Framework and evaluation methodology for Autonomous Drone Racing

Miguel Fernandez-Cortizas, Pablo Santamaria, David Perez-Saura, Javier Rodriguez-Vazquez, Martin Molina, Pascual Campoy

Computer Vision and Aerial Robotics group (CVAR), Centre for Automation and Robotics (CAR), Universidad Politécnica de Madrid (UPM-CSIC), Calle José Gutiérrez Abascal 2, 28006 Madrid, Spain.

ABSTRACT

In recent years, autonomous drone races have become increasingly popular in the aerial robotics research community, due to the challenges in perception, localization, navigation, and control at high speeds, pushing forward the state of the art every year. However, autonomous racing drones are still far from reaching human pilot performance and a lot of research has to be done to accomplish that. In this work, a complete architecture system and an evaluation method for autonomous drone racing research, based on the open source framework Aerostack 4.0, are proposed. In order to evaluate the performance of the whole system and of each algorithm used separately, this framework is validated not only with simulated flights, but also through real flights in an indoor drone race circuit by using different configurations.

1 INTRODUCTION

1.1 Motivation

Autonomous drones have been increasing their application in different tasks in recent years. The short flight time of a quadcopter limits its use in missions such as search and rescue. To take advantage of this, it is interesting to develop agile drones that can explore or traverse an area in a short time. Autonomous drone racing is a great environment to push agile drones to the limit. The high speeds and agile maneuvers required for this purpose increase the difficulties of locating, controlling, and generating trajectories. To test different techniques to get the best results, it is useful to work in a modulated environment that allows you develop and validate different algorithms independently.

1.2 Related Work

Over the years, many different techniques have been developed to get the best results in Autonomous Drone Racing. At the beginning, the speeds achieved in competition were considerably low. In ADR 2017, a combination of monocular SLAM localization algorithm and a PID control won with a mean speed of 0.7 m/s [1]. For IROS 2018 ADR, the winners used machine learning techniques for gate detection with an MPC controller. They also improved the trajectories adding 2 points to the path, one before and one after each gate, getting flight speeds near 2 m/s [2].

In 2019, there was a big improvement in how fast autonomous drones could fly. For the first AlphaPilot competition, a novel architecture was developed [3]. In this architecture, machine learning techniques are combined with a nonlinear filter for sensor detection and time-optimal trajectory planning. Using a PD controller, they reached a maximum flight speed of 8 m/s. In the same year, the winners of the first simulated drone racing, NeurIPS 2019 Game of Drones, developed a controller using reinforcement learning techniques [4], achieving a maximum speed of 16 m/s in simulation.

On the other hand, there are many simulation environments to test the different approaches. Moreover, many competitions have been hosted in this simulators. Flightgoggles [5] is a photo-realistic simulator used for AlphaPilot 2019. Some of them have some APIs for a specific development. Flightmare [6] has an API for reinforcement learning. AirSim Drone Racing Lab [7] is a framework from Microsoft with some APIs that allows you to focus your test on each of the different research directions in autonomous drone racing. However, there are not many frameworks that combine state-of-the-art algorithms with simulators to work as baseline repositories for autonomous drone racing research.

1.3 Contribution

In this paper, we present a modular framework for developing and validating new algorithms for improving agile drone flying using autonomous drone races as a perfect test environment. This framework provides a modular system architecture with some state-of-the-Art algorithms that allows researchers to concentrate efforts on improving one of the fields related to drone behavior, without needing to build a whole system by themselves. Furthermore, a simulation environment based on gazebo is provided to test the algorithms before jumping into real experiments. For real experiments, we decided to use Pixhawk as the autopilot to ease the use of this framework through the research community.

Finally, we propose a set of metrics for measuring the performance of some modules separately and the performance of the whole system to be able to compare different algorithms easily.

2 SYSTEM ARCHITECTURE

The system architecture that is presented in this work has been developed using the Aerostack framework [8], an open-source multi-purpose software framework for the development of autonomous multi-robot unmanned aerial systems created by the Computer Vision and Aerial Robotics (CVAR) group. The Aerostack modules are mainly implemented in C++ and Python languages and are based on Robot Operating System (ROS) [9] for inter-communication between the different components, we refer the reader to the extensive documentation and publications that are available on its web site [1].

Using the Aerostack software framework, a new system architecture design was developed for the tasks presented in this work. It has to be noted that, although Aerostack framework is able to provide predefined components and intercommunication methods to provide autonomy to UAVs, the components that are described in this work were completely designed and developed for the objectives described in this paper. Fig. 1 shows the functionalities that have been implemented in this work. In the figure, colored rectangular boxes represent data processing units (or processes in short) that are implemented as ROS nodes. They are organized in the following main components:

- *Sensor-Actuator Interfaces:* to receive data from sensors on the aerial platform and send commands to robot actuators.
- *Communication Channel:* Based on the Aerostack framework, our architecture uses a common communication channel that contains shared dynamic information between processes. This channel facilitates process interoperability and helps to reuse components across different types of aerial platforms. The channel is implemented with a set of ROS topics and ROS messages.
- *Robot Behaviors:* Robot behaviors implement the robot functional capabilities including motion control, feature extraction, state estimation, and navigation.
- *Mission Control:* Mission control executes a mission plan specified in a formal description. In this implementation, the mission plan is specified using the Python language using a set of prefixed functions to start and stop robot tasks. A behavior coordinator component [10] is used to translate planned tasks into consistent activations of robot behaviors.

This work utilises the behaviorlib library for programming robot behaviors with execution management functions [10]. This library is open-source and provides tools for building, executing, and monitoring behaviors, as it is influenced by the behavior-based paradigm in robotics. According to this paradigm, the global control is divided into a set of behavior controllers and each one is in charge of a specific control aspect separately from the other behavior controllers.

Next sections describe in more detail the components related to robot behaviors and mission control.

3 ROBOT BEHAVIOURS

A behavior defines a basic functionality of a system, such as moving to a point, moving an actuator, activating a sensor, and includes the three following principles:

- *Common data channel:* Each behavior controller should be able to execute separately assuming that the required input data is available in the common data channel.
- *Activation management:* Each behavior is programmed with an activation management mechanism which handles how to start and stop the execution of the behavior controller.
- *Execution monitoring:* Execution monitoring is a kind of self-awareness computing process by which the robot observes and judges its own behavior. This includes possible behavior termination states such as ”GOAL ACHIEVED”, ”WRONG PROGRESS” or ”PROCESS FAILURE”.

Aerostack provides behaviors which can be divided in the categories explained in the following subsections.

3.1 Motion control

All the set of behaviors that are responsible for controlling the movement of the drone. This category includes simple behaviors such as take-off, hovering or landing, as well as more complex behaviors such as generating a trajectory and making the drone follow it.

In order to realize aggressive maneuvers on a quadrotor, it is necessary to employ a non-linearized controller. We have implemented a quadrotor control algorithm based on differential flatness and the corresponding behaviors inspired by the work made by Melligner et al. [11] with several modifications, so the output of the controller corresponds to angular velocity references and the desired collective thrust of all motors. The input of this controller consists in the position, speed and acceleration references provided by a trajectory generator.

Due to the need of generating trajectories for the controller, a polynomial trajectory generator [12][13][11] has been used. The trajectories are generated on a set of waypoints and are optimal in acceleration, which guarantees smoothness in the actuator commands. Moreover, the trajectories generated are constrained by maximum speed v_{max} and maximum acceleration a_{max} parameters.

[1] `www.aerostack.org`

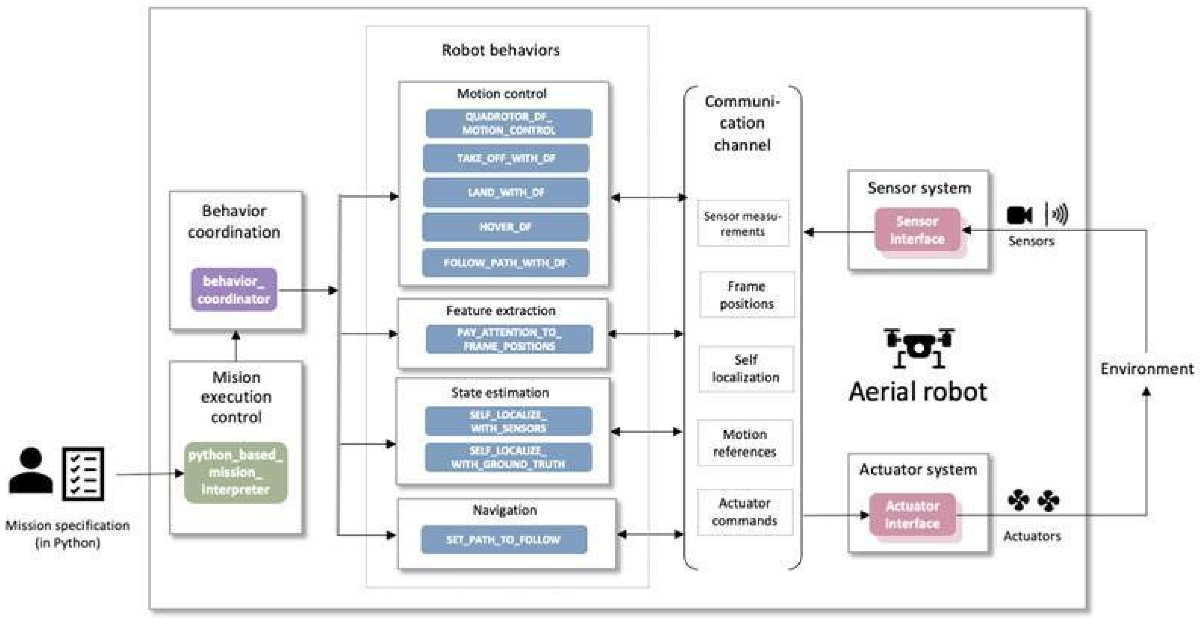

Figure 1: System architecture

3.2 State estimation

In order to achieve the level of control required for this project, it is necessary to provide the best state estimation possible to all components involved. Aerostack provides multiple components for this task, obtaining data from multiple sensors such as cameras (for relative positioning to a certain object), IMU, laser sensors, depth cameras, or lidar.

Some of this sensors or estimators should not be always trusted, as cumulative errors can and will happen, so it is necessary to combine the feedback from several of them using sensor fusion approaches, like the multi-sensor-fusion algorithm developed by Linnen et al. [14] , to achieve the most accurate and reliable state estimation possible in every situation.

However, for the real flights, we decided to use the state estimation provided by an Intel Realsense T265 Tracking Module, due to the ease of use and the reduction of the computational load of the on-board computer.

3.3 Perception

Perception is a key problem in the proposed scenario, since UAVs need to detect and locate each of the gates accurately to be able to complete the full track.

Since the detection and pose estimation of the gates is a difficult problem itself, to evaluate the rest of the proposed framework components without the influence of possible errors in perception, we placed ArUco fiducial markers [15] in each gate to estimate the relative position to the UAV's main camera. Each marker also encodes a unique gate ID, enabling us to design arbitrarily complex tracks. For the detection of such markers, we use OpenCV [16] implementation of the method proposed in [17]

3.4 Navigation

When the gates are located in the real world, it is necessary to plan the path that the aircraft must follow to pass through them and avoid other obstacles. For this approach we considered two scenarios:

- **Lack of knowledge of gate positions.** The aircraft does not have any information about where the gates are located, so it has to begin looking for them in order to generate the waypoints to pass through the gates. As long as the aircraft pass through the circuit, new gates will be in sight, so the quadrotor can add these gates to its route. In this scenario, the aircraft is always considered to see at least the following gate.

- **Gate position awareness.** The aircraft knows an approximated position of each gate of the circuit, so it can generate complete trajectories through all the circuit that must be corrected with as long as the gates are in sight, so it has to update the initial gate positions to pass through the circuit. This is how the majority of the autonomous drone racing competitions works.

In both approaches, each gate center is treated like a waypoint in a path, and this waypoint position is updated as the quadrotor flies through the circuit. Whenever the quadrotor passes through one gate this gate center is removed from the path. This path is sended to the trajectory generator, which takes charge of commanding the quadrotor to pass through the gates.

4 Experiments

To validate the system architecture proposed and the suitability of the different algorithms selected, several experiments have been realized, not only in simulation, but also in real. For analyzing this performance, several metrics have been used:

- **State Estimation error:** For measuring the accuracy of the state estimation algorithm, we compute the RMSE (Root Mean Squared Error) between the estimated state and the ground truth.
- **Trajectory following error:** To evaluate the performance of the controller proposed, we decide to measure the trajectory following error, calculated with the RMSE between the trajectory sent by the trajectory generator and the real trajectory followed by the aircraft.
- **Speeds:** In autonomous racing, other metrics like the maximum speed reached, the medium speed, or the elapsed time to go through all the circuit must be taken into consideration.

All metrics obtained from the different experiments were obtained automatically using an evaluation script on the raw data recorded during the flights.

4.1 Simulated Flights

Initially, the system was validated in simulation using Gazebo [18] simulator and the *iris* drone as a simulated quadrotor. We generate a 5 gates circuit distributed along a 25 x 20 x 3.5 m area, see Fig. 2. The position of each gate was known with an uncertainty of 3 meters, which forces the system to recalculate the gates positions to avoid crashing into them.

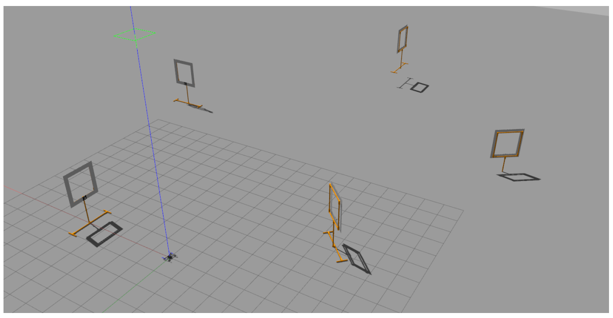
Figure 2: Gazebo environment used for simulated flight experiments

We fly through the circuit multiple times with four different speed configurations during each flight. In these experiments, the state of the aircraft was provided by the simulator, so the state estimation error was not calculated. All other metrics were obtained at different speeds, see Table 1 and Table 2.

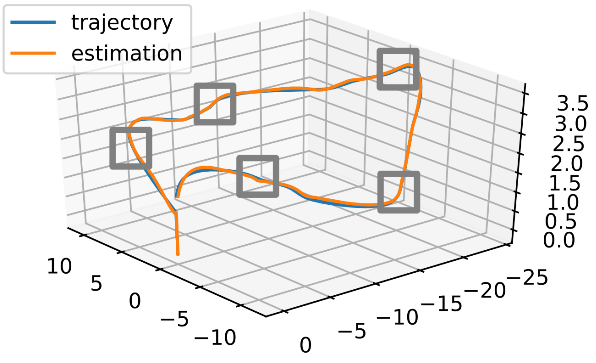

Figure 3: Path followed by the aircraft when passing through the simulated circuit with $v_{max} = 4.0(m/s)$ compared to the trajectory generated for the motion control behaviour.

	x-axis	y-axis	z-axis	total
$v_{max} = 1.0$	0.0552	0.0572	0.0602	0.0807
$v_{max} = 2.0$	0.0647	0.0831	0.0734	0.1168
$v_{max} = 3.0$	0.0959	0.0963	0.0924	0.1468
$v_{max} = 4.0$	0.1001	0.1103	0.0952	0.1583

Table 1: RMSE between trajectory reference and estimator measurements, expressed in meters, when the simulated quadrotor pass through the whole circuit with trajectories generated with different values of v_{max} parameter

	V_{max}	V_{avg}	Elapsed Time (s)
$v_{max} = 1.0$	0.8192	0.3848	127.6
$v_{max} = 2.0$	1.5414	0.6896	61.1
$v_{max} = 3.0$	2.7687	1.1828	38.8
$v_{max} = 4.0$	3.2657	1.2243	34.5

Table 2: Speeds (m/s) achieve during flights and elapsed time to complete the whole simulated circuit employing different values of v_{max} parameter in the trajectory generation.

4.2 Aerial Vehicle Platform

The aerial platform used for the real experiments was a custom quadrotor based on the DJI F330 frame, shown in Fig. 4. This platform was equipped with a Pixhawk 4 mini as the aircraft autopilot, an Intel Realsense T265 Tracking Module used for state estimation, and an USB fish-eye camera for gate detection.

Additionally, the aerial platform was equipped with a Single Board Computer (SBC) NVIDIA Jetson Xavier NX with an 6-core ARM v8.2 , 64-bit CPU running Ubuntu Linux 18.04 Bionic Beaver for on-board computing. All computations required for the real experiments occurred in this SBC.

In order to obtain the ground truth position of the aircraft during some experiments, a Motion Capture System (*mocap*) was used. For localizing the aircraft inside *mocap* area, several IR markers has been attached to the platform.

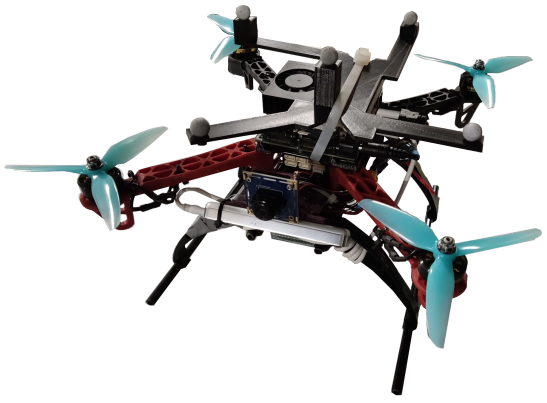

Figure 4: Quadrotor used for real flight experiments

4.3 *Real flights*

Due to space limitations in the *mocap* area we decided to do two different experiments: in the first one we make the drone pass through one gate with different speeds to evaluate the state estimation error and the trajectory following error of the controller, in the second one the aircraft had to pass through a small circuit with 4 gates to test the performance of the whole system in a complex task.

4.3.1 *One gate crossing*

For these experiments, the aircraft must localize a gate located in $gate_{position} = [2.0, 0.0, 1.5]\ (m)$ and pass through it with 4 different max speed $v_{max} = \{1.0, 2.0, 3.0, 4.0\}(m/s)$ values for the trajectory generation, see Fig. 5.

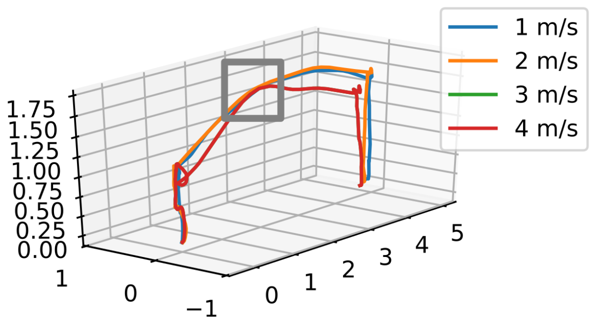

Figure 5: Path followed by the aircraft when passing through the gate at different speeds. The position measures had been obtained from the *mocap* system.

To measure the estimator error provided by the Realsense T265 Tracking Module when the quadrotor flies at different speeds we use *mocap* system for making a comparison between the ground truth and the estimated pose of the quadrotor, see Table 3.

	x-axis	y-axis	z-axis	total
$v_{max} = 1.0$	0.1140	0.0248	0.0990	0.1236
$v_{max} = 2.0$	0.1178	0.0173	0.1475	0.1561
$v_{max} = 3.0$	0.1079	0.0283	0.2313	0.2468
$v_{max} = 4.0$	0.1019	0.0545	0.3359	0.3474

Table 3: RMSE between Realsense estimation and ground truth measurements, expressed in meters, when the quadrotor flies through trajectories generated with different values of v_{max} parameter

Before flying through a more complex circuit, it is convenient to measure the trajectory following error of the controller employed when the aircraft flies at different speeds, see Table 4.

	x-axis	y-axis	z-axis	total
$v_{max} = 1.0$	0.0360	0.0262	0.0494	0.0558
$v_{max} = 2.0$	0.0683	0.0294	0.0633	0.0757
$v_{max} = 3.0$	0.1229	0.0385	0.0691	0.0948
$v_{max} = 4.0$	0.1699	0.0393	0.0820	0.0980

Table 4: RMSE between trajectory reference and estimator measurements, expressed in meters, when the quadrotor flies through trajectories generated with different values of v_{max} parameter

For evaluating the performance of the system passing through a drone racing circuit, other measures like the elapsed time to complete the circuit , the average flying speed, and the peak speed are needed, see Table 5.

	V_{max}	V_{avg}	Elapsed Time (s)
$v_{max} = 1.0$	0.9670	0.5625	9.4
$v_{max} = 2.0$	2.2556	0.9770	5.5
$v_{max} = 3.0$	3.1249	1.2059	4.7
$v_{max} = 4.0$	4.1673	1.5334	3.9

Table 5: Speeds (m/s) achieve during flights and elapsed time to complete the trajectory employing different values of v_{max} parameter in the trajectory generation.

4.3.2 *Four gates circuit crossing*

After validating the proper work of the whole system in the previous experiments, the last experiments consist in flying through a drone racing circuit with four gates arranged in the middle of the sides of a square of dimension 5x4 meters, with different heights each one.We fly through the circuit with two speed configurations: $v_{max} = 0.5$ and $v_{max} = 1.0$ as we can see in Fig 6 and Fig 7 respectively.

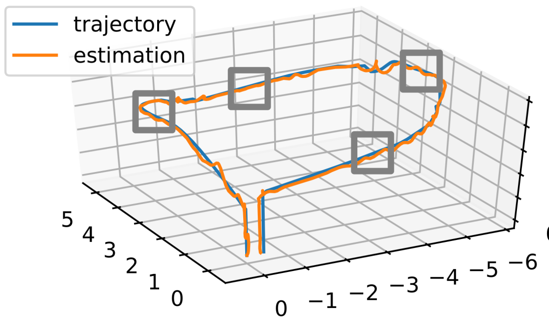

Figure 6: Path followed by the aircraft when passing through the 4 gates circuit with $v_{max} = 0.5(m/s)$ compared to the trajectory generated for the motion control behaviour.

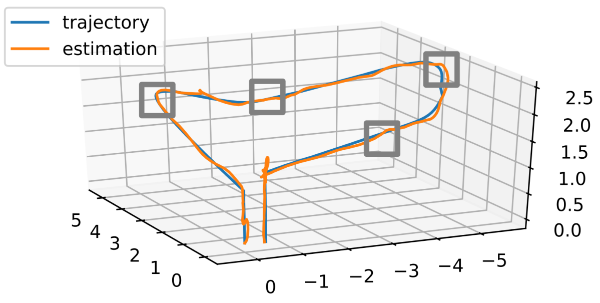

Figure 7: Path followed by the aircraft when passing through the 4 gates circuit with $v_{max} = 1.0(m/s)$ compared to the trajectory generated for the motion control behaviour.

Due to the space limitation of the arena, the ground truth poses were not acquired. However, the trajectory following errors of the controller and the time and speed metrics have been taken for comparing both flights, see Table 6 and Table 7.

	x-axis	y-axis	z-axis	total
$v_{max} = 0.5$	0.0492	0.0701	0.0576	0.0855
$v_{max} = 1.0$	0.0922	0.1491	0.1258	0.1414

Table 6: RMSE between trajectory reference and estimator measurements, expressed in meters, when the quadrotor pass through the whole circuit with trajectories generated with different values of v_{max} parameter

5 RESULTS DISCUSSION

On the simulated flights, the system was able to complete the whole circuit at different speeds reaching speeds up to

	V_{max}	V_{avg}	Elapsed Time (s)
$v_{max} = 0.5$	0.5316	0.2060	79.1
$v_{max} = 1.0$	0.9231	0.4125	32.9

Table 7: Speeds (m/s) achieve during flights and elapsed time to complete the whole circuit employing different values of v_{max} parameter in the trajectory generation.

3.2 m/s, with the trajectory following average errors around 15 centimeters. which validates the operation of the system. Real experiments show that the proposed framework allows a real quadrotor to fly through drone racing circuit gates at speeds up to 4 m/s with a small increase in the trajectory following error compared with the simulated runs. The state estimator sensor can reach average estimation errors higher than 35 centimeters, adding this error to the trajectory following error could make the quadrotor collide with the gates if their positioning were not updated with respect to the drone position as long as the quadrotor navigates through the circuit. However, the limited computing capabilities of the SBC makes that the trajectory generation spends a lot of time, which worsens performance of the system when flying at high speeds through multiple gates.

6 CONCLUSIONS AND FUTURE WORK

In this work, a modular framework for autonomous drone racing has been proposed and validated through several experiments, not only on simulation but also on real environments, being able to fly up to 4.16 m/s and to go through an small circuit successfully. The state-of-the-art algorithms proposed for each module, combined with the evaluation metrics proposed, constitute a baseline for research on improving autonomous agile drone flying.

To improve this framework, a wider range of possibilities to choose for each module must be given, like adding Predictive Model Controllers, learning-based gate estimation methods, or a Visual Inertial Odometry estimator fused with other sensors to improve the state estimation. Moreover, the domain gap between simulation and real life is substantial when non-photorealistic simulators are used. Using a simulator like Flightmare[6] would help to develop and test algorithms that rely on images taken through the flight.

ACKNOWLEDGEMENTS

This work has been supported by the project COMCISE RTI2018-100847-B-C21, funded by the Spanish Ministry of Science, Innovation and Universities (MCIU/AEI/FEDER, UE). The work of the first author is supported by the Research Assistant and Laboratory Technicians Program of the Science, University and Innovation Department of Madrid Government (partially funded by FEDER/EU) The work of the fourth author is supported Pre- and Post Doctoral Program of the Science, University and Innovation Department of Madrid Government (partially funded by FEDER/EU).

References

[1] Hyungpil Moon, Jose Martinez-Carranza, Titus Cieslewski, Matthias Faessler, Davide Falanga, Alessandro Simovic, Davide Scaramuzza, Shuo Li, Michael Ozo, Christophe De Wagter, Guido Croon, Sunyou Hwang, Sunggoo Jung, Hyunchul Shim, Haeryang Kim, Minhyuk Park, Tsz-Chiu Au, and si-jung Kim. Challenges and implemented technologies used in autonomous drone racing. *Intelligent Service Robotics*, 12, 04 2019.

[2] Elia Kaufmann, Mathias Gehrig, P. Foehn, René Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. *2019 International Conference on Robotics and Automation (ICRA)*, pages 690–696, 2019.

[3] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Alphapilot: Autonomous drone racing. 05 2020.

[4] Y.-W. Kang S.-Y. Shin and Y.-G. Kim. Report for game of drones: A neurips 2019 competition.

[5] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality, 2019.

[6] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. 2020.

[7] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish Kapoor. Airsim drone racing lab. *arXiv preprint arXiv:2003.05654*, 2020.

[8] Jose Luis Sanchez-Lopez, Martin Molina, Hriday Bavle, Carlos Sampedro, Ramon A Suarez Fernandez, and Pascual Campoy. A multi-layered component-based approach for the development of aerial robotic systems: The aerostack framework. *Journal of Intelligent & Robotic Systems*, 88(2), 2017.

[9] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[10] Martin Molina and Pablo Santamaria. Behavior coordination for self-adaptive robots using constraint-based configuration. Arxiv preprint, 2021. arXiv:2103.13128.

[11] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[12] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[13] Michael Burri, Helen Oleynikova, , Markus W. Achtelik, and Roland Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments. In *Intelligent Robots and Systems (IROS 2015), 2015 IEEE/RSJ International Conference on*, Sept 2015.

[14] S Lynen, M Achtelik, S Weiss, M Chli, and R Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2013.

[15] Sergio Garrido-Jurado, Rafael Munoz-Salinas, Francisco José Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481–491, 2016.

[16] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[17] Francisco J Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and vision Computing*, 76:38–47, 2018.

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