus itself, such as multiplication and matrix inversion functions. All operations of these types were replaced directly by their respective mathematical definitions, instead of calling a function that would perform this operation. This last implementation is named as "`lessFunctMAT2CUDA`". Running these refactoring implementations, one get the following results shown in Table 4. A reduction of 18.9% and 21.1% are reached in execution time in comparison to the first version presented in (FAGUNDES-JUNIOR et al., 2021).

Both optimization proposals brought favorable results, in T4 the code execution time, including data transfer between memories, took about one second less if compared to the original model. At first this value seems to be insignificant, but noting the speedup one can notice an increase of 1000× in relation to the best implementation of the original version, on the V100 GPU (see Table 3). Some factors may have influenced the rapid

Table 4 – Code optimizations analysis for 3 cases implemented in GPUs T4, K80, and in MATLAB. FlOps (SP) means Floating Point Operations (Single Precision), and FlOps (DP), Floating Point Operations (Double Precision).

	`dMAT2CUDA`		`factMAT2CUDA`		`lessFunctMAT2CUDA`	
size of PTX [lines]	52802		2772		2744	
store local	772		20		20	
FlOps (SP)	1.4720e+11		2.9455e+10		2.9455e+10	
FlOps (DP)	2.0684e+10		3.7678e+9		3.7678e+9	
	T4	K80	T4	K80	T4	K80
GOps	24.7	10.0	5.9	2.4	6.0	2.3
Registers	162	159	130	128	137	135

		`dMAT2CUDA`				`factMAT2CUDA`				`lessFunctMAT2CUDA`			
	CPU	T4		K80		T4		K80		T4		K80	
	MATLAB	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total
Time [s]	33371	6.8	7.6	16.8	17.2	5.6	6.4	13.7	14.2	5.5	6.6	14.8	15.1
Speedup	1	4398.6		1942.4		5189.2		2354.6		5091.8		2209.6	
flow rate (Execution Time per Instance)													
Time [ms]	509.2	0.104		0.256		0.086		0.209		0.085		0.225	

increase in performance. Besides the reduction in register usage due to the increased use of repetitive operations through the use of auxiliary variables, it can be seen that the amount of "`load.str`" in the compiler's PTX code has reduced dramatically, as shown by the size of the PTX file and the operations *store local* by the number of lines found. Another important factor to note is that the number of floating point operations has decreased in the order of 10× compared to the original code, and consequently, the time per simulation also decreases, reaching values less than 0.1ms, while the MATLAB version requires more than 0.5s (5000× more). The difference is noticeable from the solution based on GPU acceleration.

The optimized code version that achieved the highest speedup was the "`factMAT2CUDA`", with an execution speed gain of more than 5189× when compared to the sequential version of MATLAB. For this reason, this version was chosen to be further explored in order to gain more information and more specific optimization opportunities. The first optimization proposal is related to creating combinations of the input parameters of the control system ($\mathbf{K}_p$, $\mathbf{K}_d$, t_{delay}). According to the Figure 23(a), the first parameter combinations converge and are stable, i.e., in these cases the robot fully executes the simulation, accomplishing its mission until it waits for the simulation time to finish. The empty area in the figure indicates that all those gains comprised there did not satisfy the performance requirements for certain values of time delay. Since the thread "kills" the simulation when it finds that it has not met at least one of the performance parameters, most simulations run for a short time. Thus, it is expected that the most efficient approach possible with respect to creating all possible options for combining the parameters, will be the one where the delay time is the value that varies the least. In general terms, an entire block will execute horizontal planes (as the ones in red in Figure 23(a)) because they will

tend to release the block faster because as the time delay increases, the more unstable the system will become, so the simulations will last less and less time until they are discarded by the thread that is running them.

In this sense, the next evaluation is about the impact of the parameter allocation order on the input instance and whether it is possible to optimize the code execution through this analysis. The following combinations were tested: ($\mathbf{K}_d,\ \mathbf{K}_p,\ t_k$); ($\mathbf{K}_p,\ \mathbf{K}_d,\ t_k$); ($\mathbf{K}_d,\ t_k,\ \mathbf{K}_p$); ($t_k,\ \mathbf{K}_p,\ \mathbf{K}_d$); ($\mathbf{K}_p,\ t_k,\ \mathbf{K}_d$); ($t_k,\ \mathbf{K}_d,\ \mathbf{K}_p$). In this representation, the rightmost element of the set is the one that varies the least, and the leftmost element is the one that varies the most. In terms of constructing all combinations from three nested *for* loops, basically the rightmost term represents the outermost *for loop*, varying less frequently between consecutive samples. After reaching the deepest node of the nesting, one has the leftmost variable in the set, i.e., with the highest frequency of variation in a group of consecutive samples. In this experiment, the interest is the runtime, since no changes in the kernel code were made. That is, the same data for PTX code size, number of load stores and registers, including the number of floating point operations, are kept. The results for running on the Tesla K80 and T4 GPUs are shown below in Tables 5.

Table 5 – Code optimizations analysis by changing the parameters order in the combination set in the optimized `factMAT2CUDA` code version.

		($\mathbf{K}_d,\ \mathbf{K}_p,\ t_k$)				($\mathbf{K}_p,\ \mathbf{K}_d,\ t_k$)				($\mathbf{K}_d,\ t_k,\ \mathbf{K}_p$)			
	CPU	T4		K80		T4		K80		T4		K80	
	MATLAB	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total
Time [s]	33371	5.7	6.5	14.7	14.8	5.4	6.1	13.6	14.0	5.9	7.8	14.9	15.3
Speedup	1	5134		2254.8		5470.6		2383.7		4278.3		2181.1	
flow rate (Execution Time per Instance)													
Time [ms]	509.2	0.1		0.2		0.09		0.2		0.10		0.2	

		($t_k,\ \mathbf{K}_p,\ \mathbf{K}_d$)				($\mathbf{K}_p,\ t_k,\ \mathbf{K}_d$)				($t_k,\ \mathbf{K}_d,\ \mathbf{K}_p$)			
	CPU	T4		K80		T4		K80		T4		K80	
	MATLAB	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total
Time [s]	33371	6.1	7.1	14.7	15.1	5.7	6.2	14.6	15.1	5.7	6.4	14.7	15.2
Speedup	1	4700.1		2210		5382.4		2210		5214.2		2195.5	
flow rate (Execution Time per Instance)													
Time [ms]	509.2	0.09		0.2		0.09		0.2		0.09		0.2	

Again, a possible optimization is observed by changing the order of the parameters during the analysis. Although small, the performance increases especially in the version with ordering in the form ($\mathbf{K}_p,\ \mathbf{K}_d,\ t_k$). At this point, there was a gain of 300 ms compared to the other cases. Besides this, there were cases that deteriorated the performance of the system by changing the variation of the parameters. As commented earlier, this combination can vary from controller to controller, mainly because it will depend on the behavior of the volume generated from the analysis in the parameter space. However, for this application, the suggested change is sufficient. The speedup has obtained a gain of more than 5470× compared to MATLAB.

An additional proposal is presented in the previous sections, characterized by associating to a thread two simulation points. This approach has been successful used to optimize several applications (VOLKOV, 2016). That is, sequentially, each thread will run two simulations simultaneously, each for a unique combination of parameters. This approach allows the same amount of simulations as in the previous case, but halving the number of blocks or the number of threads per block in order to run the 65,536 simulations for comparison. Performance is expected to be improved, since both UAVs will share the simulations constants with each other, making better use of the registers. The results obtained in this step of the analysis are in Table 6. In this case, the execution time increased when running on K80, however, when performing the doubling of operations, the time varies slightly, showing that the GPU is not yet saturated and can perform many more operations until it gets close to saturation. It can also be seen that the increased use of registers is a crucial point for the latency of the code. The fewer registers are used by each thread, the more the GPU can perform, increasing its occupancy.

Table 6 – Code optimizations analysis for 3 cases implemented in GPU K80. Two points per thread. The number of simulations performed is equal to twice the number of threads. The implementation has 5340 PTX lines, 40 local stores, and 239 registers.

		128×256 threads		128×128 threads		64×64 threads	
	MATLAB CPU	Kern	Total	Kern	Total	Kern	Total
Time [s]	33371	10.78	10.98	10.65	10.70	3.83	3.88
flow rate [ms]	509.2	0.16		0.32		0.47	
Speedup	1	3039.25		-		-	
FlOps (SP)	-	1.5566e+10		1.5566e+10		3890839464	
FlOps (DP)	-	2652360206		2652362430		663364368	
GOps	-	1.8		1.7		1.3	

As a last analysis proposal, not only of optimization but also of extension of the proposed strategy, it was implemented a near hover controller that contains 12 control parameters, referring to the robot state (6 variables, 3 of position and 3 of orientation) and to its temporal derivatives. The evaluation of the implementation of this problem was done by gradually increasing the amount of changeable parameters, every two, in order to study scalability and GPU saturation. At first 3 parameters were varied (2 controller gains + time delay), and in each new test, 2 more parameters were added until reaching the 13 varying input parameters.

The different time execution presented in Tables 7, for each GPU architecture, is related to the effects of changing the control parameters in presence of time delay, i.e., in some cases, part of the simulations will be fully executed because the robot can achieve its goal in a stable and timely manner due to the set of control parameters for a given delay. The rest of the simulations will not fully execute due to the robot's behavior, which not obeyed the navigation performance requirements.

It is possible to note the scalability of the proposed methodology, and, mainly, the

Table 7 – Model scalability analysis using a more elaborated controller law with 12 parameters to tuning and the time delay in feedback data. All the versions execute 65536 simulations.

	3 parameters				5 parameters				7 parameters			
	T4		K80		T4		K80		T4		K80	
	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total
Time [s]	12.05	12.85	33.168	33.68	7.76	8.56	21.40	21.91	8.94	9.74	24.61	25.1
flow rate (Execution Time per Instance)												
Time [ms]	0.19		0.51		0.13		0.32		0.15		0.38	

	9 parameters				11 parameters				13 parameters			
	T4		K80		T4		K80		T4		K80	
	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total	Kern	Total
Time [s]	5.99	6.78	15.93	16.42	9.13	9.93	24.46	24.95	9.59	10.39	25.79	26.27
flow rate (Execution Time per Instance)												
Time [ms]	0.10		0.25		0.15		0.38		0.16		0.40	

performance of the code execution will depend on the number of simulations that will be run in full, that is, for combinations of gains and time delay that will lead the robot to converge to the destination point in a safe and stable way. Even with more complex models and with a larger number of parameters to be calibrated, the run time does not exceed 60 seconds.

3.5.3 Comparison with MATLAB/Simulink conversors and compilers for C/C++ & GPU

Finishing the study presented in this work, some implementation facilitates in MATLAB are analyzed and compared for enhancing the simulations in the software. The approach are divided into three main MATLAB tools to compare the built-in functionalities and the handmade code optimizations. The first MATLAB code reference is named as "`sequential_MATLAB`", the original code designed in the MATLAB software as a framework for robotics. The following approach is to use the *parfor* function for the generation of the all possible parameters combinations using parallel for loops in a multi-core CPU. The last implementation uses the MATLAB Coder toolbox to generate an automatic C/C++ algorithm from the original simulator. It is important to mention that this automatic script conversion using the MATLAB toolbox, generated code that needed to be adjusted manually. The best GPU handmade version proposed in this work is 5959 × faster than MATLAB native code, and two orders of magnitude faster than the multi-thread parallel for code and C code converter provided by the MATLAB toolbox.

Table 8 – Execution time comparison between MATLAB and Handmade CPU and GPU code. All times are in seconds.

		MATLAB		Handmade	
Code	Sequential	parallel for (16 threads)	C coder converter	Sequential	factMAT2CUDA
Original simulation with 3 parameters	33371 (1×)	1820 (18.3×)	3522 (9.5×)	82.1 (406.5×)	5.6 (5959.1×)
Original simulation with 5 parameters	15049 (1×)	855 (17.6×)	1766 (8.5×)	-	8.56 (1758.1×)
New controller with 13 parameters	55434 (1×)	3530 (15.7×)	6732 (8.2×)	-	10.39 (5335.3×)

3.6 Conclusion

Recently, in robotics applications that requires massive matrix calculation such as path planning, machine learning and vision, high performance computing strategies has been implemented to improve performance analysis and reduce the development time of new solutions. In this sense, this work aims to study the aerial robots behavior during its mission execution when adjusting the control system parameters off-line. 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. A GPU-accelerated flight route analysis for multi-UAV systems is proposed for the problem of control tuning in the parameters space considering the problem of delay in sending information to a ground control station (GCS). Considering MATLAB® implementations, experimental results show a speedup up to 325×, 629×, and 5959× in comparison to parallel version with 16 threads, c coder converter, and native Matlab code, respectively. The implement is available in Colab Google platform and can easily be expanded for analyses involving larger amounts of different parameters, robot models, strategies and controllers.

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. The estimation of the acceptable time-delay limit is performed from the analysis of performance indices.

When the application is parallelizable, the use of CUDA and C/C++ languages for GPU implementation can improve execution performance. The results obtained show that the impact on the execution time of the analysis is negligible when comparing a simple underactuated controller with 3 analyzed parameters to a more complex near hover controller with 13 parameters. Observation of the results shows that an analysis that takes about 9 hours to complete in MATLAB and another that would be computationally impossible to perform in a timely manner (due to the increased complexity of the sequential

algorithm), can be performed in less than a minute when implemented on GPU. The HPC strategy adopted is simple to extend to other types of robots and strategies, such as robot cooperation, human-robot interaction, among others. 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.

The performance gain shown offers new possibilities and perspectives for the scientific community. The experimental results show that the approach is able to generate code that provides significant speedups using a discrete GPU over equivalent sequential C code executing on a CPU. 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 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.

4 Concluding Remarks

Robotics simulators are present in most practical applications, because taking robots into the field can be costly and dangerous. Priority, time and cost are saved with simulations, since they can be performed at any time, all that is needed is for the researcher/student to have at his/her disposal a computer with the necessary tools to perform them. Aiming to fill the existing gap of a teaching tool containing documentation, narrative about models, description of controllers, and analysis of results, this work presented cloud4AuRoRA, a framework of the AuRoRA platform, used to perform simulations and experiments with mobile robots. In such a context, this thesis presented a practical application of the tool in research with over 60,000 simulations running in a few seconds, which is a metric almost impossible to achieve using AuRoRA, in MATLAB, with similar computational power. In summary, Cloud4AuRoRA is a versatile and iterative open-source tool that runs in the web browser for free without the need for the user to install additional extensions. Since it is hosted on Google Colab, cloud4AuRoRA enables parallelization of the simulation on powerful GPUs such as the T4 and P100.

In its first version, presented in this thesis, cloud4AuRoRA leaves gaps for many improvements discussed in the text. As suggestions for future work, one can mention the implementation of other robots used in the NERo, making the platform broader and facilitating its use for testing and implementing robot cooperation strategies. It can also be used more frequently in undergraduate and graduate courses, where AuRoRA platform is already used. Another branch of research and development is the inclusion of a section of Colab to help those who wish to perform parallel GPU simulations, as is done in Chapter 3, spreading this knowledge of the application of high performance computing strategies to robotics problems. This strand of work can be easily extended to applications involving machine learning, enabling the researcher/student to train and obtain complex models that can be directly applied to real robots for autonomous navigation. Finally, the implementation of Gradio to use could4AuRoRA in teaching also follows as a future prospect.

Bibliography

AL-MOUSA, A. et al. Utsim: A framework and simulator for uav air traffic integration, control, and communication. *International Journal of Advanced Robotic Systems*, SAGE Publications Sage UK: London, England, v. 16, n. 5, p. 1729881419870937, 2019. 34

ALJEHANI, M.; INOUE, M. Performance evaluation of multi-uav system in post-disaster application: Validated by hitl simulator. *IEEE Access*, IEEE, v. 7, p. 64386–64400, 2019. 34

BACHETI, V. P.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. Path-following with a ugv-uav formation considering that the uav lands on the ugv. In: IEEE. *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. [S.l.], 2020. p. 488–497. 51

BEG, M. et al. Using jupyter for reproducible scientific workflows. *Computing in Science & Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 23, n. 2, p. 36–46, 2021. 37, 38, 60

BERNHARD, J.-C. et al. Nurse-led coordinated surgical care pathways for cost optimization of robotic-assisted partial nephrectomy: medico-economic analysis of the uroccr-25 ambu-rein study. *World Journal of Urology*, Springer, p. 1–9, 2022. 14

BIEZE, T. M. et al. Design, implementation, and control of a deformable manipulator robot based on a compliant spine. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 39, n. 14, p. 1604–1619, 2020. 16

BIGURI, A. et al. TIGRE: a MATLAB-GPU toolbox for CBCT image reconstruction. *Biomedical Physics & Engineering Express*, IOP Publishing, v. 2, n. 5, p. 055010, set. 2016. Disponível em: https://doi.org/10.1088/2057-1976/2/5/055010. 68

BISONG, E. *Building machine learning and deep learning models on Google cloud platform: A comprehensive guide for beginners*. [S.l.]: Apress, 2019. 40

BRANDÃO, A. S.; FILHO, M. S.; CARELLI, R. High-level underactuated nonlinear control for rotorcraft machines. In: IEEE. *2013 IEEE International Conference on Mechatronics (ICM)*. [S.l.], 2013. p. 279–285. 25, 26, 27, 28, 62, 63, 65, 75

BRANDÃO, A. S. et al. Side-pull maneuver: A novel control strategy for dragging a cable-tethered load of unknown weight using a uav. *IEEE Robotics and Automation Letters*, IEEE, v. 7, n. 4, p. 9159–9166, 2022. 27, 28

BRANTNER, G.; KHATIB, O. Controlling ocean one. In: SPRINGER. *Field and Service Robotics*. [S.l.], 2018. p. 3–17. 15

BRANTNER, G.; KHATIB, O. Controlling ocean one: Human–robot collaboration for deep-sea manipulation. *Journal of Field Robotics*, Wiley Online Library, v. 38, n. 1, p. 28–51, 2021. 15, 19

CANESCHE, M. et al. Google colab cad4u: Hands-on cloud laboratories for digital design. In: IEEE. *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. [S.l.], 2021. p. 1–5. 59

CASAS, V.; MITSCHELE-THIEL, A. On the impact of communication delays on uavs flocking behavior. In: IEEE. *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. [S.l.], 2018. p. 67–72. 55

CASTILLO-PIZARRO, P.; ARREDONDO, T. V.; TORRES-TORRITI, M. Introductory survey to open-source mobile robot simulation software. In: IEEE. *2010 Latin American Robotics Symposium and Intelligent Robotics Meeting*. [S.l.], 2010. p. 150–155. 22

CAVALCANTI, A. et al. Verified simulation for robotics. *Science of Computer Programming*, Elsevier, v. 174, p. 1–37, 2019. 20

CHOI, H. et al. On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 118, n. 1, p. e1907856118, 2021. 20

CHOI, Y. et al. Service robots in hotels: understanding the service quality perceptions of human-robot interaction. *Journal of Hospitality Marketing & Management*, Taylor & Francis, v. 29, n. 6, p. 613–635, 2020. 14

COELHO, A. F. *System Identification and Parameter Space Control Design for a Small Unmanned Aircraft*. Tese (Doutorado) — Instituto Federal Fluminense (IFF), Campos, Brazil, German Aerospace Center - DLR, 2017. 60, 67

COLLINS, J. et al. A review of physics simulators for robotic applications. *IEEE Access*, IEEE, v. 9, p. 51416–51431, 2021. 22

CRAIGHEAD, J. et al. A survey of commercial & open source unmanned vehicle simulators. In: IEEE. *Proceedings 2007 IEEE International Conference on Robotics and Automation*. [S.l.], 2007. p. 852–857. 22

DELAVARPOUR, N. et al. A technical study on uav characteristics for precision agriculture applications and associated practical challenges. *Remote Sensing*, MDPI, v. 13, n. 6, p. 1204, 2021. 16

DREW, D. S. Multi-agent systems for search and rescue applications. *Current Robotics Reports*, Springer, v. 2, n. 2, p. 189–200, 2021. 14

EBEID, E. et al. A survey of open-source uav flight controllers and flight simulators. *Microprocessors and Microsystems*, Elsevier, v. 61, p. 11–20, 2018. 34

FAGUNDES-JUNIOR, L. et al. A nonlinear uav control tuning under communication delay using hpc strategies in parameters space. In: SBC. *Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho*. [S.l.], 2021. p. 228–239. 29, 30, 31, 56, 60, 77

FAGUNDES-JÚNIOR, L. A. et al. Navegação baseada em sensoriamento laser para uma formação líder-seguidor. In: *Simpósio Brasileiro de Automação Inteligente-SBAI*. [S.l.: s.n.], 2021. v. 1, n. 1. 30

FAGUNDES, L. A.; SOUZA, V. B.; BRANDÃO, A. S. On the evaluation of access-point handovers for uavs in long-distance missions. In: IEEE. *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. [S.l.], 2021. p. 1520–1529. 30

FREITAS, D. C. H. de et al. Navegação de robôs móveis em corredores usando sensoriamento laser. In: IEEE. *2021 14th IEEE International Conference on Industry Applications (INDUSCON)*. [S.l.], 2021. p. 842–849. 30

Google LLC. *Welcome to Google Colaboratory! (Colab)*. 2010. https://colab.research.google.com/. Accessed: 2022-02-28. 60

GUYOT, L. et al. Teaching robotics with an open curriculum based on the e-puck robot, simulations and competitions. In: *Proceedings of the 2nd International Conference on Robotics in Education. Vienna, Austria*. [S.l.: s.n.], 2011. 20

HAGN, U. et al. Dlr mirosurge: a versatile system for research in endoscopic telesurgery. *International journal of computer assisted radiology and surgery*, Springer, v. 5, n. 2, p. 183–193, 2010. 15

HAMBUCHEN, K.; MARQUEZ, J.; FONG, T. A review of nasa human-robot interaction in space. *Current Robotics Reports*, Springer, v. 2, n. 3, p. 265–272, 2021. 14

HASSANI, H.; MANSOURI, A.; AHAITOUF, A. Control system of a quadrotor uav with an optimized backstepping controller. In: IEEE. *2019 international conference on intelligent systems and advanced computing sciences (ISACS)*. [S.l.], 2019. p. 1–7. 34

HE, W. et al. Admittance-based controller design for physical human–robot interaction in the constrained task space. *IEEE Transactions on Automation Science and Engineering*, IEEE, v. 17, n. 4, p. 1937–1949, 2020. 14

HENSCHEL, A.; HORTENSIUS, R.; CROSS, E. S. Social cognition in the age of human–robot interaction. *Trends in Neurosciences*, Elsevier, v. 43, n. 6, p. 373–384, 2020. 14

HENTATI, A. I. et al. Simulation tools, environments and frameworks for uav systems performance analysis. In: IEEE. *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. [S.l.], 2018. p. 1495–1500. 34

HSU, K.-C.; TSENG, H.-W. Accelerating applications using edge tensor processing units. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. [S.l.: s.n.], 2021. p. 1–14. 57

HULIN, T. et al. Model-augmented haptic telemanipulation: Concept, retrospective overview, and current use cases. *Frontiers in Robotics and AI*, Frontiers Media SA, v. 8, p. 611251, 2021. 15, 18

JOHNSTON, D. W. Unoccupied aircraft systems in marine science and conservation. *Annual review of marine science*, Annual Reviews, v. 11, p. 439–463, 2019. 14

KALITA, H.; THANGAVELAUTHAM, J. Exploration of extreme environments with currentand emerging robot systems. *Current Robotics Reports*, Springer, v. 1, n. 3, p. 97–104, 2020. 14

KANG, D. et al. Toonnote: Improving communication in computational notebooks using interactive data comics. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2021. p. 1–14. 37

KAPETANOVIĆ, N. et al. Marine robots mapping the present and the past: Unraveling the secrets of the deep. *Remote Sensing*, MDPI, v. 12, n. 23, p. 3902, 2020. 14

KERY, M. B. et al. The story in the notebook: Exploratory data science using a literate programming tool. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2018. p. 1–11. 37

KHATIB, O. et al. Ocean one: A robotic avatar for oceanic discovery. *IEEE Robotics & Automation Magazine*, IEEE, v. 23, n. 4, p. 20–29, 2016. 15, 19

KIM, J. et al. Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. *Ieee Access*, IEEE, v. 7, p. 105100–105115, 2019. 14

KOCER, B. B. et al. Forest drones for environmental sensing and nature conservation. In: IEEE. *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)*. [S.l.], 2021. p. 1–8. 14

KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IEEE. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. [S.l.], 2004. v. 3, p. 2149–2154. 22, 23

KONIETSCHKE, R. et al. The dlr mirosurge-a robotic system for surgery. In: IEEE. *2009 IEEE international conference on Robotics and automation*. [S.l.], 2009. p. 1589–1590. 15

LEIDNER, D. et al. Object-centered hybrid reasoning for whole-body mobile manipulation. In: IEEE. *2014 IEEE International conference on robotics and automation (ICRA)*. [S.l.], 2014. p. 1828–1835. 15, 18

LEISERSON, C. et al. There's plenty of room at the top: What will drive growth in computer performance after moore's law ends. *Unpublished manuscript*, 2016. 58

LIANG, J.; AL et. Gpu-accelerated robotic simulation for distributed reinforcement learning. In: PMLR. *Conference on Robot Learning*. [S.l.], 2018. p. 270–282. 61

LIANG, J. et al. In-hand object pose tracking via contact feedback and gpu-accelerated robotic simulation. In: IEEE. *2020 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2020. p. 6203–6209. 61

LOPEZ, I. B. et al. Robotics in spine surgery: systematic review of literature. *International Orthopaedics*, Springer, p. 1–10, 2022. 14

LORETO-GÓMEZ, G. et al. Analysing the effect of the use of 3d simulations on the performance of engineering students in a robotics course: findings from a pilot study. *The International Journal of Electrical Engineering & Education*, SAGE Publications Sage UK: London, England, v. 56, n. 2, p. 163–178, 2019. 16

LV, F.; HE, W.; ZHAO, L. A multivariate optimal control strategy for the attitude tracking of a parafoil-UAV system. *IEEE Access*, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 43736–43751, 2020. Disponível em: https://doi.org/10.1109/access.2020.2977535. 28

LYTRIDIS, C. et al. An overview of cooperative robotics in agriculture. *Agronomy*, MDPI, v. 11, n. 9, p. 1818, 2021. 14

MA, F. et al. Parameter-space-based robust control of event-triggered heterogeneous platoon. *IET Intelligent Transport Systems*, Institution of Engineering and Technology (IET), v. 15, n. 1, p. 61–73, nov. 2020. Disponível em: https://doi.org/10.1049/itr2.12004. 67

MARTINEZ-GONZALEZ, P. et al. Unrealrox: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation. *Virtual Reality*, Springer, v. 24, n. 2, p. 271–288, 2020. 20

MILLAN, B. et al. A scoping review of emerging and established surgical robotic platforms with applications in urologic surgery. *Société Internationale d'Urologie Journal*, v. 2, n. 5, p. 300–310, 2021. 14

NAGAR, S. IPython. In: *Introduction to Python for Engineers and Scientists*. [S.l.]: Apress, 2017. p. 31–45. 36

NETO, V. E.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. Load manipulation by a triangular formation of quadrotors. In: IEEE. *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. [S.l.], 2020. p. 744–753. 51

NETO, V. E.; SARCINELLI-FILHO, M.; BRANDÃO, A. S. Trajectory-tracking of a heterogeneous formation using null space-based control. In: IEEE. *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. [S.l.], 2019. p. 187–195. 67

ONO, J. P.; FREIRE, J.; SILVA, C. T. Interactive data visualization in jupyter notebooks. *Computing in Science & Engineering*, IEEE, v. 23, n. 2, p. 99–106, 2021. 37

OROZCO-ROSAS, U. et al. GPU accelerated membrane evolutionary artificial potential field for mobile robot path planning. In: *Fuzzy Logic Hybrid Extensions of Neural and Optimization Algorithms: Theory and Applications*. Springer International Publishing, 2021. p. 233–247. Disponível em: https://doi.org/10.1007/978-3-030-68776-2_13. 61

PÉREZ, F.; GRANGER, B. E. Ipython: a system for interactive scientific computing. *Computing in science & engineering*, IEEE, v. 9, n. 3, p. 21–29, 2007. 35, 36, 40

PHANDEN, R. K.; SHARMA, P.; DUBEY, A. A review on simulation in digital twin for aerospace, manufacturing and robotics. *Materials Today: Proceedings*, Elsevier BV, v. 38, p. 174–178, 2021. Disponível em: https://doi.org/10.1016/j.matpr.2020.06.446. 60

PIMENTEL, J. F. et al. A large-scale study about quality and reproducibility of jupyter notebooks. In: IEEE. *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*. [S.l.], 2019. p. 507–517. 37

PISKORSKI, S.; BRULEZ, N. Ar. drone developer guide parrot. sdk version 2.0. *tech. rep.*, 2012. 62

PIZETTA, I. H.; BRANDAO, A. S.; FILHO, M. S. Sistema embarcado para controle de vants de pás rotativas: Parte de alto nível da plataforna aurora. In: *Anais do XX Congresso Brasileiro de Automática (CBA)*. Belo Horizonte: SBA, 2014. Disponível em: http://www.swge.inf.br/cba2014/anais/PDF/1569934655.pdf. 21, 23, 24

PIZETTA, I. H. B.; BRANDAO, A. S.; SARCINELLI-FILHO, M. A hardware-in-loop platform for rotary-wing unmanned aerial vehicles. In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. [S.l.]: IEEE, 2014. 21

PIZETTA, I. H. B.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. Modelling and control of a quadrotor carrying a suspended load. In: IEEE. *2015 workshop on research, education and development of unmanned aerial systems (RED-UAS)*. [S.l.], 2015. p. 249–257. 25

PIZETTA, I. H. B.; BRANDAO, A. S.; SARCINELLI-FILHO, M. A hardware-in-the-loop platform for rotary-wing unmanned aerial vehicles. *Journal of Intelligent & Robotic Systems*, Springer, v. 84, n. 1, p. 725–743, 2016. 21, 23, 24

PIZETTA, I. H. B.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. Avoiding obstacles in cooperative load transportation. *ISA transactions*, Elsevier, v. 91, p. 253–261, 2019. 51

PLASENCIA, A. et al. Open source robotic simulators platforms for teaching deep reinforcement learning algorithms. *Procedia Computer Science*, Elsevier, v. 150, p. 162–170, 2019. 23

POPOV, A. V.; SAYARKIN, K. S.; ZHILENKOV, A. A. The scalable spiking neural network automatic generation in matlab focused on the hardware implementation. In: IEEE. *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. [S.l.], 2018. p. 962–965. 58

QUAGLIA, G. et al. Design of a ugv powered by solar energy for precision agriculture. *Robotics*, MDPI, v. 9, n. 1, p. 13, 2020. 14

QUERALTA, J. P. et al. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *Ieee Access*, IEEE, v. 8, p. 191617–191643, 2020. 14

RABELO, M. F. S.; BRANDAO, A. S.; SARCINELLI-FILHO, M. 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). *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. [S.l.]: IEEE, 2018. 29

REIS, L.; BISPO, J.; CARDOSO, J. M. Compilation of matlab computations to cpu/gpu via c/opencl generation. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 32, n. 22, p. e5854, 2020. 58

ROSS, R. et al. Wombot: An exploratory robot for monitoring wombat burrows. *SN Applied Sciences*, Springer, v. 3, n. 6, p. 1–10, 2021. 14

ROSSANT, C. *Learning IPython for interactive computing and data visualization*. [S.l.]: Packt Publishing Birmingham, UK, 2013. 36, 37, 40

RULE, A. et al. Ten simple rules for writing and sharing computational analyses in jupyter notebooks. *PLOS Computational Biology*, Public Library of Science, v. 15, n. 7, p. 1–8, 07 2019. Disponível em: https://doi.org/10.1371/journal.pcbi.1007007. 35, 40, 59

RULE, A.; TABARD, A.; HOLLAN, J. D. Exploration and explanation in computational notebooks. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2018. p. 1–12. 37, 59

RULE, A.; TABARD, A.; HOLLAN, J. D. Exploration and explanation in computational notebooks. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2018. p. 1–12. 37, 40

SANTANA, L. V.; BRANDAO, A. S.; SARCINELLI-FILHO, M. Outdoor waypoint navigation with the AR.drone quadrotor. In: 2015 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS). *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2015. Disponível em: https://doi.org/10.1109/icuas.2015.7152304. 62

SANTANA, L. V.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. Navigation and cooperative control using the AR.drone quadrotor. *Journal of Intelligent & Robotic Systems*, Springer Science and Business Media LLC, v. 84, n. 1-4, p. 327–350, fev. 2016. 29

SCHMAUS, P. et al. Preliminary insights from the meteron supvis justin space-robotics experiment. *IEEE Robotics and Automation Letters*, IEEE, v. 3, n. 4, p. 3836–3843, 2018. 14, 15

SCHMAUS, P. et al. Knowledge driven orbit-to-ground teleoperation of a robot coworker. *IEEE Robotics and Automation Letters*, IEEE, v. 5, n. 1, p. 143–150, 2019. 15, 18

SEIBOLD, U. et al. The dlr mirosurge surgical robotic demonstrator. In: *The Encyclopedia of MEDICAL ROBOTICS: Volume 1 Minimally Invasive Surgical Robotics*. [S.l.]: World Scientific, 2019. p. 111–142. 14

SEIFI, A. et al. Modeling and uncertainty analysis of groundwater level using six evolutionary optimization algorithms hybridized with ANFIS, SVM, and ANN. *Sustainability*, MDPI AG, v. 12, n. 10, p. 4023, maio 2020. Disponível em: https://doi.org/10.3390/su12104023. 60

SERRANO-PÉREZ, O. et al. Offline robust tuning of the motion control for omnidirectional mobile robots. *Applied Soft Computing*, Elsevier, v. 110, p. 107648, 2021. 55

SHABANINEZHAD, M.; AWAN, M.; RAMAKRISHNA, G. Matlab package for discrete dipole approximation by graphics processing unit: Fast fourier transform and biconjugate gradient. *Journal of Quantitative Spectroscopy and Radiative Transfer*, Elsevier, v. 262, p. 107501, 2021. 59

SHAFIEKHANI, A.; FRITSCHI, F. B.; DESOUZA, G. N. Vinobot and vinoculer: from real to simulated platforms. In: SPIE. *Autonomous Air and Ground Sensing Systems for Agricultural Optimization and Phenotyping III*. [S.l.], 2018. v. 10664, p. 90–98. 17

SHAFIEKHANI, A. et al. Vinobot and vinoculer: Two robotic platforms for high-throughput field phenotyping. *Sensors*, MDPI, v. 17, n. 1, p. 214, 2017. 17

SHARIAR, S.; HASAN, K. A. Gpu accelerated indexing for high order tensors in google colab. In: IEEE. *2020 IEEE Region 10 Symposium (TENSYMP)*. [S.l.], 2020. p. 686–689. 60

SOLEIMANI, H. Hardware realisation of nonlinear dynamical systems for and from biology. *arXiv preprint arXiv:2108.04928*, 2021. 57

STARANOWICZ, A.; MARIOTTINI, G. L. A survey and comparison of commercial and open-source robotic simulator software. In: *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. [S.l.: s.n.], 2011. p. 1–8. 22

ŠTEFEK, A. et al. Optimization of fuzzy logic controller used for a differential drive wheeled mobile robot. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 11, n. 13, p. 6023, 2021. 60

STONE, J. E. et al. Accelerating molecular modeling applications with graphics processors. *Journal of computational chemistry*, Wiley Online Library, v. 28, n. 16, p. 2618–2640, 2007. 57

STRIPINIS, L. et al. On matlab experience in accelerating direct-glce algorithm for constrained global optimization through dynamic data structures and parallelization. *Applied Mathematics and Computation*, Elsevier, v. 390, p. 125596, 2021. 59

SUH, J.; KIM, Y. *Accelerating MATLAB with GPU computing: a primer with examples*. [S.l.]: Newnes, 2013. 58, 59

TAKAOĞLU, F. et al. Robust nonlinear non-referenced inertial frame multi-stage pid controller for symmetrical structured uav. *Symmetry*, MDPI, v. 14, n. 4, p. 689, 2022. 34

TANG, Y. et al. Leader-following consensus control for multiple fixed-wing uavs' attitude system with time delays and external disturbances. *IEEE Access*, IEEE, v. 7, p. 169773–169781, 2019. 55

TANG, Y.-R.; XIAO, X.; LI, Y. Nonlinear dynamic modeling and hybrid control design with dynamic compensator for a small-scale UAV quadrotor. *Measurement*, Elsevier BV, v. 109, p. 51–64, 2017. 27, 62

TOBERGTE, A. et al. The sigma. 7 haptic interface for mirosurge: A new bi-manual surgical console. In: IEEE. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.], 2011. p. 3023–3030. 15

TSELEGKARIDIS, S.; SAPOUNIDIS, T. Simulators in educational robotics: A review. *Education Sciences*, MDPI, v. 11, n. 1, p. 11, 2021. 23

TURKER, T.; YILMAZ, G.; SAHINGOZ, O. K. GPU-accelerated flight route planning for multi-UAV systems using simulated annealing. In: *Artificial Intelligence: Methodology, Systems, and Applications*. Springer International Publishing, 2016. p. 279–288. Disponível em: https://doi.org/10.1007/978-3-319-44748-3_27. 61

VALLEJO, W.; DÍAZ-URIBE, C.; FAJARDO, C. Google colab and virtual simulations: Practical e-learning tools to support the teaching of thermodynamics and to introduce coding to students. *ACS Omega*, ACS Publications, 2022. 59

VASCONCELOS, J. V. R. d. (re) planejamento de rotas em tempo real para missões estratégicas. Universidade Federal de Viçosa, 2021. 29

VOLKOV, V. *Understanding latency hiding on GPUs*. [S.l.]: University of California, Berkeley, 2016. 80

WADA, D.; ARAUJO-ESTRADA, S. A.; WINDSOR, S. Unmanned aerial vehicle pitch control using deep reinforcement learning with discrete actions in wind tunnel test. *Aerospace*, Multidisciplinary Digital Publishing Institute, v. 8, n. 1, p. 18, 2021. 55

WEN, Y. et al. Online reinforcement learning control for the personalization of a robotic knee prosthesis. *IEEE transactions on cybernetics*, IEEE, v. 50, n. 6, p. 2346–2356, 2019. 55

XU, Y.; VATANKHAH, H. Simspark: An open source robot simulator developed by the robocup community. In: SPRINGER. *Robot Soccer World Cup*. [S.l.], 2013. p. 632–639. 23

YUAN, K.; CHATZINIKOLAIDIS, I.; LI, Z. Bayesian optimization for whole-body control of high-degree-of-freedom robots through reduction of dimensionality. *IEEE Robotics and Automation Letters*, IEEE, v. 4, n. 3, p. 2268–2275, 2019. 55

ZAGAL, J. C.; RUIZ-DEL-SOLAR, J. Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*, Springer, v. 50, n. 1, p. 19–39, 2007. 20

ZHANG, B. et al. Accelerating MatLab code using GPU: A review of tools and strategies. In: 2ND INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, MANAGEMENT SCIENCE AND ELECTRONIC COMMERCE (AIMSEC). *2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*. IEEE, 2011. Disponível em: https://doi.org/10.1109/aimsec.2011.6010978. 68

ZHU, S.; AL et. 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). *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018. Disponível em: https://doi.org/10.1109/cdc.2018.8619749. 67

ŽLAJPAH, L. Simulation in robotics. *Mathematics and Computers in Simulation*, Elsevier, v. 79, n. 4, p. 879–897, 2008. 20