BOARR : A Benchmark for quadrotor Obstacle Avoidance based on ROS and RotorS

Thibaut Tezenas Du Montcel, Amaury Nègre, Jose-Ernesto Gomez-Balderas, Nicolas Marchand

BOARR : A Benchmark for quadrotor Obstacle Avoidance based on ROS and RotorS

T. Tezenas Du Montcel*[1], A. Nègre[1], E. Gomez-Balderas[1], and N. Marchand†[1]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, 38000 Grenoble, France

Abstract

Multiple obstacle avoidance algorithms have been proposed over the past years but they were not tested using a common protocol. Some were tested statistically by repeating a task in specific simulated worlds. Others went through a number of real handmade obstacles. The last ones did lengthy real flights in environments that are tough to characterize. This paper proposes the BOARR benchmark that aims to give a common framework to test and compare obstacle avoidance algorithms for quadrotors. It offers multiple sensors and multiple indicators relevant to all quadrotor obstacle avoidance algorithms. It uses ROS, Gazebo and RotorS and can be easily deployed.

1 Introduction

Benchmarks are a lot less common in robotics than in other fields such as machine learning or computer vision. This can be easily explained considering the numerous difficulties to reach conditions that are close enough from one place to another to make the results of two experiments comparable. Among

*Corresponding Author: thibaut.tezenas-du-montcel@gipsa-lab.fr
†Corresponding Author: nicolas.marchand@gipsa-lab.fr

the difficulties, we can evoke the characterization of a precise environment which very often is important for a mission to succeed. At the least, building environments that anyone could reproduce in the lab would create conditions that would be far from those of the outside world. At the same time, it would be difficult to ask a research team to test numerous algorithms because it is time consuming and involves risks in damaging costly robots. An alternative to benchmarks to compare robotics solutions are challenges such as the DARPA robotics challenge or the Autonomous Drone Racing Competition [1]. However those challenges are costly and often considered as goals for projects more than an opportunity for early test phases. Benchmarks are appealing because they can be used both in the course of a research project as well as to share results. They also allow for direct comparison between different solutions to a specific task. More importantly maybe, using a common benchmark is a step towards compatibility between multiple algorithms and therefore towards reproducibility. For all these reasons and despite the difficulties, some benchmarks have been proposed in robotics. Some of them test a single robotic function, like KITTI [2] that tests depth estimation in road environments or the RGB-D dataset for SLAM [3]. Others consist in precise test protocols to reduce real life test disparities [4, 5, 6]. Finally, another common way to benchmark in robotics is to use simulation. Despite giving results that are approximative when compared to those obtained in real conditions, simulation can still give an idea of the advantages and drawbacks of the proposed algorithms. Moreover, a full simulation-based benchmark environment is often of much help during the development phase of a project. Finally, because of time constraints, statistical analysis are rarely done while doing real experiments as it is much easier to do so in simulation [7]. To the authors knowledge, there are no existing benchmarks for obstacle avoidance. This is the reason why the BOARR benchmark has been developed. It can be downloaded at `https://github.com/Gipsa-lab-PFP/BOARR`.

Obstacle avoidance is a core functionality that is needed when considering high level tasks. For quadrotors, those high level tasks can be for instance 24/7 monitoring, package delivery or fire fighting. All projects trying to create such an application would benefit from a local re-planner that is collision free, allowing to focus on the specificity of the targeted application instead of developing its own collision free local re-planner. If multiple works focus solely on collision avoidance for quadrotors (see for instance the survey paper [8] and the references therein), it remains complex for non specialists to select the best existing algorithm in order to save time for higher level

tasks. Is it for instance safer to choose an algorithm that can fly across 20 trees without any collision or is it safer to choose an algorithm that can fly 600 m before a collision ? There is an obvious lack of a generic comparison framework for obstacle avoidance algorithms.

In this paper, the first obstacle avoidance benchmarking environment for quadrotors is proposed. It is fully open since it is based on a simulator that the authors believe to be one of the most commonly used by many teams working on low level control features. Multiple sensors are also provided in order to be really close to the different choices that have been made during the last years. A forest was chosen as test environment. Indeed, it appears to be the environment that is one of the more complex and undoubtedly the most widely chosen for research to test navigation algorithms. This paper is organized in three main sections. In the first one, the test environment is presented. It includes the simulation engine choice, the frame and sensor choices and the generated worlds. In the second section, a step by step use of the benchmark is proposed. It starts by explaining how the check in a fast and easy way the compatibility of an algorithm with the benchmark and then details how to obtain advanced statistical results with the benchmark. In the third and last section, all the indicators given by the benchmark are explained as well as their statistical meanings. It has been chosen to have multiple statistical indicators since the choice of a good obstacle avoidance algorithm highly depends on the application and especially on the needed ratio between efficiency and safety.

2 Test environment

2.1 Simulation engine

The state of the art shows that, to the exception of custom simulators and MATLAB/Simulink which hardly manages simulated worlds, two open source simulators are mainly used when working with simulated quadrotors: Microsoft AirSim [9] for high level tasks and RotorS [10] for low level control purpose. If both are ROS compatible-a feature that now appears essential for research in the field-the integration of RotorS appears more convenient because easier and faster. Microsoft AirSim allows for flight in environments that are way more realistic than what RotorS-which is based on Gazebo [11]-can offer. The main advantage of using those realistic environments is the

greater quality of the visual outputs compared to what Gazebo can offer. However, as Microsoft AirSim uses either Unity or Unreal engine-neither of which is open-source-it restricts the use and distribution of AirSim-based benchmarks as simulation engines. Moreover, since obstacle avoidance testing requires high dynamics flights, the focus should be on closely approaching the dynamics of the quadrotors as it is the case with RotorS. If, noisy sensing is crucial to test robustness, it is not important for the noise be a realistic. For all those reasons, RotorS has been chosen over Microsoft AirSim.

RotorS, and by extension the BOARR benchmark, uses ROS-kinetic and Gazebo 7.0. It can simulate UAV frames of any number of rotors. In this benchmark, the four rotors configuration known as the quadrotor was chosen since it is by far the configuration which is the most commonly used by researchers. Once the configuration has been chosen, the dimension of the frame (size and weight) is the discriminating factor. In the quadrotor literature, the frames diameters are usually comprised between 30 cm (e.g. Parrot Bebop 2) and up to approximately 1 meter (e.g. Astec Pelican). These recent years, a tendency to go with smaller frames and less expensive sensors has been observed. To illustrate, one of the first papers on obstacle avoidance for UAV featured a 95kg Helicopter [12], when in 2015/2016, one of the most common frame in used to be the Asctec Pelican [13, 14] to reach nowadays Bebop-like platforms as in [15, 16, 17, 18]. It is to be noted that these works focused solely on monocular videos. In order to stay close to a large number of used platforms in terms of size and weight, an Astec Hummingbird frame-55cm diameter and 710 g weight-was chosen. These size and weight characteristics are a trade-off between what has been observed in the literature of the last years and the recent tendency to implement obstacle avoidance algorithms on smaller frames. It is situated in the lower half of what can be found in the literature.

The RotorS package includes multiple controllers that allow a control of the simulated quadrotor either in position, in attitude or directly in motor speed. To ease the use of the benchmark, all of these RotorS controllers are available so that it is possible to control the system by using any of these control approaches.

2.2 Generated worlds

Two types of environments are proposed. The first one is composed of vertical, randomly positioned cylinders. It is a 40 m×40 m square that is sur-

rounded by walls constraining the quadrotor to stay close to the cylinders. This environment is a simplified environment that has the advantage to be really light to run. It can be used to check the compatibility of an obstacle avoidance algorithm with the proposed benchmark. Figure 1 presents a view of this environment.

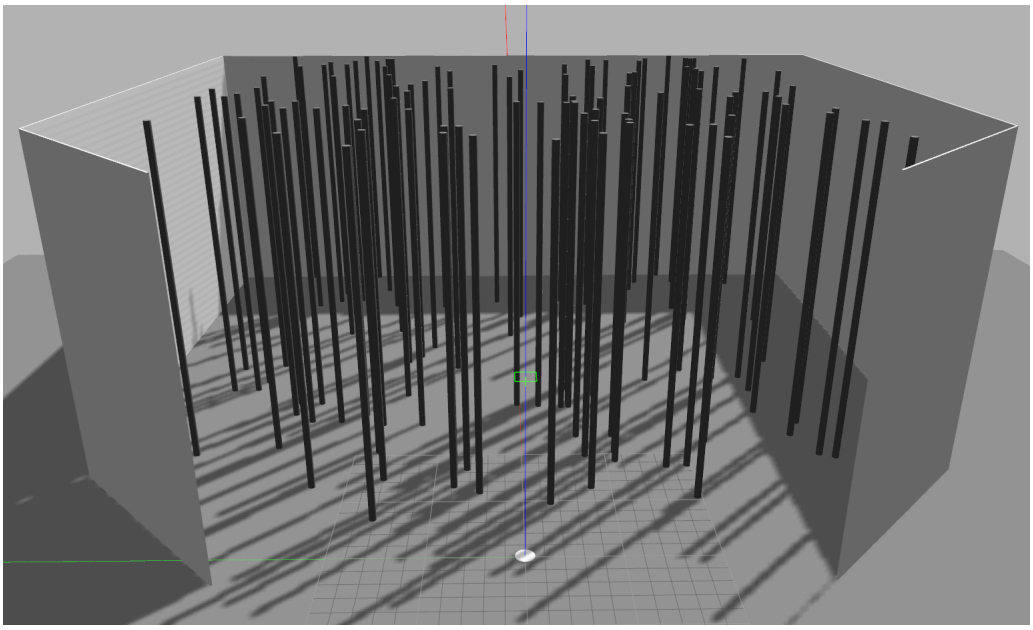


Figure 1: A simple world with cylindrical obstacles.

The second and main testing environment consists in randomly generated 80 m×80 m square forests. Their ground reliefs are extracted from Digital Elevation Model files of multiple locations in the US that can be found in The National Map [19]. There are 5 realistic ground profiles of different types: one profile is flat, another one is a constant slope, while the last three are mixed terrain with valley and hills. 20 different low-poly models of trees generated by a procedural tree generator are used [20]. Each tree is slightly randomly tilted along the horizontal axes away from verticality and then randomly rotated along the vertical axis. Thanks to these slight modifications on the orientation of the trees, 20 models are enough to generate real-like forests with sufficient diversity whether in shape or in color as shown in Figure 2. With those ground profiles and tree models, the forests are generated by randomly selecting 100 trees and then by placing each one of the trees on

a ground profile. This positioning is a sum of three Gaussian distributions which have different standard deviations allowing to create a world that includes a low density part (when looking from above, less than 40% of the space is occupied by a tree), a mid density forest (between 40 and 70% occupation) and a high density forest (over 70% occupation). It is expected that those high/medium/low density areas will play an important role in helping to differentiate the tested algorithms. Some algorithms will outperform others in low density areas whereas others may perform better in high density areas.

Figure 2: A forest view from above. The high density area is inside the red ellipse, the mid density area inside the orange one and the low density zone is in green.

Finally some wind effect has been added in the forest environments using a wind plugin that is slightly different from the one which is proposed by

RotorS. Indeed, RotorS proposes either a constant wind with a single wind gust or a static wind field. Neither of those scenarios is adapted to obstacle avoidance tests which require challenging wind conditions changing over period of several minutes. The proposed plugin consists in a continuous base wind with changing speed and direction on which recurrent wind gusts are added. It can be found in the `gazebogazebo_plugins` directory.

In order to discover the environments and tools proposed in the benchmark, a dummy algorithm that will navigate in straight line is provided. This algorithm can be started using the `dummy_perfect_sensing_benchmark` and the `dummy_noisy_benchmark` scripts. Its use will allow the user to see a live report of the progress of the quadrotor in the environment alongside an on-board view of one of the quadrotor cameras.

2.3 Sensors

The sensors used are mostly taken from the RotorS package which offers a broad range of sensors. All the sensors, except cameras, are proposed in an exact and a noisy version. The exact version gives, as its name indicates, perfectly sensed values while the noisy version adds noises observed on physical sensors. All the sensors are streaming the standard ROS message dedicated to their type of output data. The simulated quadrotor is always equipped with all the sensors, but, for practical reasons, all of them are considered weightless (or of weight already included in the frame weight considered above).

IMU/GPS/baro sensors: The detailed list of the chosen sensors is the following. The IMU (Inertial Measurement Unit) is either a perfect IMU running at 500 Hz or a simulated MPU-6000 limited at 500 Hz. A perfect *magnetometer* running at 100 Hz is proposed while the noisy one corresponds to a HMC5883L at 75 Hz. The perfect *barometer* runs at 50 Hz while the noisy barometer corresponds to a MS5611 at the same frequency. For the *GPS*, the perfect one runs at 10 Hz while the noisy one is a simulated version of a neo-m8n GPS at the same frequency.

Cameras: Two *cameras* at 30 Hz are also available. One is looking downwards as some algorithms use such cameras to compute the visual odometry, and the second one is looking frontward which is the appropriate direction to

detect obstacles for monocular based algorithms. Those cameras are only proposed in their perfect version since working on a noisy rendering on Gazebo worlds seems clearly limited compared to Microsoft AirSim that already provides high quality rendering thanks to the Unreal Engine.

Collision sensor: In Gazebo, a collision sensor is dedicated to the detection of any collision between a single collision box and the rest of the world. To keep a single collision sensor, a cylinder was defined so that it just includes the shape of the quadrotor as figure 3 shows. This collision box is slightly bigger than the quadrotor itself but for in-flight safety reasons, this is acceptable. A quadrotor that would fly less than 2 or 3 cm away from obstacles would be vulnerable to wind gust and could not be considered safe.

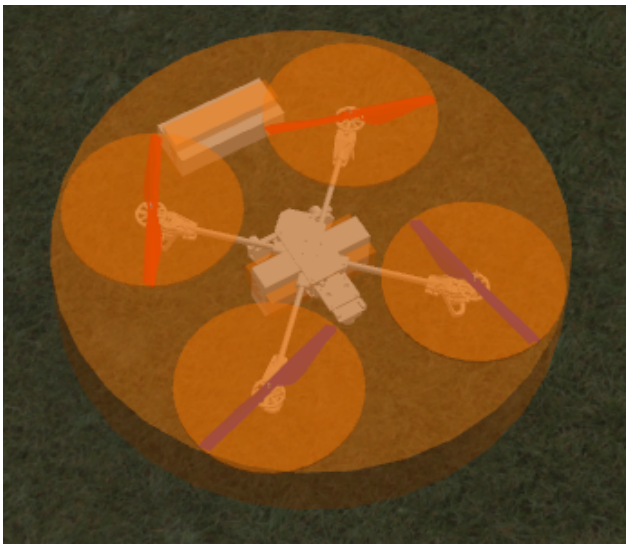
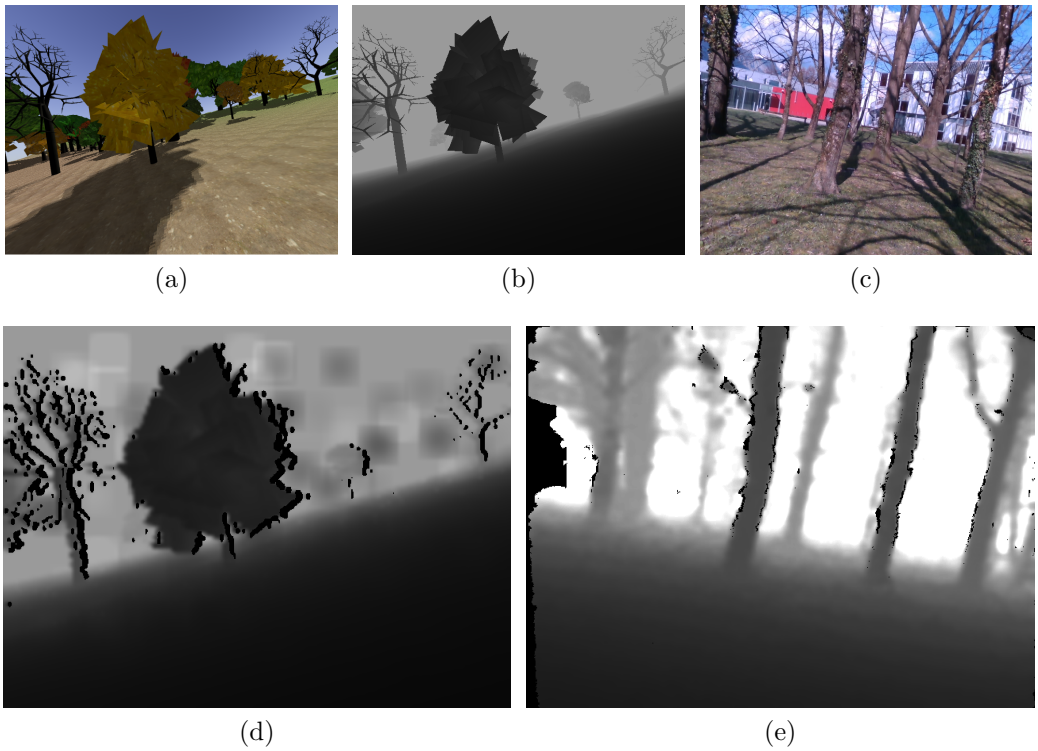


Figure 3: In orange, the cylindrical collision box of the simulated quadrotor.

Depth sensors: Finally, focusing on the depth sensors, two frontward looking depth sensors are proposed. The field of view of the first depth sensor is 67° horizontal and 42° vertical while its depth range is [0.15m, 10m] corresponding to an Intel® RealSense™. A perfect version of this depth sensor is available and a noised version has been created by adding some localized gaussian noise in the depth image. This noise is proportional to the square of the depth. Figure 4(a) shows the color image of a simulated

scene. Figure 4(b) shows the depth perceived by what would be a perfect Intel® RealSense™. Figure 4(d) shows the artificially noised output. Figure 4(c) shows the color image of a real scene of an environment comparable to (a) and (e) gives a real depth image perceived by a real Intel® RealSense™. The second depth sensor aims to simulate a Velodyne HDL-32E with a 41° vertical and 360° horizontal field of view at 10 Hz with a minimum depth of 1 m and a maximum depth of 100 m. A noised version that adds a point by point Gaussian noise ($\sigma = 2\,\text{cm}$) to the point cloud is also proposed.



(a) (b) (c)

(d) (e)

Figure 4: In (a), a Realsense RGB view of a simulated forest. (b) is the perfect depth corresponding to the perception of a perfect Realsense and (d) the noised detph perceived by the noised realsense. (c) and (e) are an RGB and a depth image perceived by a Realsense in a real forest.

It is possible and quite fast to create a specific sensor setup. However any sensor choice should be detailed when giving the statistical results of the benchmark, whether it is in the proposed configurations or corresponds to a specific setup.

3 Step by step use of the benchmark

The benchmark is based on ROS. Its usage requires basic knowledge of this middleware. A tutorial can be found at [21]. The benchmark has been created with 3 progressive steps. The first step is a preliminary step that allows to verify, particularly with the use a real-time video output, the compatibility between any proposed algorithm and the benchmarking environment. The second step focuses on visually verifying the performances before starting the statistical analysis while the third step consists in the statistical analysis of the algorithm itself.

3.1 Simple test in a perfect light-weight environment

After downloading the benchmark code from `https://github.com/Gipsa-lab-PFP/BOARR` and RotorS forked sources from `https://github.com/Gipsa-lab-PFP/RotorS`, the first step will consist in a compatibility test using the light world with cylindrical obstacles that is displayed in Figure 1. The test in itself is quite simple, it simply consists in travelling 100 m between three predefined waypoints in this world. There is no wind and no noise on any sensors. The light weight of this environment allows for fast restarts if needed until the quadrotor flies as expected. This phase consists in three main tasks :

- editing your current launch files to remap to the defined topics, renaming the files and placing them in a specific directory.
- editing the sole configuration file which consists in selecting your control method
- adapting your inertial frame and sensors frames if needed

Those three simple actions are described in detail in the doc/UseYourOwnAlgorithm.md file and should be sufficient to obtain a compatible algorithm. Once your algorithm is compatible, the compatibility test can be executed using a single bash script as stated in the documentation file.

3.2 Unit test in the forest environment

After a couple of successful flights in the perfect world, a transition to a Unit test in the forests is the last step before starting the statistical analysis. What is called a Unit test is a single test from the hundred tests that will be executed during the statistical analysis phase. It consists in attempting to fly, using the noisy sensors, for exactly one kilometer between successive waypoints in a windy forest environment without the collision sensor detecting any collision.

Sixteen waypoints have been defined, with the same layout, in each generated forest. They are represented as crosses in black, red, blue or green on Figure 5. Starting from the bottom right waypoint and moving clock-wise, each of them have been given an ID from 0 to 15. Each test randomly starts at a waypoint. Lets note L the ID of this waypoint, the next waypoint is then randomly selected between waypoints with the Id $(L + 5 \vee 6 \vee 7)\%16$. This process is repeated as long as needed to generate flight paths of the desired length. After multiple tests, this was an efficient strategy to cover the whole terrain while avoiding round trips. Figure 5 shows this process for the first two waypoints of a test. The distance travelled by the quadrotor is defined as the linear distance between all the previously reached waypoints (distance between W_1 and W_2 in figure 5) added to the distance between the last reached waypoint (W_2 in figure 5) and the perpendicular projection of the current position of the quadrotor on the straight line linking the previous (W_2 in figure 5) and the targeted waypoints (W_3 in figure 5). Each test is stopped either at the first detected collision or when this travelled distance exceeds 1km.

For all tests, a video can be automatically generated and is available in real time. This is particularly enlightening in this Unit test phase as it allows to monitor the behavior induced by the tested algorithm and to correct it if needed. As the first test presented in subsection 3.1, the latter can be executed using a single bash script.

3.3 Multiple tests in the forest

This is the last step. It consists in doing all the tests that will then be statistically analyzed to characterize your algorithm. As for the other phases, the whole phase is launched using a single bash script. Its completion should take around a week working with Gazebo running in real time. This is

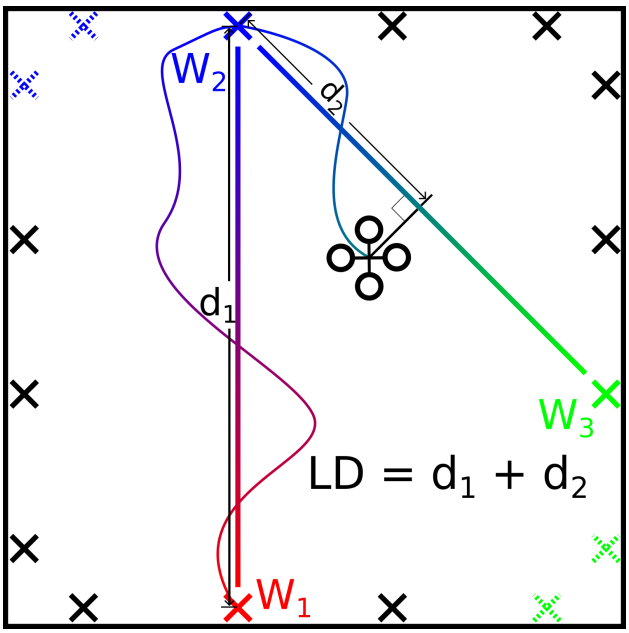


Figure 5: A schematic view from above of the waypoints and of the computation of the travelled distance.

most likely possible using a 500$ computer if the tested algorithm is not too computationally expensive. It can even be faster using the Gazebo speed up factor if your algorithm and computer performances allow it. Each Unit test will append a line describing its results in a text file that is generated in the result/generated_files/<date_of_the_test> directory. The generated file will then be used as the source for the statistical analysis detailed in the next section.

4 Statistical analysis of the benchmark results

The main goal of this analysis is to extract a number of indicators describing the general performances of the tested algorithm from the previously saved file describing each of the performed Unit tests. A primary indicator based on the probability of a collision occurring for a given travelled distance is calculated. This indicator will also determine the number of Unit tests needed

to conduct this statistical analysis. Other indicators are also proposed to help distinguish the advantages and drawbacks of each of the tested algorithms.

4.1 Collision probability and number of tests

As stated before, each Unit test consists in a flight of one linear kilometer between successive way-points. Using a probabilistic approach, it is possible, for N tests, to compute an estimation $\hat{p}$ of the probability p to fly one linear kilometer without collision as :

$$\hat{p} = \frac{1}{N}\sum_{i=1}^{n} X_i \quad \text{with } X_i = \begin{cases} 1, & \text{if No Collision on Test } i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

It is then natural to try to express a bound on the relative error between the estimated probability and the real probability to complete a one kilometer flight between waypoints without collision. In a probabilistic way, with ϵ being the estimation error and λ being the probability that the estimation is outside of this bound error, it can be expressed as followed :

$$Pr(|p - \hat{p}| \leq \epsilon) \geq 1 - \lambda \tag{2}$$

The Chernoff bound then gives an expression of a minimum bound N_{min} on the number of test needed N to ensure any λ and ϵ :

$$N > \frac{\ln\left(\frac{2}{\lambda}\right)}{2\epsilon^2} \tag{3}$$

An arbitrary choice then needs to be made in order to find a trade off between the precision on the bound ϵ, the probability to be inside this bound λ and the number of test N. Table 1 shows the number of tests necessary to ensure different precision and incertitude bounds.

ϵ	0.01	0.02	0.05	0.05	0.1
λ	0.01	0.01	0.01	0.05	0.01
N_{min}	26 492	6 623	1 060	738	265

Table 1: Number of tests for different precision and incertitude bounds.

We chose a 5% probability bound with a 1% probability to be outside the bound. As a consequence, the Chernoff bound indicates that at least 1 060

tests ensure those bounds. For everyone to have exactly the same conditions during the tests, the necessary 1 060 Gazebo worlds have all been generated and are available in the `gazebo/benchmark_v1.0_worlds` directory of the benchmark package. For each world, a sequence of waypoints is generated and saved in a `yaml` file associated to the world file.

4.2 Other indicators

While the collision probability p is the main indicator of the performance of the proposed algorithm, other indicators play an important role and give clear indications on how an algorithm performs. These indicators are listed below:

Average completion time: This is the mean over the tests of the time needed to travel the one kilometer. Lowering the mission execution time of a task often comes as a very important matter, either to do more with a single battery or to increase efficiency.

Average flying speed: This is not necessarily the same as the completion time as some algorithm may do more detours than others. This indicator will be particularly relevant when flying in sparse environments in which the possible detours will have less importance compared to the average flying speed.

Average flying distance: This indicator quantifies the detours that are taken in dodging the obstacles in the environment. It will benefit the algorithm that tends to fly close to the obstacles but may reduce the quadrotor's speed to do so.

Average energy spent to complete a test: Algorithms with low completion time should best perform in this category as they also lower the cost of gravity compensation. This indicator quantifies the smoothness of the pace of the UAV since higher accelerations have higher impact on the energy consumption.

5 Conclusion

Multiple obstacle avoidance algorithms have been proposed over the years and it is right now nearly impossible to compare one's contribution to the

existing algorithms. One reason is that some are closed-source and can not be used by everyone. Another reason is that any comparison to an existing code requires to reprogram/interface it and implies spending a great deal of time and an important risk of making mistakes. Therefore, this paper has been focused on proposing a statistical obstacle avoidance benchmarking environment for quadrotors. By creating such a benchmark that is as generic and as user-friendly as possible, we aim to help obstacle avoidance algorithm creators to develop and test their own algorithms as well as to give indicators to compare them with other existing algorithms. The proposed benchmark being open-source, the modelling, sensors, quadrotor simulation can be improved by the community which can in turn add tests to better fit its particular needs. Also note that this kind of statistical testing could be extended to other tasks such as grasping or short distance inspection.

6 Acknowledgment

This work is founded by the CAP2018 project [22].

References

[1] "Iros 2018 autonomous drone racing competition." [Online]. Available: http://rise.skku.edu/iros2018racing/index.php/iros-2018-adr/

[2] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354–3361.

[3] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 573–580.

[4] C. Sprunk, J. Röwekämper, G. Parent, L. Spinello, G. D. Tipaldi, W. Burgard, and M. Jalobeanu, "An experimental protocol for benchmarking robotic indoor navigation," in *Experimental Robotics*. Springer, 2016, pp. 487–504.

[5] J. Leitner, A. W. Tow, N. Sünderhauf, J. E. Dean, J. W. Durham, M. Cooper, M. Eich, C. Lehnert, R. Mangels, C. McCool *et al.*, "The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4705–4712.

[6] O. Michel, F. Rohrer, and Y. Bourquin, "Rat's life: A cognitive robotics benchmark," in *European Robotics Symposium 2008*. Springer, 2008, pp. 223–232.

[7] J. Quilbeuf, M. Barbier, L. Rummelhard, C. Laugier, A. Legay, B. Baudouin, T. Genevois, J. Ibañez-Guzmán, and O. Simonin, "Statistical model checking applied on perception and decision-making systems for autonomous driving," in *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at the IEEE International Conference on Intelligent Robots and Systems, october 2018*, 2018.

[8] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, vol. 29, no. 2, pp. 315–378, 2012.

[9] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.

[10] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular gazebo mav simulator framework," in *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[11] Gazebo. [Online]. Available: http://gazebosim.org/

[12] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *International Journal of Robotics Research*, vol. 27, no. 5, pp. 549–574, 2008.

[13] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1484–1491.

[14] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[15] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, and L. Van Eycken, "Cnn-based single image obstacle avoidance on a quadrotor," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 6369–6374.

[16] S. Yang, S. Konam, C. Ma, S. Rosenthal, M. Veloso, and S. Scherer, "Obstacle avoidance through deep networks based intermediate perception," *arXiv preprint arXiv:1704.08759*, 2017.

[17] W. G. Aguilar, V. P. Casaliglla, and J. L. Pólit, "Obstacle avoidance based-visual navigation for micro aerial vehicles," *Electronics*, vol. 6, no. 1, p. 10, 2017.

[18] T. Nägeli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges, "Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1696–1703, 2017.

[19] "Usgs : The national map." [Online]. Available: https://viewer.nationalmap.gov/basic/

[20] jarikomppa. [Online]. Available: https://github.com/jarikomppa/proctree/

[21] O. S. R. Foundation. [Online]. Available: https://wiki.ros.org/ROS/Tutorials/

[22] CAP 2018. [Online]. Available: http://cap2018.minalogic.net/