Delft University of Technology

AvoidBench
A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors

Yu, Hang; De Croon, Guido C.H.E.; De Wagter, Christophe

DOI
10.1109/ICRA48891.2023.10161097

Publication date
2023

Document Version
Final published version

Published in
Proceedings - ICRA 2023

Citation (APA)
Yu, H., De Croon, G. C. H. E., & De Wagter, C. (2023). AvoidBench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors. In *Proceedings - ICRA 2023: IEEE International Conference on Robotics and Automation* (pp. 9183-9189). (Proceedings - IEEE International Conference on Robotics and Automation; Vol. 2023-May). IEEE. https://doi.org/10.1109/ICRA48891.2023.10161097

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher
is the copyright holder of this work and the author uses the
Dutch legislation to make this work public.

2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10161097

AvoidBench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors

Hang Yu, Guido C. H. E de Croon and Christophe De Wagter

***Abstract*— Obstacle avoidance is an essential topic in the field of autonomous drone research. When choosing an avoidance algorithm, many different options are available, each with their advantages and disadvantages. As there is currently no consensus on testing methods, it is quite challenging to compare the performance between algorithms. In this paper, we propose AvoidBench, a benchmarking suite which can evaluate the performance of vision-based obstacle avoidance algorithms by subjecting them to a series of tasks. Thanks to the high fidelity of multi-rotors dynamics from RotorS and virtual scenes of Unity3D, AvoidBench can realize realistic simulated flight experiments. Compared to current drone simulators, we propose and implement both performance and environment metrics to reveal the suitability of obstacle avoidance algorithms for environments of different complexity. To illustrate AvoidBench's usage, we compare three algorithms: Ego-planner, MBPlanner, and Agile-autonomy. The trends observed are validated with real-world obstacle avoidance experiments. Code is available at: https://github.com/tudelft/AvoidBench**

I. INTRODUCTION

Autonomous drones can be applied in several novel fields such as forest rescue, cave exploration [1], [2], [3], and greenhouse or warehouse monitoring [4]. One common theme between all these cases is that obstacle avoidance plays a large role for the safety and effectivity of these vehicles. In particular, vision-based obstacle avoidance is popular, since vision sensor can be compact, light-weight and low cost. Unfortunately, there is currently no shared consensus on how vision-based obstacle avoidance algorithms should be tested. This makes it not only difficult to see how the field of obstacle avoidance as a whole is progressing, but it also makes it hard to compare the performance of different obstacle avoidance algorithms. There is a dire need for reliable benchmarking.

There are many benchmarks for fields like computer vision or natural language processing (NLP), such as KITTi [5], ImageNet [6] and GLUE [7]. They all provide large datasets and proper metrics for evaluating and comparing different algorithms. This allows researchers not only to compare state-of-the-art algorithms but also to see how performance has increased over time. Furthermore, it offers an invaluable platform for researchers to showcase and share their algorithms and garner more interest from the community. If executed well, a benchmark can give a boost to the entire community and research field.

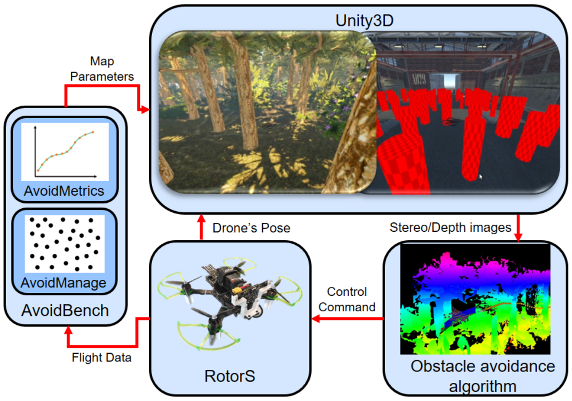

Fig. 1. The basic framework of AvoidBench. It uses the dynamics model from RotorS and photo-realistic scenes from Unity3D. Users can implement their obstacle avoidance algorithms by Python or C++ interfaces.

However, benchmarking algorithms for robotics is more difficult than for fields such as computer vision and NLP. This is mainly due to two reasons: (1) We cannot rely on a fixed dataset, since robots interact with their environment. Consequently, the sensory data and situations they encounter depend on the proposed autonomy algorithms. (2) A robot's success is dependent on a myriad of factors. These factors include real-world environment factors, ranging from its visual appearance to disturbances such as wind gusts, to an intricate internal pipeline with processes such as perception, planning and control. Hence, the number of interfering factors of performance are more than fields of computer vision and NLP.

There are already some mature simulators both for drones and rendering systems. For dynamics models, Gazebo-based simulators Hector [8] and RotorS [9] are both good choices, but Gazebo does not provide a photo-realistic visual environment. This low fidelity would lead to a large sim2real gap, and hence reduce the generalization of the benchmarking performances to real robotic platforms and environments.

The progress in the realism of game simulators is an opportunity for benchmarking in robotics. Specifically, rendering engines such as Unity3D and Unreal Engine generate photo-realistic environments. Airsim is a robot simulation platform developed by Microsoft and uses Unreal Engine 4 [10], and it provides a pure C++ and Python API to interface with the simulator. Another advantage of Airsim is that it supports hardware-in-the-loop (HITL) as well as software-in-the-loop (SITL) with flight controllers such as PX4. However, Airsim is tightly coupled with the rendering

All authors are with Faculty of Aerospace Engineering, Delft University of Technology, 2629 HS Delft, The Netherlands. (email: `h.y.yu@tudelft.nl`; `g.c.h.e.deCroon@tudelft.nl`; `c.dewagter@tudelft.nl`).

979-8-3503-2365-8/23/$31.00 ©2023 IEEE

engine for the drone's dynamics and control system, so the simulation speed is limited and researchers usually use two devices to handle the rendering process and control process relatively. Flightmare [11] is a more flexible simulator for quadrotors. It uses Unity3D for rendering and a Gazebo-based quadrotor dynamics [9] as well as a parallelized implementation of classical quadrotor dynamics model used for reinforcement learning.

There are also some benchmarks for autonomous drones. MAVBench [12] is a benchmark simulator to measure the performance of drones executing different missions such as search-and-rescue, mapping and planning. Using Airsim and Unreal Engine 4 as a backend, MAVBench is the first end-to-end photo-realistic benchmarking suite for MAVs. Although it supports many tasks, only time and energy are considered as the metrics and the complexity of environments is ignored. BOARR [13] is a benchmarking suite specifically designed to evaluate the performance of obstacle avoidance algorithms. Unfortunately, as BOARR is using Gazebo as a simulator, the visual fidelity is quite poor. We believe that in applications such as vision-based obstacle avoidance, visual realism plays a large role in how an algorithm performs.

In this article, we propose AvoidBench as a benchmarking suite for evaluating the performance of vision-based obstacle avoidance algorithms. We choose Flightmare as the basic backbone of AvoidBench, because it is lighter and can achieve higher simulation speed than Airsim. Based on Flightmare, we have for now built two simulation scenes for benchmarking: a forest environment and an indoor environment. It is easy to change the distribution of obstacles and complexity of map so that researchers can reveal the potential of drones using their algorithms. And we propose a complete set of metrics which contain the flight performance and environment complexity to evaluate the obstacle avoidance algorithms.

Basically, AvoidBench has the following three contributions: (1) We have developed a benchmark with high-fidelity visual scenes that allows to easily and objectively compare different algorithms in different conditions. The benchmark has been validated with real-world experiments; (2) Expanding upon Nous et al.'s work [14], a complete set of evaluation metrics including the flight performance and environment complexity is proposed; (3) Different types of obstacle avoidance algorithms including learning-based, optimization-based and motion-primitive-based are tested in AvoidBench and the performance of each algorithm is evaluated objectively both in simulation and the real world; (4) We provide an open source software framework in which users can easily enter their algorithms in Python or C++, determining their performance in relation to various environment metrics.

II. BENCHMARKING PIPELINE

Based on Flightmare, we propose AvoidBench which can evaluate the performance of vision-based obstacle avoidance algorithms for multi-rotors in simulation. AvoidBench assigns performance scores to obstacle avoidance algorithms by subjecting them to a series of tasks. Due to the photo-realistic of Unity3D, we can minimize the discrepancy between simulated and real-world sensors' observations. As shown in Figure 2, AvoidBench is a software suite that consists of four main modules: (1) virtual scenes based on Unity3D; (2) communication interface (AvoidLib) between the C++ side and the Unity side which is based on ZeroMQ; (3) AvoidManage which is used to combine the rendering engine and dynamics model as well as the management of the whole obstacle avoidance task; (4) AvoidMetrics which contains both performance and environment metrics to evaluate the obstacle avoidance task. In the remainder of this section, we explain each of the AvoidBench modules.

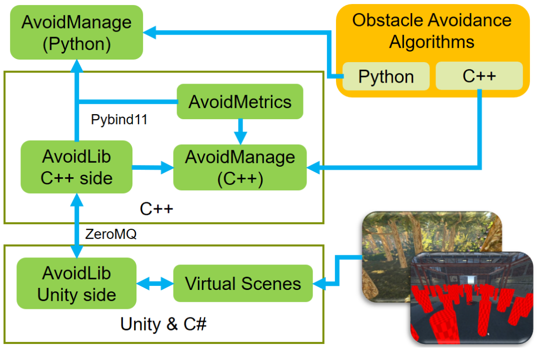

Fig. 2. Different modules of AvoidBench. AvoidLib is used to communicated between the Unity side and C++ side. AvoidManage contains the setup of virtual scenes and the whole process of flight and benchmarking. AvoidMetrics is the package of performance and environment metrics.

A. Virtual Scenes

The scene in AvoidBench is defined as an editable environment in which the obstacle avoidance algorithms are benchmarked. Due to the limited interface with the environment in Flightmare, it is difficult to meet our benchmarking requirements, such as adding terrain or bushes in the forest and changing the texture of obstacles in the indoor scene. Due to time restrictions, AvoidBench for now has two different environments: an outdoor scene and an indoor scene. As shown in Figure 1, the outdoor scene is a forest with the size of $160\,\mathrm{m}\times160\,\mathrm{m}$ which is composed of red trees, bush trees and terrain, while the indoor scene is a warehouse with geometrical but differently textured obstacles. Here are some key concepts related to the setting of environments.

1) Map: The benefit of a simulator is that it is easy to create and auto-generate different maps for the flight tests. Users can easily change the number or position of outdoor trees as well as the size and opacity of texture on indoor obstacles. To evaluate the obstacle avoidance ability of drones, it is necessary to follow certain rules to generate obstacles. To generate the actual obstacle field, AvoidBench utilizes a rudimentary and well-known method for procedural generation named Poisson Disc Sampling [15]. This method has a considerable advantage over random sampling as it creates a uniform distribution of obstacles. Furthermore, it allows us to control the minimum distance between obstacles.

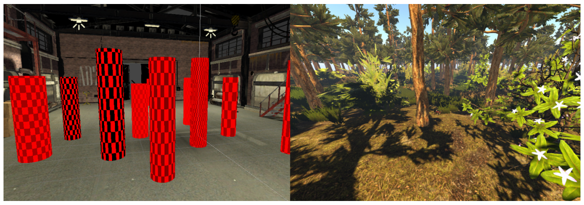

Fig. 3. Indoor and outdoor scenes. There are many geometrical obstacles but with different opacity of texture in indoor scene (left). Outdoor scene contains red trees and non-geometrically-shaped obstacles such as the bushes (right).

As shown in Figure 3, for the indoor scene, we use cylinders as the obstacles. Users can change the size and opacity of the texture on each cylinder as well as the radius of Poisson distribution[1]. To get a new map we just need to set a new random seed. For the outdoor scene, we want to test the effectiveness of algorithms for nongeometrical obstacles, we set both red trees and bushes on the undulating terrain, users can also set the ratio of red trees to bush trees.

2) Trial: Given a general map, we perform different *trials*. Each trial has a different start point and goal point generated by a random seed. The drone can traverse the whole map as much as possible. We use the mesh collider from Unity3D to check if the drone collided with obstacles. If the algorithm can command the drone to the goal within one meter without collision, this trial is marked as "*finished*". Apart from collisions, there can be a variety of situations that lead to trial failure. An algorithm e.g., might get stuck in a dead end, unable to find or move to the goal. By setting a maximum time parameter, AvoidBench automatically stops the trial if the time limit has been exceeded.

B. AvoidLib

Like Flightmare, we also use ZeroMQ to build the communication interface between Unity and C++. ZeroMQ is an asynchronous messaging library, aimed at use in distributed or concurrent applications. Through AvoidLib, we can transfer the parameters of the map to Unity, update the pose of the drone in rendering engine, obtain the images collected by stereo cameras and get the collision detection results from Unity3D. Since the environment metrics which will be introduced below require an accurate point cloud map, we retain the function of saving point clouds in Flightmare.

C. AvoidManage

AvoidManage is used to combine the rendering engine with a simulated dynamics model of multi-rotors. As the management of whole benchmark, AvoidManage is also the state machine of the simulated drone. Firstly, it needs to get the control commands from an obstacle avoidance algorithm and obtain the flight data in real time. Then the flight data will be sent to AvoidMetrics to evaluate the performance of each *trial* and determine if the *trial* should be terminated according to the feedback of collision checking. Furthermore, we also use Pybind11 to wrap the AvoidLib and provide a Python version of AvoidManage for researchers that want to program their obstacle avoidance algorithms in Python.

D. AvoidMetrics

AvoidBench aims to capture the relation between avoidance performance and the environmental conditions. By measuring both performance and environment metrics, it becomes possible to relate the performance of an algorithm to the type of environment and its corresponding difficulty. The grand goal here is that based on extensive benchmarking results in simulation, we will be able to accurately predict an algorithm's performance in real-world environments.

1) Environment Metrics: AvoidBench currently measures two different environment metrics, named *Traversability* and *Relative Gap Size*.

a) Traversability: Traversability was introduced in [16] as a metric to quantify how difficult it would be for a drone to traverse an environment. It was presented as an alternative to obstacle density which had the issue that it did not accurately capture avoidance difficulty as the dimensions of the drone were not taken into account. Moreover, traversability does not depend on a specific obstacle geometry. It can be calculated in generic environments by picking N random position p in the environment. For each position, a random heading h would be selected, and a collision checker is used to calculate the free-flight distance s. All the results are then averaged to obtain the uncorrected traversability. The final step to make the metric dimensionless was to divide the result by the drone diameter d_{drone}.

$$TRAV = \frac{1}{d_{drone} \cdot N} \sum_{i=0}^{N} s(p(i), h(i)) \tag{1}$$

In AvoidBench, we apply some changes to the original traversability metric to make the results more repeatable. Intuitively, the traversability represents how far the drone can travel without an active obstacle avoidance capability. Specifically, we use a grid-based sampling method instead of a uniformly random to sample locations at different heights of the 3D map. Additionally, at each location, instead of randomly picking a single direction vectors are generated evenly across a 2D circle. For each direction vector, the free-path length is calculated, and all results are averaged to obtain the traversability. Traversability is based on sampling points, so that it can also be easily applied to more complex environments than those with cylindrical obstacles.

b) Relative Gap Size: The second environment metric, *relative gap size*, is less generic, as it is directly related to the radius of Poisson Distribution. This value represents the minimum space between obstacles expressed in multiples of the drone's diameter. It adjusts Poisson radius $r_{poisson}$ by subtracting the average width $\hat{w}_{obstacle}$ and divides it by the drone diameter d_{drone}:

$$RGS = \frac{r_{poisson} - \hat{w}_{obstacle}}{d_{drone}} \tag{2}$$

The relative gap size should never be below a value of 1. In that case, the drone would be unable to traverse through the obstacle field as the obstacles are positioned too close together.

---

[1] Custom shape and texture properties will also be added soon.

2) Performance Metrics: AvoidBench currently measures seven different types of performance metrics, which will be introduced as following.

a) Success Rate: This metric is the most straightforward result of AvoidBench. Each map has several trials, according to the ratio of the number of "*finished*" trials $T_{finished}$ to the whole number of trials T_{whole}, we can see the basic performance of this algorithm in the current map for a pre-selected start-to-target distance. The *success rate* can be calculated as:

$$SR = \frac{T_{finished}}{T_{whole}} \tag{3}$$

b) Path Excess Factor: The *path excess factor* measures how optimal the drone's path is from the start point to the goal point. It can be calculated using:

$$PEF = \frac{d_{trav} - d_{min}}{d_{min}} \times 100\,\% \tag{4}$$

Where d_{trav} means the total distance travelled by the drone and d_{min} is the minimum free-path distance from start point to end point calculated by a path searching algorithm (A* [17] in our case). The path excess factor should be interpreted as the excess distance covered by the drone relative to the shortest path. So, $0\,\%$ means no excess distance, whereas a $100\,\%$ means that the drone covered twice the distance it could have.

c) Energy Cost: We use the integral of jerk (derivation of acceleration) to express the *energy cost*. In real world, the lower of energy cost means the longer drones can fly.

$$EC = \int_0^{t_{trial}} \|jerk\|_2 \, \mathrm{d}t \tag{5}$$

d) Average Goal Velocity: The purpose of the *average goal velocity* metric is to measure the speed with which an algorithm is able to traverse the obstacle field towards the goal point. Usually, the trial time it takes for the drone to travel from the start point to the goal point is used to evaluate the speed performance. However, to ensure that results are also comparable between different maps and path lengths, the trial time is converted to a velocity value using the free-path minimum distance, as shown in (4). Here, t_{trial} is defined as the trial time and the average goal velocity can be calculated as:

$$AGV = \frac{d_{min}}{t_{trial}} \tag{6}$$

e) Relative End Distance: The *relative end distance* metric measures how far the drone has progressed to the goal point, which is most useful for comparing runs when the drone does not reach it. In AvoidBench, relative end distance is calculated based on the percentage of two vector norms. Given vector **a** as the start-goal vector, **b** as the vector from start point to the drone's final location shown as Figure 4, the relative end distance can be calculated by:

$$RED = \frac{|\mathbf{a} - \mathbf{b}|}{|\mathbf{a}|} \tag{7}$$

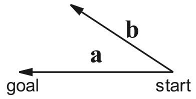

Fig. 4. Calculation method of relative end distance. **a** is the vector of start point to goal point. **b** is the vector of start point to drone's final location.

f) Processing Time: The last performance metric, *processing time*, directly measures how long a single iteration of the obstacle avoidance algorithm takes. During the mission, all processing periods are stored, which allows AvoidBench to calculate statistics such as the mean and standard deviation of these values. The processing time is machine-specific, and hence algorithms would have to run on the same machine for a fair comparison. Processing time influences the control delay and as such already influences other performance metrics, such as success rate. However, it is also of interest by itself, as a lower processing time means that more computational resources are available for other autonomy tasks.

g) Contrast Factor: The *contrast factor* is the only metric that compares two algorithms. It can not only be used to measure the performance ratio of two different algorithms without the influence of flying distance, but also the performance differences of the same obstacle avoidance algorithm in the simulator and real world.

From the perspective of probability, the success rate is related to the tested algorithms, the minimum free-path distance from start point to goal point, and the complexity of maps which can be represented by traversability. We use Ξ to represent the obstacle avoidance algorithm and τ as a certain map. The normalized traversability P_τ can be calculated as:

$$P_\tau = \frac{TRAV}{TRAV_{max}} \tag{8}$$

Where $TRAV_{max}$ is the value of traversaility when there is no obstacle in the map (only boundaries of the map have an impact on this value). Assuming the obstacles in the map are uniform, the probability of no collision for a certain algorithm within a certain distance is p, and the probability of no collision at twice distance should be p^2. Obviously, the success rate is exponentially related to the flying distance in theory. Then the success rate $\hat{SR}$ can be estimated as:

$$\hat{SR}_{\Xi} = f\left(\Xi, P_\tau\right)^{\frac{{}^{\Xi}d_{min}}{\lambda}} \tag{9}$$

Where λ is a scale factor of distance, and ${}^{\Xi}d_{min}$ is the minimum free-path distance from start point to goal point randomly generated for algorithm Ξ. $f\left(\Xi, P_\tau\right)$ indicates the non-collision probability for a certain distance determined by λ when algorithm Ξ is excuted in the map with normalized traversability P_τ. From AvoidBench, ${}^{\Xi}d_{min}$ and P_τ are known, and $\hat{SR}_{\Xi}$ can be replaced by statistical results from (3). However, we still cannot get $f\left(\Xi, P_\tau\right)$ because the λ is unknown. By logarithmizing both sides of (9), we can obtain the ratio of different algorithms' performance in the map with same traversability:

$$CF = \frac{\ln f(\Xi_1, P_\tau)}{\ln f(\Xi_2, P_\tau)} = \frac{{}^{\Xi_2}d_{min} \ln SR_{\Xi_1}}{{}^{\Xi_1}d_{min} \ln SR_{\Xi_2}} \tag{10}$$

If Ξ_1 and Ξ_2 represent the experiments of the same algorithm in the simulator and real world, then contrast factor can be used to measure the sim2real gap even though they have different flying distance.

III. SIMULATION AND EXPERIMENT

A. Obstacle Avoidance Algorithms

In order to fully verify the configuration effect of obstacle avoidance algorithms on AvoidBench, we choose several mainstream, state-of-the-art obstacle avoidance algorithms, including: Agile-Autonomy [18] (learning-based), Ego-planner [19] (optimization-based), and MBPlanner [3] (motion-primitive-based). For Agile-Autonomy, we use their original code and retrain the network through our scenes. MBPlanner is a exploration algorithm, we changed the exploration gain to the distance of current point to the goal point so that the drone can always fly to a fixed goal. All these three algorithms are tested by a virtual stereo camera which can generate depth map through the Semi-Global Matching algorithm [20].

B. Simulation Results

For the outdoor scene, we set 30 maps and 30 trials. The ratio of bush trees to red trees is 2:3. The radius of the Poisson distribution is within the interval [2.3, 5.8]. Start points and end points are all randomly selected according to a random seed so that these three algorithms will encounter the same environments and tasks. All simulation experiments are run on a standard laptop with 16GB memory, i7 11800H CPU, and RTX3050 GPU. The frame rates of images obtained in outdoor and indoor environments are 20Hz and 40Hz respectively.

Figure 5(a) shows the distribution of the success rate of the three algorithms with the increase of traversability. we divide the data into five groups so that it is easy to see the tendency. Obviously, for the three algorithms, success rate increases significantly with the increase of traversability, which shows that the performance of obstacle avoidance algorithms has a great relationship with the environment metrics we proposed. Figure 5(a) indicates that Ego-planner and MBPlanner have a similar performance concerning safety (success rate) and that both are better than Agile-Autonomy. Figure 5(b) is the result of contrast factor calculated by success rate. The value of "Ego / MBPlanner" is closed to 1, meaning that they have a similar performance while "Ego / Agile-Autonomy" is below 1, meaning that Ego-planner has a higher success rate than Agile-Autonomy. The relationship between traversability and relative gap size is shown in Figure 5(c). The traversability of outdoor scene does not always increase linearly according to relative gap size as the indoor scene because of nongeometrical bush trees. Basically, traversability is a more general metric to evaluate the complexity of environments.

Figure 6(a) and 6(b) show the result of path excess factor and energy cost respectively for both lower is better. Mission number has been sorted by traversability. The lower of path excess factor means drone's trajectory is closer to the minimum distance, and the lower of energy cost means the drone cost less energy for a same trial. Basically, both Ego-planner and MBPlanner show a downtrend at the metrics of path excess factor and energy cost following the increase of traversability. Moreover, we can see Agile-Autonomy has the best performance in this metric, with a less clear relation to traversability.

Another metric with significant differences is average goal velocity (Figure 6(c)). Agile-Autonomy has been designed for high-speed flight, which results in higher flight speeds than those of Ego-planner and MBPlanner. Relative end distance can also indicate the safety of obstacle avoidance. As shown in Figure 6(d), in the results of Relative end distance, Ego-planner is still the most safe algorithm and we can see relative end distance also has a decreasing trend with traversability.

For the result of processing time (Figure 6(e) and 6(f)), Ego-planner and Agile-autonomy both have the best performance. Although Ego-planner has a process of mapping, it avoids the construction of the ESDF map, thus greatly reducing the processing time. Agile-Autonomy uses an end-to-end neural network, it does not need to build a map, so the running speed is also fast. The redundant map construction of MBPlanner makes its processing time the longest among the three algorithms.

C. Real World Tests

1) Experiment Platform: To verify that our proposed metrics can objectively evaluate different algorithms in the real world, we also built a real drone for experimental verification as shown in Figure 7(a). A 6inch frame from Armattan is used for the drone which equipped with Emax 2306 motors and 5inch, three-bladed propellers.

The platform's computational unit is an NVIDIA Jetson Xavier NX module, with 384-core GPU, 48 Tensor Cores, and 6-core ARM CPU. The output of this framework is a low-level control command including ratio of collective thrust and bodyrates to be achieved for flying. The desired commands are sent to a flight controller named Pixhawk mini 4 which running PX4. The drone is also equipped with a stereo camera, Oak-D which can get a dense depth in 400p resolution at 30 Hz.

2) Experiment Results: For real world experiments, we arrange three maps of different complexity. To calculate the traversability of real world, we use a lidar sensor to build high-precision point cloud maps, one of the map is shown as Figure 7(b). And for each map, we run 20 trials. From (8), the normalized traversability can be obtained, then we find the experimental results of simulation in the corresponding interval according to the normalized traversability of real world, and the results are shown as Figure 8.

From Figure 8(a), we can see at similar normalized traversability, simulation results are always better than those

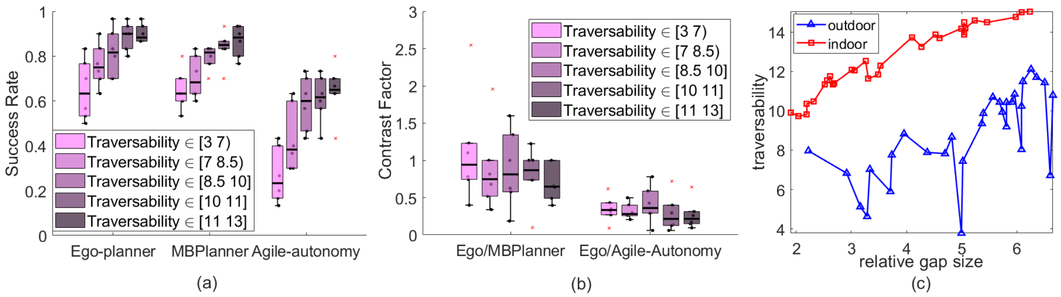

Fig. 5. (a) is the success rate of different algorithms. (b) is the contrast factors of Ego-planner to MBPlanner and Ego-planner to Agile-Autonomy. (c) is the relationship between traversability and relative gap size of both indoor scene and outdoor scene.

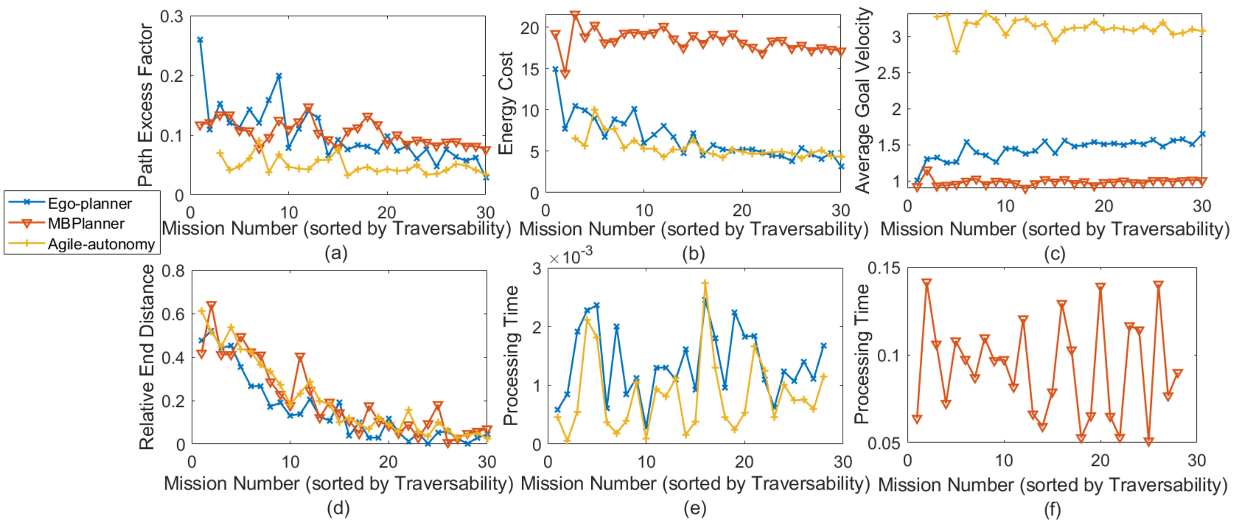

Fig. 6. Results of different metrics relating to the increasing traversability missions. (a) shows the results of path excess factor. (b) shows the results of path cost. (c) shows the results of average goal velocity. (d) shows the results of relative end distance. (e)-(f) show the results of processing time.

of real world. And we also calculate the contrast factor of ”Simulation / Real world” as shown in Figure 8(b). From this metric, the influence of distance can be removed, so the results are closed to 1 except the easiest map (This may be caused by the limitation of the number of experiments). And we also show the results of path excess factor as shown in Figure 8(c), obviously, this metric has a decreasing trend with traversability as we mentioned in simulation part.

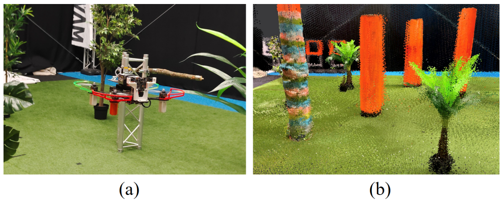

Fig. 7. (a): Experiment platform for real world tests. (b): High-precision point cloud map built by a lidar sensor.

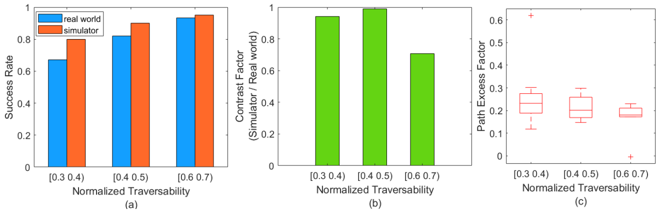

Fig. 8. Comparison of real world tests and simulation tests. (a) shows the results of success rate when real world maps have similar traversability with simulator. (b) shows the results of contrast factor of ”simulator / real world”. (c) shows the results of path excess factor in real world tests.

IV. CONCLUSIONS

In this work, we proposed AvoidBench which can evaluate the performance of obstacle avoidance algorithms. In AvoidBench, we can easily set many maps with different obstacles. We combined both performance and environment metrics to explore the performance of different algorithms with increasing environment complexity. Both of the simulation and real world experiments indicate our proposed metrics are reasonable. And the comparison with the real world experiment data also shows that AvoidBench can realize high-fidelity simulation experiments.

REFERENCES

[1] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, “Graph-based subterranean exploration path planning using

aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020.

[2] M. Kulkarni, M. Dharmadhikari, M. Tranzatto, S. Zimmermann, V. Reijgwart, P. De Petris, H. Nguyen, N. Khedekar, C. Papachristos, L. Ott *et al.*, "Autonomous teamed exploration of subterranean environments using legged and aerial robots," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 3306–3313.

[3] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis, "Motion primitives-based agile exploration path planning for aerial robotics," in *IEEE International Conference on Robotics and Automation (ICRA), Paris, France*, 2020.

[4] R. Mlambo, I. H. Woodhouse, F. Gerard, and K. Anderson, "Structure from motion (sfm) photogrammetry with drone data: A low cost method for monitoring greenhouse gas emissions from forests in developing countries," *Forests*, vol. 8, no. 3, p. 68, 2017.

[5] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving," in *The KITTI Vision Benchmark Suite, 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3354–3361.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[7] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[8] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. v. Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *International conference on simulation, modeling, and programming for autonomous robots*. Springer, 2012, pp. 400–411.

[9] 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.

[10] 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.

[11] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," *arXiv preprint arXiv:2009.00563*, 2020.

[12] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. Reddi, "Mavbench: Micro aerial vehicle benchmarking," in *2018 51st annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 2018, pp. 894–907.

[13] T. T. Du Montcel, A. Nègre, J.-E. Gomez-Balderas, and N. Marchand, "Boarr: A benchmark for quadrotor obstacle avoidance based on ros and rotors," 2019.

[14] R. Veder and G. de Croon, "Avoidbench," http://resolver.tudelft.nl/uuid:302426ff-a4a4-4f8a-967a-331ea71b1ba1.

[15] R. Bridson, "Fast poisson disk sampling in arbitrary dimensions." *SIGGRAPH sketches*, vol. 10, no. 1, p. 1, 2007.

[16] C. Nous, R. Meertens, C. De Wagter, and G. De Croon, "Performance evaluation in obstacle avoidance," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3614–3619.

[17] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/tssc.1968.300136

[18] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," in *Science Robotics*, October 2021.

[19] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2020.

[20] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2007.