Simulation and path planning for quadcopter obstacle avoidance in indoor environments using the ROS framework

Conference Paper · January 2018

DOI: 10.1007/978-3-319-69341-5_27

CITATIONS
8

READS
2,351

4 authors:

Yadira Quiñonez
Universidad Autónoma de Sinaloa
35 PUBLICATIONS **119** CITATIONS
SEE PROFILE

Fernando Barrera Pincheira
Universidad Católica del Norte (Chile)
1 PUBLICATION **8** CITATIONS
SEE PROFILE

Ian Bugueno
Universidad Católica del Norte (Chile)
1 PUBLICATION **8** CITATIONS
SEE PROFILE

Juan Bekios-Calfa
Universidad Católica del Norte (Chile)
23 PUBLICATIONS **322** CITATIONS
SEE PROFILE

Simulation and path planning for quadcopter obstacle avoidance in indoor environments using the ROS framework

Yadira Quiñonez[1] , Fernando Barrera[2], Ian Bugueño[2], Juan Bekios-Calfa[2],

[1] Universidad Autónoma de Sinaloa, Facultad de Informática Mazatlán,
Av. Universidad y Leonismo Internacional S/N, México
[2] Universidad Católica del Norte, Departamento de Ingeniería de Sistemas y Computación,
Avda. Angamos 0610, Antofagsta, Chile

yadiraqui@uas.edu.mx, fernando.barrera.pincheira@gmail.com, ibv001@alumnos.ucn.cl,
juan.bekios@ucn.cl

Abstract. This work focuses on the analysis of different algorithms dedicated to the planning of trajectories in a quadcopter. The times and distances of the paths from one point to another have been evaluated autonomously, and the evaluation of the reactive algorithms Bug1, Bug2 and DistBug have been considered to carry out the planning of the quadcopter paths. From the experiments and metrics defined, the efficiency and robustness of the algorithms have been determined. To carry out the implementation of the experiments, we have chosen and evaluated an environment based on the Robot Operating System (ROS) and Gazebo development platform.

Keywords: Bug Algorithm, Unmanned Aerial Vehicle, Robotic Software Framework, ROS.

1 Introduction

In recent years, Unmanned Aerial Vehicle (UAV) has been object of study for different researchers in the scientific community in order to develop tools that facilitate the tasks of repetitive, complex or dangerous work. For example, autonomous exploration in urban environments [1], exploration and generation of three-dimensional maps by autonomous devices of underground mines [2], support to rescue teams to avoid losses in case of landslides or when searching in inaccessible places [3], assistance in decision-making when high-risk situations occur [4, 5], among others.

UAV is a vehicle capable of carrying out a mission without needing of a crew, although it does not necessarily exclude an operator on the ground. It is possible to classify UAVs in different ways, one of the most useful is based on their take-off, dividing vertical take-offs, also known as VTOL (Vertical Take-Off and Landing), which is the fastest growing sector [6], as non-vertical take-offs, known as CTOL (Conventional Take-Off and Landing).

J. Mejia et al. (eds.), *Trends and Applications in Software Engineering*,
Advances in Intelligent Systems and Computing 688,
https://doi.org/10.1007/978-3-319-69341-5_27

In this sense, calculating the appropriate movements for a machine to reach a specific point in space is a great challenge, and involves the execution of complex algorithms to achieve this end, the study of the trajectory planning has been widely developed and an endless number of methods are known to give a suitable solution in time and computational cost, which motivates and justifies to perform a comparative analysis specifically for an aerial mobile robot, in order to demonstrate the viability and performance of the algorithms. Some works focus on the bug algorithm using visual topological maps in unmanned ground vehicle and unmanned aerial vehicle [7, 8].

This work focuses on the analysis of different algorithms dedicated to the planning of trajectories in a quadcopter, where the times and distances traveled from one point to another are evaluated autonomously avoiding obstacles to define the effectiveness and efficiency of the algorithms. The analysis has been done through tests in a simulated environment, which seeks to emulate the real behavior of robots in various environments. The simulator also allows not depending on parts or sensors that may imply a cost in time or money for its maintenance. The algorithms have been implemented in the ROS development platform, an open-source robotic operating system. ROS is oriented to the specific needs of robotic platforms using reusable and modular components.

2 Robotic Software Framework

One of the basic principles of Robotic Software Framework (RSF) is to run many programs or processes in parallel that must be able to exchange data synchronously or asynchronously. For example, a RSF needs to consult sensors of the robot at an established frequency (ultrasounds, temperature sensor, gyroscope, accelerometer, cameras, among others), to recover this data, to debug them, and to direct them to the processes that are dedicated to its processing (Speech processing, computer vision, SLAM). Finally, to act on the motors or other electromechanical devices. All this process is performed continuously and in parallel. In addition, the robotic operating system needs to contain an administrator to ensure efficient access to the robot resources. Currently in the RSF there is no standard, although one of the most used for prototyping in the robotic field is ROS.

2.1 Robot Operating System

ROS is an operating system for robots created by the research laboratory *Willow Garage* in collaboration with Stanford University, is an open-source initiative for the development of robotics [9]. ROS has different tools used during programming, simulation or execution of the robot tasks, such as: Stage, Gazebo, Rviz, TF Library, OpenCV, PointCloud Library, among others.

3 Navigation algorithm

Reactive algorithms do not know the environment, their function is simply limited to the robot reaching the destination point, regardless that path found is optimal or not, an advantage of this type of algorithm is that it decreases the cost of complexity. In this sense, Bug algorithms are navigation methods for mobile robot using local planning. The Bug algorithms make three assumptions about the robot. First, the robot is considered a point in space. Second, the robot has the perfect location capability. And third, the robot has perfect sensors. Although these assumptions are not realistic in a real environment, Bug algorithms are considered as the simplest algorithms and the first step towards a solution to the navigation problem. Even often are used as a top-level monitoring element of a system. The most used and referenced in path planning in mobile robots are Bug1, Bug2, and DistBug algorithms.

3.1 Bug1 algorithm

Bug1 algorithm is the oldest algorithm for obstacle avoidance, it is very simple, it uses a minimum of memory and it does not undergo local minimums [10]. The execution procedure is simple, first, the robot moves along a line until it hits the i-th obstacle, defining a point H_i. Second, it begins to follow the edge of the i-th obstacle, in search of the goal in order to achieve the minimum distance. That is, it simultaneously calculates the distance from the current position to the destination, and finally, stores the point that has the minimum distance.

3.2 Bug2 algorithm

Bug2 algorithm is an improved version of Bug1; it is characterized because it can be run at any point on a continuous path. In this sense, the algorithm starts by generating a slope from initial position to the destination; the robot begins to follow it until it is interrupted by an obstacle. Once it is interrupted, it begins to follow the edge of the obstacle and recalculates the new slope from the new position until the new slope becomes equal to the original slope. Finally, when reaching the point that has the same slope as the previous one, it begins to move to the destination following the previous route generated.

3.3 DistBug algorithm

DistBug algorithm is an improved version of the Bug family, based on distance. That is, the robot travels the shortest distance to get better navigation and reach the destination in less time. Different behaviors are used to avoid obstacles, when the robot encounters an obstacle in the road, begins to follow it and simultaneously calculates the distance from current position to the destination.

4 Experiments and results

One of the main problems of quadcopters is flight instability, which leads to loss or damage of device, especially in the first experimental tests. For this reason, the architecture used to carry out the experiments is necessary which covers the realistic flight dynamics, vision and range sensors, besides an easy integration as the robotic middleware ROS, Gazebo has been selected as a simulator since it has a variety of robots, both generic and commercial.

4.1 Simulation environment

The criteria considered for selecting the simulator were as follow:

- **License:** Authorization that the author(s) grant to third to use his system, there are different types of licenses, commercial, free and gratuitous. Those lower-cost licenses are preferred.
- **Multi-platform** Capacity that a project can be used in different operating systems. The fact that a system is multi-platform, gives it an additional quality, which makes it more versatile.
- **Complexity** Indicates the difficulty that presents when using a simulator, either by its installation, programming, or use. The more complex a system becomes, greater number of people who avoid its use, so it receives less interest.
- **Information** Simulator support found in books, web pages, forums, papers, etc. When there is more information (quality), there is a better handling of the subject, which also implies greater ability to solve problems and errors.
- **Simulated Robots** Ability to simulate different types of robots. The availability of a greater number of robots or of those necessary for a given project, avoids the time invested in simulating a specific robot.
- **Supported languages** Programming languages with which it is possible to work in a system. Having a greater number of languages facilitates the task to the programmers.
- **Physical motor** Software capable of performing simulations of certain physical systems such as rigid-body dynamics, the movement of a fluid, etc. The better the physical engine of a simulator, the simulation will be more real and therefore better results can be obtained.

Table 1 presents a summary of the evaluation obtained between the different simulators considered: Gazebo[1], V-REP[2] y MORSE[3]. Each of the criteria scored between 1 and 7, where 1 is the worst score and 7 is the best.

[1] gazebosim.org
[2] coppeliarobotics.com
[3] openrobots.org/wiki/morse

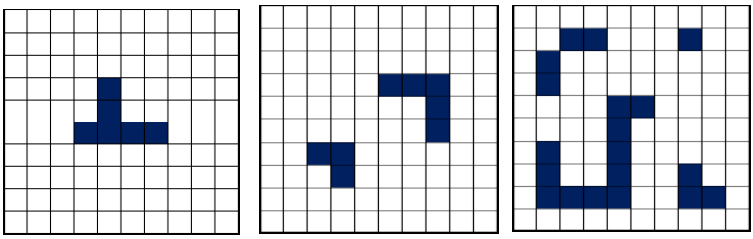
Finally, we have selected Gazebo above V-REP (both obtained the same score) because we have considered that the licensing conditions are marginally better than V-REP.

Table 1. Evaluation of the chosen simulators.

	Gazebo	V-REP	MORSE
License	3	2	2
Multi-platform	1	3	1
Complexity	3	2	1
Information (access	3	2	1
Simulated robots	3	3	2
Supported languages	2	3	1
Physical motor	2	2	1
TOTAL	**17**	**17**	**9**

4.2 Design of the experiment

The experiment consists of a set of different flight simulations for a quadcopter. This has to go from an initial point to an objective avoiding different obstacles. These simulations were run on three different maps Fig. 1), a number of times with different path planning algorithms. The algorithms selected were the reactive algorithms (Bug Family). Each map is defined as a grid that is divided into squares of 1 x 1 meter, this map format is used to facilitate the positioning of obstacles and the initial position of the quadcopter, in addition, it is the default format of the Gazebo simulator. The execution of the algorithms will have the execution time and the distance covered as metrics.



(a) Map A (b) Map B (c) Map C

Fig. 1. Maps proposed

4.3 Simulations

The experiments were implemented on the ROS platform using the Gazebo simulator and encoded in the Python programming language. As mentioned, in tests six times each of the algorithms per map were executed. The objectives for each map that were defined can be seen in Table 2. In addition, the results obtained for the time metric are shown when executing the algorithms Bug1, Bug2, and DistBug. This

metric is based on the **run-time** and it is defined as the time it takes the quadcopter to reach the established goal. For the displacement metric the **distance traveled** is calculated, and is defined as the distance that quadcopter travels until to reach the goal. Table 2 shows the results obtained. It is clear that the **run-time** of Map C is higher because it has more obstacles.

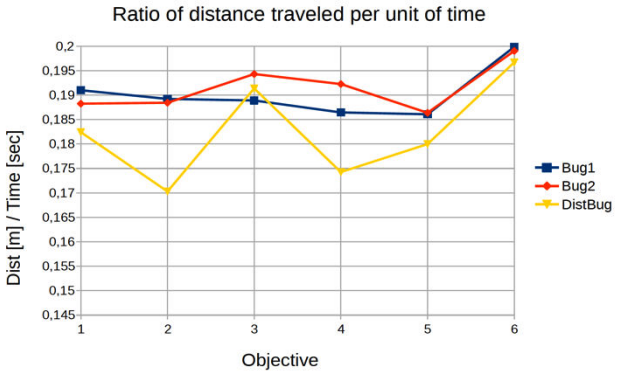
Table 2. Results obtained from the run-time and distance metrics for maps A, B, and C respectively.

Map A	Position	Bug1		Bug2		DistBug	
Objectives	(x, y)	Time	Distance	Time	Distance	Time	Distance
1	(8.0,8.0)	173,351	33,109	86,476	16,279	75,254	13,728
2	(7.0,9.0)	195,058	36,908	91,783	17,295	88,844	15,129
3	(8.0,6.0)	163,367	30,860	62,166	12,079	58,862	11,264
4	(6.0,6.0)	70,061	13,063	68,601	13,189	73,949	12,886
5	(5.0,9.0)	201,64	37,522	101,031	18,830	92,888	16,718
6	(3.0,7.0)	37,957	7,586	38,663	7,695	38,318	7,537

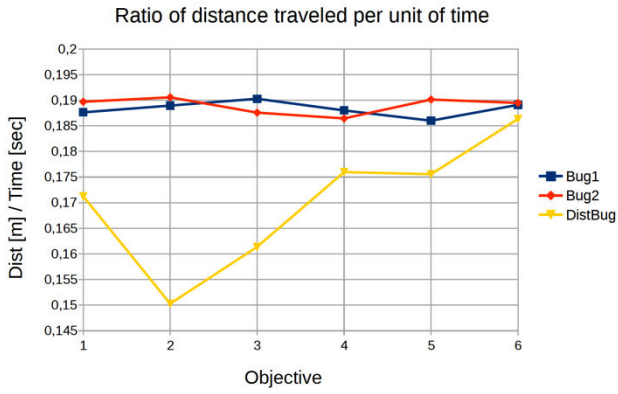
Map B	Position	Bug1		Bug2		DistBug	
Objectives	(x, y)	Time	Distance	Time	Distance	Time	Distance
1	(3.0,6.0)	140,203	26,307	75,593	14,340	72,119	12,35
2	(6.0,5.0)	124,198	23,468	49,868	9,503	59,003	8,867
3	(9.0,5.0)	139,675	26,578	67,502	12,662	68,869	11,114
4	(9.0,9.0)	278,681	52,394	120,756	22,515	102,781	18,087
5	(5.0,9.0)	305,825	56,885	88,307	16,79	139,122	24,424
6	(3.0,9.0)	158,101	29,897	97,544	18,482	82,72	15,416

Map C	Position	Bug1		Bug2		DistBug	
Objectives	(x, y)	Time	Distance	Time	Distance	Time	Distance
1	(3.0,7.0)	249,597	45,663	147,849	25,735	102,633	16,583
2	(3.0,3.0)	115,958	20,274	112,000	20,128	117,842	19,939
3	(1.0,9.0)	299,435	53,245	217,269	38,27	137,479	22,336
4	(6.0,8.0)	238,068	43,689	88,150	15,989	86,905	14,313
5	(9.0,9.0)	304,334	55,697	115,274	20,559	111,633	18,929
6	(9.0,5.0)	229,391	41,998	65,219	12,448	81,715	14,221

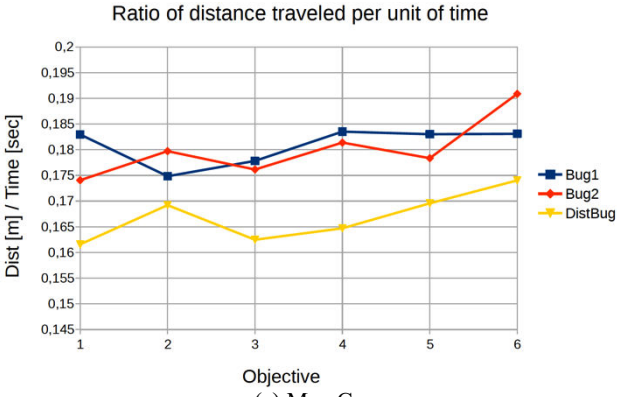
In general, the results are similar for the Bug2 and DistBug algorithms, although, DistBug shows a significant improvement, because its characteristic is to move to the goal once it finds the opportunity to do so. In relation to the Bug1 algorithm the run-time is much greater than the other algorithms because it completely surrounds the obstacle. Fig. 2 presents the results obtained in relation to evaluation of the average speed of displacement (*m/sec*). It can be observed that DistBug is marginally worse than Bug1 and Bug2 algorithms, because it requires a large computational load of both inputs and outputs in each iteration of motion.



(a) Map A



(b) Map B



(c) Map C

Fig. 2. Average speed graphs for maps A, B, and C.

Then, in Fig. 3 and Fig. 4 present the results obtained in Map A and Map C for each of the algorithms, for these simulations in general the results are similar, however, it can be clearly seen that the Bug1 algorithm performs navigation longer

than Bug2 and DistBug, because it completely surrounds the obstacle. In relation to the DistBug algorithm, it can be seen that it reaches the goal in less time.

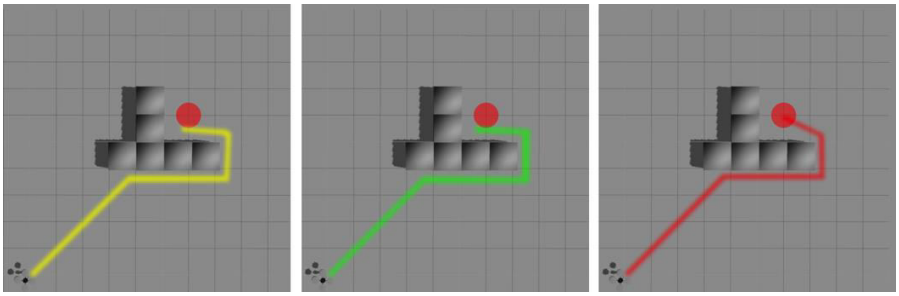


Fig. 3. Comparison of the paths generated by the robot for the Bug1, Bug2 and DistBug algorithms in map A.

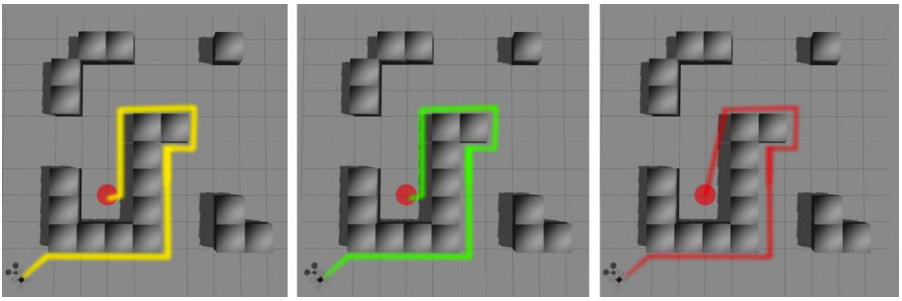


Fig. 4. Comparison of the paths generated by the robot for the Bug1, Bug2 and DistBug algorithms in map C.

Additionally, an experiment has been carried out which includes a modification of the Bug2 algorithm, which in addition to its own performance, the possibility of overflying the obstacle was added. That is, when the robot encounters an obstacle in the path, it rises to find a new point where it is possible to move towards the target. In case it is not possible, the robot begins to follow the edge of the obstacle until to find a new point belonging to the imaginary line. Fig. 5 shows the simulation of quadcopter overflying the obstacle. If the results are compared with the Table 2 it can be noted that for map A and B the improvement is significant in terms of distance traveled (8,596 and 6,548 meters respectively).

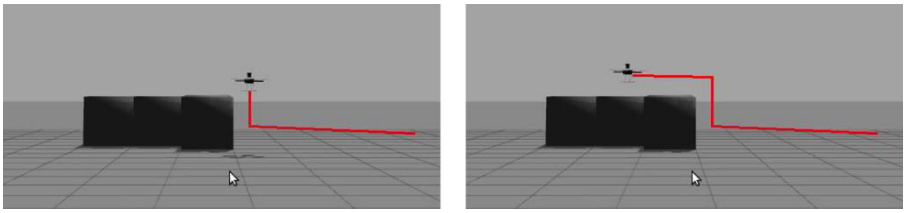


Fig. 5. Simulation of the modified Bug2 algorithm.

On the contrary, in the map C the distance traveled is much greater when the modified Bug2 algorithm is applied. That is, for environments with many obstacles those algorithms tend to fail. Finally, the values obtained in **run-time** are worse than the results obtained in the original algorithm, because the quadcopter tries to evade the wall by making an ascent. This operation has a cost in time especially if the algorithm decides not to continue ascending and return to the original position with the original version of Bug2 algorithm.

Table 3. Results obtained from the run-time and distance metrics for maps A, B, and C respectively.

Map	Goal (x, y)	Distance (m)	Time (s)
A	(5, 9)	10,234	53,690
B	(5, 9)	10,242	54,689
C	(1, 9)	9.309	49,517

5 Conclusions

It has performed an evaluation on three different types of simulators for the development of tests in simulated environments. It has verified that it is possible to perform this type of tests in a simple development environment such as ROS and Gazebo. To achieve this goal, it has executed a set of flight simulations on a quadcopter using different trajectory planning strategies. Three reactive methods of Bug family have been considered as they offer a simple solution to go from an initial point to a target point. In addition, two metrics been defined based on run-time and distance traveled. According to the results obtained, it has been observed that there are obvious differences between the Bug1, Bug2, and DistBug algorithm.

According to the results, Bug1 algorithm obtains the worst results in relation to the run-time and distance traveled metrics, on the other hand, Bug2, and DistBug show similar results. In the case of map C (which has a greater number of obstacles) DistBug has a better performance than the rest. Under these parameters, it can be said that DistBug algorithm performs better than Bug1 and Bug2 algorithms. However, when the average speed of displacement (m/sec) is considered, DistBug is marginally worse than Bug1 and Bug2 algorithms because DistBug is an easy code to implement, but it is difficult to perform for any robot due it requires a large computational load of both inputs and outputs in each iteration of motion. Finally, the modified Bug2

algorithm shows interesting previous results, and although these are inconclusive, it shows an interesting way to explore for the development of new reactive planning algorithms for aerial vehicles.

It would be interesting to study the possibility of varying the speed of the quadcopter and implement simulations with evasion of mobile obstacles, this would imply faster reaction times in both the sampling and the quadcopter. Finally, to contrast the simulations in real environments, since although the simulators are becoming more realistic, they do not consider all the problems that are in a real environment.

References

1. Adler, B., Xiao, J., Zhang, J.: Autonomous exploration of urban environments using unmanned aerial vehicles. Journal of Field Robotics, vol. 31, no. 6, pp. 912–939 (2014)
2. Thrun, S., Thayer, S., Whittaker, W., Baker, C., Burgard, W., Ferguson, D., Hahnel, D., Montemerlo, D., Morris, A., Omohundro, Z., Reverte, C., Whittaker, W.: Autonomous exploration and mapping of abandoned mines. IEEE Robotics & Automation Magazine, vol. 11, no. 4, pp. 79–91 (2004)
3. Kumar, V., Michael, N., Yoshida, K., Naganati, K.: Aerial robots for rapid response: Remote autonomous exploration and mapping. University of Pennsylvania, Philadelphia, USA, (2012)
4. Chen, J. Y.: UAV-guided navigation for ground robot tele-operation in a military reconnaissance environment. Ergonomics, vol. 53, no. 8, pp. 940–950 (2010)
5. Nugroho, G., Satrio, M., Rafsanjani, A. A., Sadew, R. R. T.: Avionic system design Unmanned Aerial Vehicle for disaster area monitoring. In IEEE International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation, pp. 198-20, (2015)
6. Garcia, R., Valavanis, K., Kandel, A.: Autonomous helicopter navigation during a tail rotor failure utilizing fuzzy logic. Mediterranean Conference on Control Automation, pp. 1–6, (2007)
7. Maravall, D., de Lope, J., Fuentes, J.P.: A Vision-Based Dual Anticipatory/Reactive Control Architecture for Indoor Navigation of an Unmanned Aerial Vehicle Using Visual Topological Maps. International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC, vol. 7931, pp. 66-72, (2013)
8. Maravall, D., de Lope, J., Fuentes, J.P.: Visual Bug Algorithm for Simultaneous Robot Homing and Obstacle Avoidance Using Visual Topological Maps in an Unmanned Ground Vehicle. International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC, vol. 9108, pp. 301-310, (2015)
9. The Robot Operating System (ROS), www.ros.org
10. Sezer, V., Gokasan, M.: A Novel Obstacle Avoidance Algorithm: "Follow the Gap Method". Robotics and Autonomous Systems vol. 60(9), pp. 1123-1134 (2012)