2016 12th European Dependable Computing Conference

Virtual Worlds for Testing Robot Navigation: a Study on the Difficulty Level

Thierry Sotiropoulos, Jérémie Guiochet, Félix Ingrand and Hélène Waeselynck
LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France
Email: firstname.lastname@laas.fr

***Abstract*—The ability to navigate in diverse and previously unknown environments is a critical service of autonomous robots. We propose a test framework based on MORSE (Modular Open Robots Simulation Engine), and using the generation of virtual 3D worlds to challenge the navigation service. We elaborate on the notion of the difficulty of the generated worlds, which we characterize in terms of mission achievement, mission duration and trajectory curves. We experimentally study our ability to control the difficulty level by means of the generation parameters. We also assess the indeterminism of the navigation and how it evolves depending on the difficulty level. The experimental outcomes provide insights toward the definition of test strategies to further stress the navigation service.**

I. Introduction

Risky of testing mobile robots in unstructured / human shared / diverse / unknown env.

The deployment of mobile robots in unstructured and human shared environments raises safety, performance and mission achievement issues. Indeed failures of such systems may lead to unacceptable consequences. Among dependability techniques, testing in real world is a way to reveal and remove faults. But real world tests of mobile robots are very expensive, can explore a limited number of operational situations and are not without risk (for the hardware, the environment or the humans).

Adv. of simulation-based testing

Another way to test these systems can be simulation-based testing. It is potentially more complete compared to a real world testing campaign, and may avoid catastrophic consequences. Moreover, the tests may immerse the robots in many virtual worlds to explore a large number of situations.

The goal is to define a systematic simulation-based testing approach

Although existing simulators have reached sufficient maturity to allow testing, systematic simulation-based testing approaches are still to be defined. The long-term goal of our work is to propose such an approach.

Object testing: the navigation service
Framework: MORSE
Virtual worlds: PGA

As a case study, we focus on testing a critical service of mobile robots: the navigation service. Our testing framework is based on MORSE (Modular Open Robots Simulation Engine), which uses the 3D game engine of Blender. Virtual worlds are generated using a procedural generation approach.

In this paper, we propose a notion of the difficulty of the generated worlds, in terms of the feasibility of navigation missions and the effort required for their successful completion. Predicting difficulty *a priori* is very complex, so we observe it *a posteriori* (after running the tests). Based on the observed mission success rate, mission duration and trajectory curves, we classify worlds on a scale ranging from easy to very difficult. We then experimentally study our ability to control the difficulty level by means of the 3D world generation parameters. We also assess the indeterminism of the navigation and how it evolves depending on the difficulty level.

The paper is organized as follows. Section II presents related work: some previous experiments on autonomous systems, existing simulators and procedural generation techniques. In Section III, we present the system under test. Section IV presents the testing framework we developed based on MORSE. We used this framework to address a set of experimental questions presented in Section V, pertaining to the notion of difficulty. The results are presented and discussed in section VI. We then conclude.

II. Background

Simulation-based testing permits to avoid hazardous consequences, while providing means for controlling the inputs and for performing complex output analysis. Some robot simulators are specialized for a particular system or component, like iCub [1] or HRP [2]. Others can simulate various types of robots in various environments, as it is the case for MORSE [3], Gazebo [4] or Webots [5]. The survey in [6] compares several simulators including MORSE and Gazebo. A *Software-in-the-loop* philosophy is present in most of the simulators mentioned above. It allows the execution of real control software using a model that simulates its environment.

Recent work has studied test selection strategies for autonomous systems. Most of the proposed strategies are based on an abstract model of test situations [7], describing the involved entities, their relationships and some interaction patterns. The authors of [8] use UML (Unified Modeling Language) to specify a metamodel of entities and a set of interaction scenarios. The approach is applied to a vacuum cleaner robot, using metaheuristic search techniques to generate abstract test data from the models. In [9], the structural model of entities is in UML and the dynamic part consists of Petri nets. The work of [10] defines several types of mid-air collision situations, and uses them to guide the evolutionary testing of a drone collision avoidance algorithm.

Limitation of recent studies

As a general comment, the vast majority of developed approaches adopt a simplified view of the simulated environment. To the best of our knowledge, the work of [11] is the only one to consider complete virtual world environments. In order to test a path-following navigation algorithm for ground robots, the authors randomly generate virtual worlds consisting of 2D

978-1-5090-1582-5/16 $31.00 © 2016 IEEE
DOI 10.1109/EDCC.2016.14

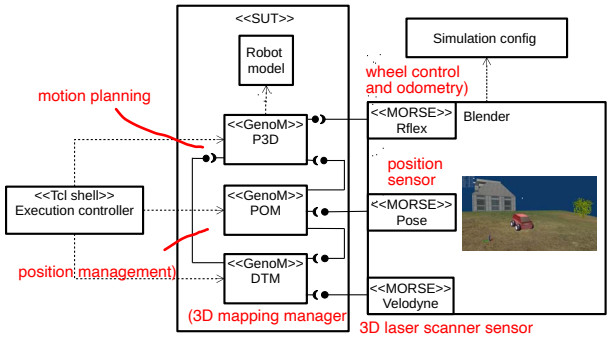


Fig. 1: Simplified diagram of the test architecture

maps filled with fixed and mobile obstacles. An interesting contribution of this work is to establish a connection with world content generation techniques used in the domain of video games.

How does PCG work?

In the video game community, generating worlds is well studied and developed. *Procedural content generation* (PCG) is an algorithmic approach such that, from a few selected parameters, a large number of possible game contents can be automatically created [12]. PCG is widely used to generate maps, dialogues, missions, characters, or more decorative aspects such as textures. The main idea is to have some upstream parameters – compact description of the content (or genotype) – describing the content that will be generated downstream – extended description of the content (or phenotype). This generation can take place offline or online, i.e. during development or while playing the game. In [12], the authors distinguish between constructive and generate-and-test techniques. The constructive techniques generate the content once, while the other ones have iterations at which candidate contents are produced and evaluated. For example, generate-and-test algorithms have been used to produce maps for the *Dwarf Fortress* game, and also for a game based on *Super Mario Bros* (in which case the evaluation function included learned information about the player [13]). Another approach in [14] uses an evolutionary approach to optimize a population of maps for a real time strategy game, based on a multi-objective fitness function.

In conclusion, simulation-based testing of autonomous systems is an emerging topic that has started to attract interest from the testing community. Previous work has suggested the application of PCG techniques to generate the test situations, which we believe is a promising direction of research. This paper presents a testing framework based on the MORSE simulator and our first experimental results on the generation of virtual worlds.

III. System under test

Object để test

In this study we use MORSE, a robotic simulator built upon Blender.It includes a physical engine (*Bullet Engine*) and provides numerous plugins for popular robotic platforms, sensors and effectors. Blender also provides numerous useful functions for automatic generation of worlds. We focus here on the *Mana* robot (a Segway RMP 400 platform) and the Velodyne lidar sensor, which are both handled by MORSE.

The Mana robot software is organised according to a classical robotic architecture [15], but we focus here on functional layer and more precisely the navigation service. Six functional modules are involved in this service. As can be seen in Figure 1, Rflex (wheel control and odometry), Velodyne (3D laser scanner sensor) and Pose (position sensor) are all handled by MORSE. DTM (3D mapping manager), POM (position management) and P3D (motion planning) are the exact replicas of the modules running on the real robot. Together, these three modules represent the system under test (SUT) for a total number of 35k lines of code. They are developed with *GenoM* (Generator of Module) [16]. *GenoM* is a tool that helps developers to deploy and encapsulate the needed algorithms into standardized server components. *GenoM* modules communicate via the pocolibs middleware, using shared memory primitives called posters.

In robotics, motion planning has been extensively studied. The tested P3D module implements the local planning algorithm presented in [17]. Its principle is to choose the path that minimizes both a traversability-stability cost and the distance to the goal. The algorithm considers a fixed number of arc-shaped paths in front of the robot, and different points (called nodes) along each arc. The cost of putting the robot at a particular point increases with the slope at this location. Cost is infinite if the terrain is unknown (no perception). Perception problems may come from the laser sensor range or from terrain irregularities, as shown in the right side of Figure 2: the white areas correspond to patches of the map that have not been perceived by the laser. P3D computes the cost and reward associated with each arc and selects the best one.

thay thế cho In fact, To elaborate

In our test cases, P3D performs its search over 20 arcs with 15 nodes each. It fails when no arc is elected, i.e., all the costs are above a given threshold. Without going into the details of the P3D navigation, one should note that this is a local method, so it is expected to fail if the robot ends up in a dead end (or inside a small U shape area). In such a situation, a global navigation such as $D*$ [18] could then be used to pursue the exploration of the "unknown" environment and find a path to the destination point.

How the study will come - motion planning / P3D

It is important to note that even if one starts two navigations with the same initial setup (world and mission), the observed trajectories may be different. Indeterminism is inherent to the perception/action loop. It occurs on real robots as well as in simulation.

IV. Testing framework

MORSE typically serves prototyping purposes, but we consider its use for conducting more extensive test campaigns. It requires us to set up a richer testing framework, allowing us to automate the generation of the navigation scenarios, their execution, the collection of output data sets and their analysis.

Fig. 2: Left: the robot in the simulation environment MORSE. Right: DTM environment representation

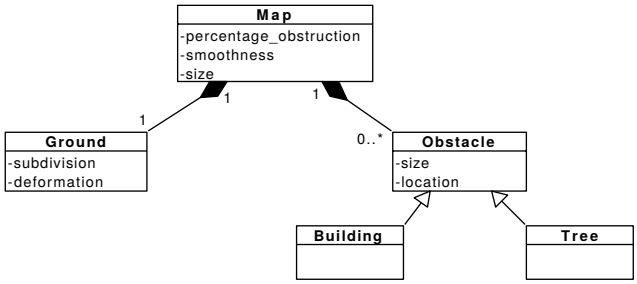


Fig. 3: World model derived from the use case

A. *Virtual worlds as test inputs*

Navigation Scenarios

When designing test experiments, a key element is the world model underlying the navigation scenarios. It is obtained by studying the specification of the robot (marine, aerial or terrestrial) and its typical use cases. The study must identify relevant environment characteristics as well as constraints that eliminate impossible worlds. Once the world model is built, we investigate solutions to automatically produce world instances. The solutions may combine world content generation functions and selection from a library of predefined elements.

Robot developers gave us a 3D image of an area where the physical robot was deployed for experimentation in real conditions. This use case helped us to identify the types of obstacles and terrain that the robot can potentially encounter. We derived the world model represented in Figure 3, composed of the ground and obstructing objects like trees and buildings. The ground itself does not have a rugged topology with mountains or canyons. It is rather smooth but with many local irregularities. We model these irregularities by local deformations applied to subdivisions of the ground. Overall, a map is characterized by its size, its percentage of obstruction (due to objects), and its degree of smoothness (resulting from the ground local deformations). Some constraints are added to the model: objects must have a location inside the map, they are laid on the ground and cannot overlap each other.

Robot Navigation Testing

The inputs to robot navigation testing are a world instance and a navigation mission in this world, defined by a starting position and a target arrival position. The mission adds constraints: both positions must be at a location free of obstacles. Moreover, the free area around the starting position must be large enough for Mana to properly initialize. The robot first moves some few meters from its initial location to scan its surroundings and to build an initial map of the perceived environment. The size of the free area around the starting position was determined after discussion with the developers of the navigation software. We took the same size for the free area around the arrival point.

To instantiate the world model, the generation of the ground and obstacles may uses different solutions: predefined functions of Blender, predefined objects or basic objects.

For the topology, we choose the Blender `Subdivide` function. It subdivides the faces of objects (called *mesh* in the Blender terminology), to obtain a grid controlling the mesh geometry. The `Fractal` transformation then displaces the grid vertices in random directions. Figure 4 shows the result of a transformation applied to a flat plane mesh representing the ground. We choose to constrain the displacements along the normal z-axis to obtain a less chaotic and more realistic terrain. Control parameters: *s*, the number of subdivisions, and *d*, the amplitude of the grid deformation; allow us to obtain from smooth to rough terrain.

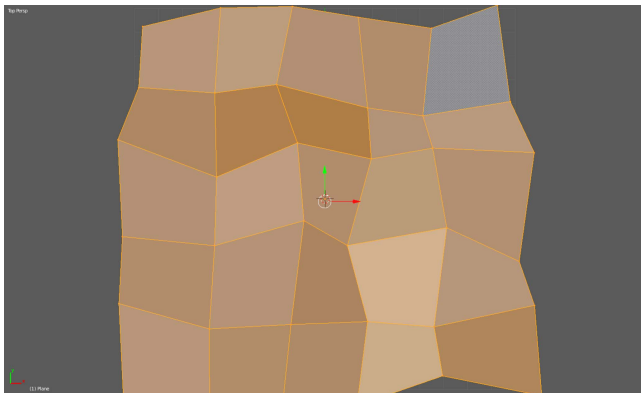


Fig. 4: The resulting topography of the subdivision algorithm with $s = 4$ and $d = 1$

For buildings, we decided that it was not worth using a convoluted function. The robot lidar only perceive the general shape of objects, and so a building can merely be represented as a big rectangular box mesh, which can be easily generated from Blender. For a first approximation, we also use cubes to simulate trees.

These solutions, attached to each basic element of the world model, provide convenient building blocks of generation profiles to produce world instances.

B. *Data collection and analysis*

During a simulation run, we collect events and data traces that are relevant to analyse the navigation behaviour. Some of them are taken from the robot's point of view, including its perception of the environment and the decisions it took, while others are the view of an external observer sensing what really happened in the simulated world instance.

The readouts for the robot's point of view are the following:

- Perceived positions during the run, captured each time the corresponding poster is updated by POM. Each position is timestamped and includes yaw, pitch, roll, x, y and z,

- Perceived map at the end of the run, corresponding to the last value of the poster updated by DTM,
- Error messages produced by all the GenoM modules.

The interface of GenoM modules makes it possible to configure them to log the values of posters each time they are updated, as well as to log them on demand. Should we test alternative navigation algorithms the logged data would still be meaningful.

The readouts for the external observer's point of view are the following:

- Real positions of the robot during the run (in the same format as the perceived ones),
- Collision events,
- Timeout events, used to abort the run if it does not end after a certain amount of time (e.g. because the robot failed to approach toward the goal).

All these readouts are raw data that must be processed to produce synthetic outcomes. Our testing framework currently provides a number of data processing and data visualization facilities.

Each simulation run is classified according to the success or failure of the mission. In case the run ended normally (no time-out, no error from GenoM modules), there was no collision, and the last real robot position was at the target destination within some tolerance, the mission is considered a *success*. All other cases fall into a mission failure category, which can be *Fail-Collision*, *Fail-TimeOut*, *Fail-Error* or *Fail-Other*. Runs exhibiting multiple issues (e.g., both a collision and an error message) are classified into the most severe category (e.g., *Fail-Collision*). The distance between the end position of the robot and the target destination point is also systematically reported. Note that a failed mission does not necessarily mean that the navigation is faulty. While collisions obviously reveal faults, *Fail-Error* may be an expected behaviour if obstacles make it impossible for the robot to reach its target. It is then necessary to have a closer look at the trajectory adopted by the robot.

A script has been developed by colleagues at LAAS to draw trajectories on the 3D image of the world instance. It allows us to visualize a real trajectory during one run, compare it with the perceived trajectory, or visualize different trajectories taken in multiple similar runs due to the indeterminism as one can see in Figure 5. Finally, the perceived map at the end of the run may be visualized. It may be helpful to understand the reasons for a failed mission, in particular in cases where an error message reported that no arc for navigation could be selected.

V. Design of experiments

The map model in Figure 3 can later be enriched by considering dynamic elements, like mobile obstacles (e.g., pedestrians, bikes, cars), noise applied to the robot sensors, etc. But before considering these extensions, it is interesting to gain deeper insights into how the static characteristics of the map may stress the navigation. We use the previously described testing framework to experimentally address three questions.

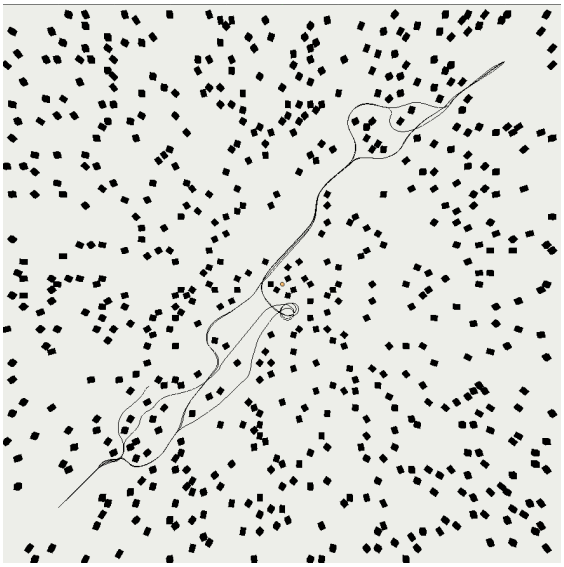


Fig. 5: Five paths on the same ground with 6% of obstruction

The first one concerns controllability of the difficulty of a generated map. The candidate control parameters are the smoothness degree and percentage of obstruction: these are inputs to the generation process. The difficulty of the resulting maps is determined *a posteriori*, from the observed success rate of navigation missions, the time to get to the destination point and the complexity of the trajectory (detours). We aggregate these outcomes to obtain a classification into three levels of difficulty: easy, challenging and very difficult. Ideally, we would like to find a close correspondence between ranges of generation parameters and the supplied difficulty levels.

Q1 - To what extent can the difficulty level of the navigation mission be identified and controlled by the value of generation parameters?

The second question is related to the indeterminism of the navigation. Given one map and one mission in this map, different runs might yield drastically different results. This would impact the applicability of iterative test generation techniques. For example, consider a search-based generation process, in which the fitness of a map is calculated based on the result of a single run at each iteration. Or consider the incremental production of neighbouring maps, in which obstacles are gradually added on the trajectory of the robot. A high degree of indeterminism would compromise such techniques. We conjecture that the degree of indeterminism might depend on the difficulty level.

Q2 - To what extent does the navigation exhibit indeterminism, and how does indeterminism evolve depending on the difficulty level?

The third question concerns the use of difficulty levels in a test strategy. It may be desirable to generate maps from all levels of difficulty, to exercise the navigation service in

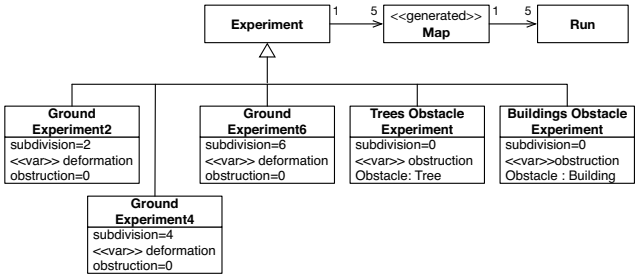


Fig. 6: Classes of experiments

diverse ways. Or one may consider that the test process should not spend much time exploring easy cases, hence favouring the challenging and very difficult levels. Conversely, one may avoid the very difficult cases, considering it useless to assign infeasible missions to the robot. The issue is open, and will require extensive empirical investigation. As a preliminary step, we experiment with a faulty version of the navigation that may exhibit collisions.

Q3 - For the faulty navigation service, how does the fault revealing power evolve with the difficulty level?

To address these questions, we consider navigation runs in maps of size 100 m x 100 m. The starting and destination points are fixed, respectively in the bottom left and the top right areas of the map: a straight trajectory in a flat map is then a travel of about 140 m. The experiments consider one control parameter at a time. If we let the smoothness degree vary, no obstacle is put on the map. Conversely, if the percentage of obstruction is the focus, the map is kept flat. There are five classes of experiments as presented Figure 6. The three ground experiments study the smoothness degree, by letting the deformation d vary for a fixed number of subdivisions s = 2, 4 or 6. The two remaining experiments study the obstruction by buildings or trees, considered separately. Obstacles are placed randomly until the desired percentage of obstruction is obtained. Buildings are boxes having a basis of 9m x 9m, while trees have a basis of 1m x 1m. Hence, for a given percentage of obstruction, the map may either contain a few large obstacles or many small ones.

Class of experiments

For each class of experiment, the values of the studied control parameters are sampled with a fixed increment, yielding a set of control configurations. Given a configuration, 5 different maps are generated and then 5 runs are executed for each map, yielding a total of 25 runs per configuration. For example, we produce 5 different maps with small obstacles filling 6% of the surface. Then, robot navigation is launched five times on each map in turn, so that we can observe indeterminism. Note that a run duration is typically in the order of minutes.

Setup runs for each experiments

The data collected during the runs is used to assign a difficulty level to each control configuration (Q1). The assignment depends on: the overall mission success rate for the 25 runs, the median duration of successful runs, and the median "tortuousness" of successful trajectories. The tortuousness is introduced to characterize the trajectory curves: it corresponds to the average absolute angle between successive positions of the robot. Since duration and tortuousness concern successful runs only, a first step consists in extracting the configurations for which zero or few successes are observed. They are assigned the level *very difficult*. In our experiments the threshold is less than 25% of successes. The remaining configurations are partitioned by the kmeans clustering algorithm, with a weight of 0.5 for the success rate, 0.25 for the duration and 0.25 for the tortuousness. The cluster including the flat configuration with no obstacle has the *easy* level, while the other one has the *challenging* one. It is then studied whether the obtained classification matches subdomains of parameter values.

Class of experiment	Total runs	Total time
Buildings	900	48h30min
Trees	570	23h53min
Ground s = 2	375	16h39min
Ground s = 4	300	14h35min
Ground s = 6	200	8h10min

TABLE I: Experimental effort

Indeterminism (Q2) occurs when, for a same map (and a same mission, since the destination point is fixed), the robot takes different paths. It is detected by comparing the timestamped positions for runs on the same map. We derive a synthetic measurement of the maximal Euclidean distance of the positions at any time t. Details of the differences can also be observed manually by using the trajectory visualization facility.

The maps produced for studying the obstruction by buildings and trees are reused in experiments with an intentionally faulty version of the navigation (Q3). The injected fault consists in configuring P3D for a robot that is slightly smaller (0,84m x 0,45m) than the avatar used by the simulator (1.14m x 0.67m): in Figure 1, the Robot model contains an erroneous size. As a result, P3D underestimates the space required by the robot, which may lead to collisions in the simulation. Like previously, five runs are executed for each map. The fault revealing power is assessed by reporting the number of runs exhibiting collisions.

VI. Experimental Results

All the experiments have been performed on a PC with an Intel Core i7-4800MQ CPU at 2.70GHz, and 16 GB of RAM. Table I shows the numbers of runs for each class of experiment and their total duration.

A. Controllability of difficulty levels (Q1)

Figures 7 to 9 show the assignment of difficulty levels to experimental configurations for three classes of experiments: obstruction with trees, obstruction with buildings, and smoothness degree with one of the considered numbers of subdivisions (the one with s = 2). Each figure contains three graphs corresponding to the measurements used in the assignment: the mission success rate, the duration of successful runs and their tortuousness. The graph plotting the success rate also visually indicates the assigned difficulty level, using a symbol ● for the easy configurations, ▲ for the challenging ones and ■ for the very difficult ones. The experiments with buildings (Fig.

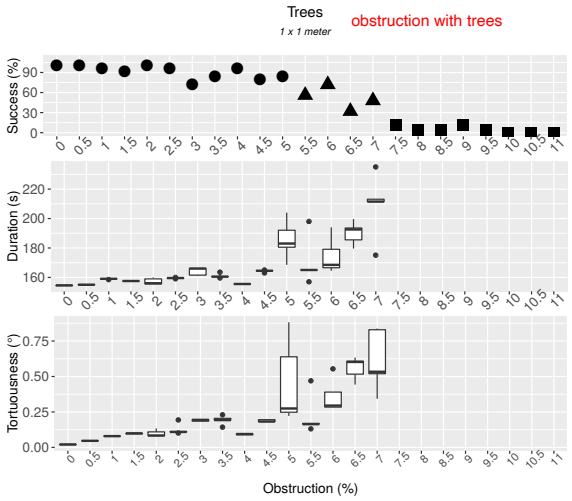
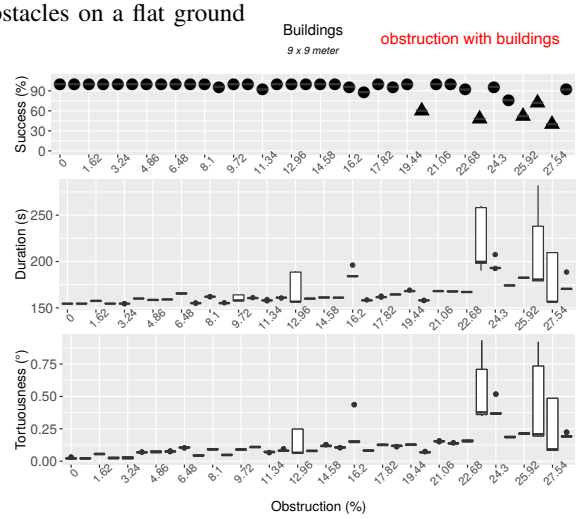


Fig. 7: Success rate, times, and tortuousness results for trees obstacles on a flat ground



(1) did not allow us to observe very difficult configurations

Fig. 8: Success rate, times, and tortuousness results for buildings obstacles on a flat ground

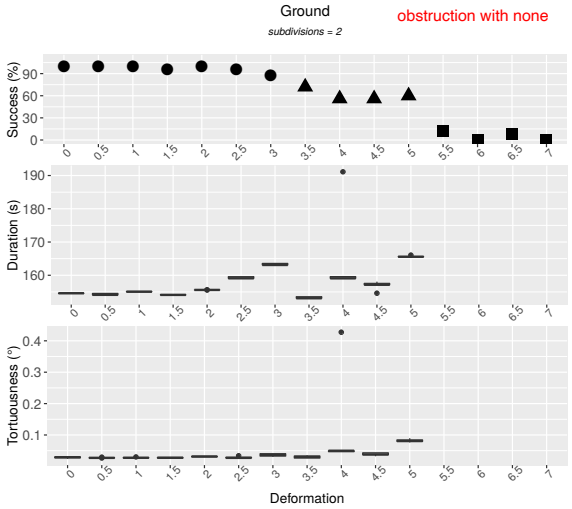


Fig. 9: Success rate, times, and tortuousness results for subdivision $s = 2$ and with no obstacle

(1) 8) did not allow us to observe very difficult configurations. (2) Beyond an obstruction rate of 28.25%, the random placement of buildings failed to find enough free space in the map to add obstacles without overlapping. We had to stop the experiment.

No collision was reported in any of the experiments. The large majority of missions failed because P3D considered that the continuation of the mission was impossible (*Fail-Error*): it stopped the robot and issued an error. All other cases correspond to runs that were forced to abort at the expiration of timeouts (*Fail-TimeOut*). The robot was trapped into a zone surrounded by obstacles or steep slopes, it went round and round in circles without finding an exit and P3D was not able to decide to stop the mission.

Fall error

As a general observation, there is a pretty good correspondence between increasing values of control parameters on the one hand, and increasing levels of difficulty on the other hand. For example, in Figure 7, the configuration is easy up to 5% of obstruction by trees, challenging in 5-7% and very difficult above 7%. The thresholds may however be fuzzy, as exemplified by Figure 8 for buildings. Both easy and challenging configurations can be found above 19% of obstruction by buildings. Fuzzy thresholds are also observed for one class of experiment studying the smoothness degree, the one with $s = 4$: as the deformation value increases, we first observe easy configurations only, then a mix of easy and challenging ones, then a mix of challenging and very difficult ones, and then only very difficult ones. It must be reminded that we considered a limited number of five maps per configurations. Precise threshold values are not significant, but the overall trend of increasing difficulty levels is clear.

It is also significant to compare the relative values of thresholds obtained for similar classes of experiments, respectively for experiments on the obstruction rate and on the smoothness degree. By comparing the results for trees and buildings, it is obvious that the size of obstacles matters and not merely the percentage of obstruction. A map obstructed at 8% by trees will tend to be very difficult, while 8% by buildings will tend to be easy. Intuitively, it is much more stressful to slalom between many small obstacles, than to negotiate a few large ones. Similarly, the comparison of results on the smoothness degree shows that the size of the deformed ground surfaces matters. The smaller the size (i.e., the larger the number of subdivisions of the ground), the more stressful the deformation. In Figure 9 displaying results for 2 subdivisions, the very difficult configurations start at deformation $d = 5.5$. They start earlier at $d = 3.5$ for 4 subdivisions, and as early as at $d = 2.5$ for 6 subdivisions.

To conclude, the macroscopic parameters of a map are relevant from the perspective of a coarse control of the difficulty levels. But the difficulty introduced by obstacles strongly depends on their size. As a map may contain several types of obstacles (e.g., both trees and buildings) or obstacles of varying size in a certain envelope (e.g., buildings which are not all identical), the generation parameters should not merely consider the desired percentage of obstruction. They should also include weights to control the generation of alternative

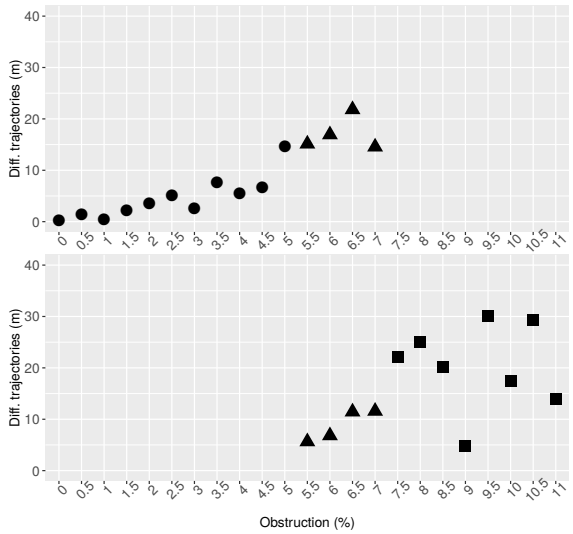


Fig. 10: Indeterminism for the trees experiment. Top graph: successful runs, bottom graph: unsuccessful runs

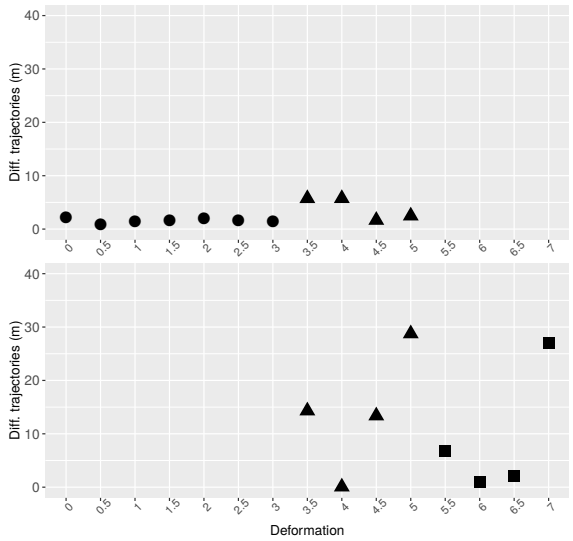


Fig. 11: Indeterminism for the $s = 2$ smoothness experiment. Top graph: successful runs, bottom graph: unsuccessful runs

types of obstacles, and possibly additional control parameters for intra-type variability. The difficulty introduced by the irregularities of the terrain may be coarsely controlled by the size of the map, the number of subdivisions and the amplitude of the deformation.

B. Indeterminism (Q2)

We compare the timestamped positions of the robot observed for repeated runs on the same map. Figures 10 and 11 show the results for the obstruction by trees and for deformations applied to a map with $s = 2$. Each figure includes two graphs: we separately study the successful and unsuccessful trajectories. Max intra-map distances at time t are calculated, and then we report the average max distance observed for the various maps of a given parameter configuration. Comparison of successful trajectories is focused on the easy and challenging configurations, because the very difficult configurations have too few successes. Likewise, comparison of unsuccessful trajectories is not done for the easy configurations.

Sao cái gì cũng thiếu thì làm experiment sao chính xác được

For all classes of experiments, the results confirm the indeterminism of the navigation. The difference between trajectories is significant, in the order of meters and even tens meters. The max difference seems random. In particular, we did not observe any clear correspondence between increasing difficulty levels and increasing distances (see e.g., Fig. 11). The results do not support the conjecture that the degree of indeterminism depends on the difficulty level. Rather, we must expect diverse trajectories to occur at any parameter configuration. It concerns both the successful and unsuccessful trajectories. The robot may follow diverse paths to successfully reach the destination point (see e.g., Fig. 5) or may fail ending its route in diverse areas of the map.

It means that the test design cannot ignore indeterminism. The behaviour of the navigation must be checked using repeated runs. If a search-based generation process is implemented, the fitness function must synthesize the outcomes of the repeated runs to evaluate a candidate map. If some dynamic elements are included in the test profile, like mobile obstacles traversing the route of the robot, it might be necessary to implement on-the-fly control procedures. How to manage indeterminism will be a major challenge for future work.

C. Study of a faulty version of navigation (Q3)

The faulty version under study has the navigation service "believe" that the physical platform is smaller than its real size. Collisions are all the more likely as the planned trajectory gets closer to obstacles. Figure 12 reports the number of observed collisions for obstruction by trees and buildings. As a general comment, the results are consistent with the previous observation that many small obstacles are more stressful than a few large ones. A much higher number of collisions are observed for trees than for buildings, at all difficulty levels.

Easy configurations tend to have a lower revealing power than the challenging or very difficult ones. For buildings, most of the easy configurations did not allow us to observe collisions. For trees, the average number of collisions was 3.82 for the easy configurations, versus 9.75 for the challenging ones and 9.25 for the very difficult ones. So, testing with the easy configurations is the least effective with respect to this fault, but there is no significant difference between the challenging and very difficult configurations.

We conjectured that creating infeasible missions could be useless to test the navigation. For this fault, it is true that they do not bring an added value compared to the challenging ones. But the very difficult configurations proved surprisingly effective. An explanation might be that the tested navigation algorithm is indeed robust. Even in very difficult maps the robot may manage to travel a sufficiently long distance to experience diverse obstruction situations.

VII. Conclusion

In this paper we presented a framework able to drive large robot navigation test campaigns, running tasks such as:

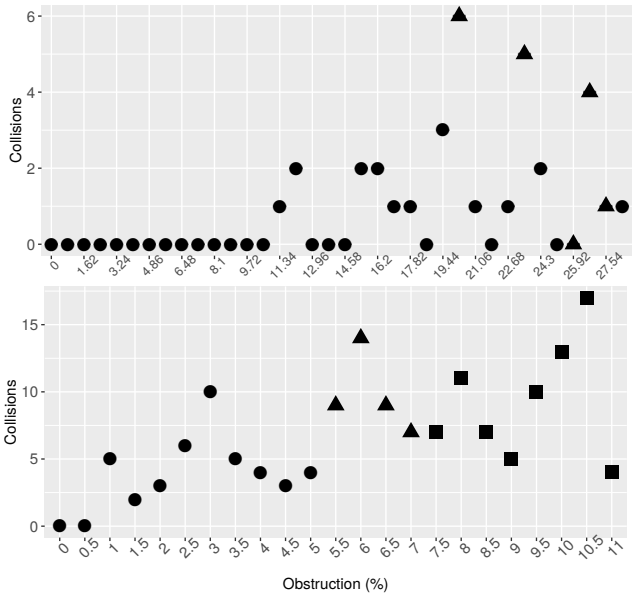


Fig. 12: Number of collisions. Top graph: for the buildings maps, bottom graph: for the trees maps

generating inputs (e.g., a map and a mission) with respect to obstruction and smoothness parameters, running simulations, collecting data and analysing them.

We proposed measurements to characterise the difficulty, in the sense of the difficulty of a navigation problem: its feasibility (a route is found, measured by the observed success rate) and the effort to solve it (time, detours). We determine difficulty levels based on these measurements, yielding a classification of the generated maps. We studied this classification along two macroscopic parameters used to generate the map: smoothness and obstruction. We observed that coarse control of the difficulty can be obtained by tuning these parameters. For obstruction, the size of obstacles was observed to be an important factor. So, obstruction should be tuned by considering not only the obstruction rate but also the weights of the various types of obstacles. These control parameters could be used in the framework of generate-and-test procedures.

While the results we obtained are specific to the SUT, the chosen notion of difficulty and the *a posteriori* approach to classify the maps are general enough to be used for other navigation algorithms. Also, they could be reused for studying controllability by other generation parameters than the ones we studied. The long-term goal is to integrate the control of difficulty into efficient test strategies. As a very preliminary example, we considered a configuration fault related to the robot size. The challenging and very difficult levels proved more capable of revealing the fault than the easy one.

We observed indeterminism through the difference between trajectories by comparing their timestamped positions. A significant indeterminism, from meters to tens of meters, was found for all levels of difficulty and for successful as well as unsuccessful runs. It appears to be impossible to conduct test experiments without facing this phenomena. As a result, test outcomes must be evaluated based on results of several runs. Moreover, we plan to enrich the world model with dynamic elements. The control of their interactions with the robot will have to be performed online to adapt to the current trajectory.

ACKNOWLEDGEMENTS

This publication has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644400.

REFERENCES

[1] V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori, "An open-source simulator for cognitive robotics research: the prototype of the icub humanoid robot simulator," in *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pp. 57–61, ACM, 2008.

[2] F. Kanehiro, K. Fujiwara, S. Kajita, K. Yokoi, K. Kaneko, H. Hirukawa, Y. Nakamura, and K. Yamane, "Open architecture humanoid robotics platform," in *Proceedings of IEEE International Conference on the Robotics and Automation ICRA*, vol. 1, pp. 24–30, 2002.

[3] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *IEEE International Conference on Robotics and Automation ICRA*, pp. 46–51, 2011.

[4] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154, 2004.

[5] O. Michel, "Webots: Symbiosis between virtual and real mobile robots," in *Virtual Worlds*, pp. 254–263, Springer, 1998.

[6] D. Cook, A. Vardy, and R. Lewis, "A survey of auv and robot simulators for multi-vehicle operations," in *Proceedings of the IEEE/OES Conference on Autonomous Underwater Vehicles AUV*, pp. 1–8, 2014.

[7] R. Alexander, H. Hawkins, and D. Rae, "Situation coverage–a coverage criterion for testing autonomous robots," tech. rep., University of York, Department of computer science, 2015.

[8] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik, "A concept for testing robustness and safety of the context-aware behaviour of autonomous systems," in *Agent and Multi-Agent Systems. Technologies and Applications*, pp. 504–513, Springer, 2012.

[9] A. Andrews, M. Abdelgawad, and A. Gario, "World model for testing autonomous systems using petri nets," in *IEEE 17th International Symposium on High Assurance Systems Engineering HASE*, pp. 65–69, 2016.

[10] X. Zou, R. Alexander, and J. McDermid, "Safety validation of sense and avoid algorithms using simulation and evolutionary search," in *Computer Safety, Reliability, and Security*, pp. 33–48, Springer, 2014.

[11] J. Arnold and R. Alexander, "Testing autonomous robot control software using procedural content generation," in *Computer Safety, Reliability, and Security*, pp. 33–44, Springer, 2013.

[12] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[13] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.

[14] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards multiobjective procedural map generation," in *Proceedings of the Workshop on Procedural Content Generation in Games*, p. 3, ACM, 2010.

[15] F. Ingrand, S. Lacroix, S. Lemai-Chenevier, and F. Py, "Decisional autonomy of planetary rovers," *Journal of Field Robotics*, vol. 24, no. 7, pp. 559–580, 2007.

[16] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, and F. Ingrand, "GenoM3: Building middleware-independent robotic components," in *IEEE International Conference on Robotics and Automation*, pp. 4627–4632, 2010.

[17] D. Bonnafous, S. Lacroix, and T. Siméon, "Motion generation for a rover on rough terrains," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, pp. 784–789, 2001.

[18] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *IEEE International Conference on Robotics and Automation*, pp. 3310–3317, 1994.