An aerial robot path follower based on the 'Carrot chasing' algorithm *

Hector Perez-Leon, Jose Joaquin Acevedo, Jose A. Millan-Romera, Alejandro Castillejo-Calle, Ivan Maza, and Anibal Ollero

Robotics, Vision and Control Group, University of Seville, Avda. de los Descubrimientos s/n, 41092, Sevilla, Spain
{hectorperez,jacevedo,jmromera,acastillejo,imaza,aollero}@us.es

Abstract. This paper presents a three-dimensional path follower implementation for an aerial robot based on the carrot-chasing algorithm. The main objective was to improve the performance of the position controller of the PX4 autopilot when following a list of waypoints. This autopilot is widely used in the aerial robotics community, but we needed to improve its performance for navigation in cluttered environments. Different simulations have been carried out under the ROS (Robotic Operating System) environment for the comparison between the position controller of the PX4 and the proposed path follower. In addition, we have implemented different modes to generate the path from the input list of waypoints that are also analyzed in our simulation environment.

Keywords: Aerial robotics, Path following, Carrot chasing algorithm

1 Introduction

The use of aerial robots for different applications, such as surveillance [1, 2], wildfire tracking [3, 4], transportation [5] and bridge inspection [6, 7] has been increased significantly during the last years. A common requirement for all these applications is the precise, robust and efficient autonomous tracking of predefined paths by the aerial robots.

The path following problem for aerial robots is well studied in the literature, and there are different control based or geometric methods. Carrot-chasing [8], pure pursuit [9], vector field [10] and line-of-sight (LOS) [11] methods are some common geometric algorithms.

Sujit et al. [12] compared path following algorithms for straight lines and loiter paths that are easy to implement, take less implementation time and are robust to disturbances. The authors proved that the carrot-chasing algorithms have the worst performance due to wind disturbances and vector field algorithms are more accurate than the other two dimensional path following algorithms. To fix this issue, Nunez et al. [13] took into account the wind gusts as they play a key role in small prototypes.

* This work is partially supported by the MULTIDRONE (H2020-ICT-731667) European project and the ARM-EXTEND (DPI2017-89790-R) Spanish project.

Xavier et al. [14] compared three dimensional path following algorithms for loiter paths with and without wind disturbances. The authors demonstrated that vector field algorithms have largest errors than carrot-chasing and pure line-of-sight (PLOS) [15] methods.

In this paper, a three dimensional path follower implementation based on the carrot-chasing algorithm is presented. It can be used without any configuration or based on a list of parameters and increases the performance of the position controller of the PX4 autopilot when following a list of waypoints. It has been integrated with the *UAV Abstraction Layer* [1] (UAL) [16] previously developed by our research group within the ROS-MAGNA framework [17].

The rest of the paper is organized as follows. Section 4 defines the system architecture, which frames the proposed system. The path following problem is stated in Sect. 2 and Sect. 3 describes the proposed solution. Finally, validation results are presented in Sect. 5 and conclusions in Sect. 6 close the paper.

2 Problem Statement

This paper poses the path following problem for velocity-controlled aerial robots. An aerial robot Q, which current position is defined by $\mathbf{p}(t) \in \mathbb{R}^3$ at any time t, has to track a path Γ of length L, defined by a curve $\gamma(\lambda) \in \mathbb{R}^3$ with $\lambda \in [0, L]$.

Let us assume that Q is holonomic and velocity-controlled, being its velocity defined as $\mathbf{v}(t)$ at any time t. Then, the aerial robot motion is controlled via velocity commands, such that $\frac{\mathrm{d}\mathbf{p}(t)}{\mathrm{d}t} = \mathbf{v}(t)$. On the other hand, $\mathbf{v}(t)$ is bounded by $v_{\max}$, such that $|\mathbf{v}(t)| \leq v_{\max}$ at any time t.

The objective is to implement a control system to generate velocity commands in order to track the path, minimizing the minimum normal distance between the actual trajectory travelled by Q and the path Γ, which is given by

$$J = \frac{1}{T} \int_0^T \min_{\lambda < L} |\mathbf{p}(t) - \gamma(\lambda)|, \tag{1}$$

where T is the time taken to complete the task.

3 Proposed Approach

The proposed system has two main components: the path generator and the path follower. The user can interact with both components or just with the follower, which is the default way to use the proposed framework (see Fig. 1).

3.1 Path Generator

The path generator is in charge of generating a path Γ based on the ordered list of waypoints WP_l received. The generated path is a much more dense list of

[1] https://github.com/grvcTeam/grvc-ual

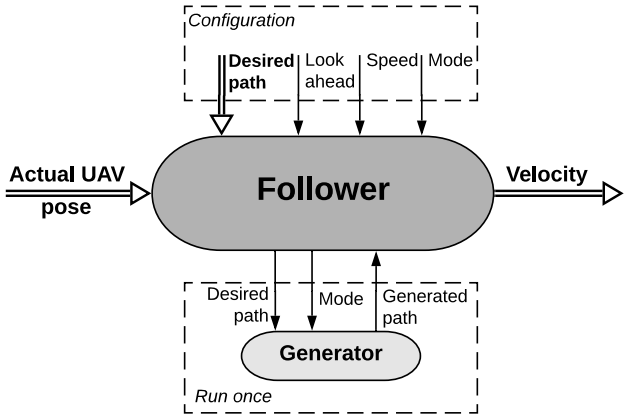


Fig. 1. The path follower design allows to use it by simply entering the desired path and the current position of the aerial robot. It also provides more configuration options to suit the user needs. The generator is called by the follower and runs once to generate a discrete curve.

waypoints, which can be approximated to the continuous curve $\gamma(\lambda)$ described in Sect. 2. It has three modes (m) to generate a new path interpolating the initial list of waypoints, related to the type of curve used for the interpolation. Each mode has advantages and disadvantages, as it will be shown in Sect. 5, and the users should select the one that better fits their needs.

3.2 Path Follower

Initially, the path follower receives the desired path Γ defined as a list of waypoints WP_l and may receive three parameters: the look-ahead distance d (1.0 m by default), the cruising speed v_c (1.0 m/s by default) and the generator mode m (0 by default, see Sect. 5 to find more details about the modes). Parameter default values are conservative, but setting these values properly is crucial to obtain a good performance, depending on the desired path.

A much more dense list of waypoints is required to apply the path following method efficiently. Hence, it uses the path generator to get a discrete curve $\gamma(\lambda)$ from the ordered list of waypoints WP_l, based on the generator mode m. Then, continuously, it receives the aerial robot pose $\mathbf{p}(t)$ and generates the velocity commands $\mathbf{v}(t)$, based on the method described below.

Path Following Method The proposed path following method is based on the 'Carrot chasing' algorithm and illustrated in the Fig. 2. The method runs as follows:

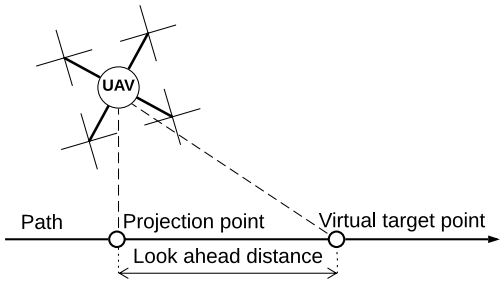


Fig. 2. Top view of the three dimensional path follower based on the carrot-chasing algorithm without taking into account the orientation error.

1. Obtain the λ_p argument as

$$\lambda_p(t) = \underset{\lambda \in [0,L]}{\operatorname{argmin}} |\mathbf{p}(t) - \gamma(\lambda)|, \tag{2}$$

   which minimizes the distance from the aerial robot position to the path.
2. Add the look ahead distance and get the target virtual pose in the path as

$$\mathbf{p}_t(t) = \gamma(\lambda_p(t) + d). \tag{3}$$

3. Calculate the velocity command, based on the cruising speed, as

$$\mathbf{v}(t) = v_c \frac{\mathbf{p}_t(t) - \mathbf{p}(t)}{|\mathbf{p}_t(t) - \mathbf{p}(t)|} \tag{4}$$

   to reach the target virtual pose.

The developed method includes two modes: following the path without changing yaw or aiming at the virtual point.

4 Software Implementation Details

The work described in this paper has been integrated with the UAL, which tries to abstract the user-programmer from the platform's autopilot, defining a common interface with a collection of the most used information and functionalities of an aerial robot. In particular, the developments presented in this paper are based on the release 2.2. of UAL and the Kinetic version of ROS [2] [18]. The proposed system receives a list of waypoints, generates a path using this list, and calculates which velocity vector should use UAL as reference to reach these waypoints.

The software architecture is split into four main layers, as depicted in Fig. 3. In the upper half is the proposed path follower, which has been packaged as a

[2] https://www.ros.org/

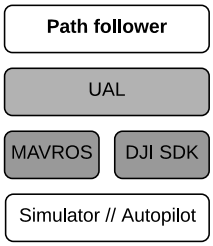


Fig. 3. The different layers of the software architecture make the system modular. Different autopilots and simulators can be used due to the advantages of using the UAL.

node in the widespread ROS to facilitate experimentation and integration, and is built on top of UAL. The lower half of the software architecture is composed by the autopilots, simulators, and communication drivers. The UAL provides a back-end that works with MAVROS [3] which is in charge of providing a communication driver to ROS for various autopilots that uses MAVLink [19] as communication protocol. MAVROS is the ROS adaptation of MAVLink protocol. The simulator used in these developments is based on the PX4 *Software In The Loop* (SITL) [20] development which is the official SITL environment for the Pixhawk autopilot [21]. UAL has implemented another back-end which works using the ROS SDK that DJI provides to communicate with DJI protocols.

4.1 Software User Interface

The system is written in C++, allowing a high performance, and offers a double interface in its current implementation:

- C++: the user may have access to all the functionalities of the framework creating an object in his code. Any ROS topic, service or action is not required to run this interface.
- ROS: The framework may work using ROS communications (topics, services and actions) if the user prefers to work with another programming language like Python. It publishes continuously its output and responds to service calls.

As this framework aims to improve behaviour using velocity control, it is recommended to use the C++ class interface to avoid communications delays present on ROS communications. The path follower is the main module in the system and the user can interact just with it to have a successful path following. The class generator is called automatically by the follower to simplify the interface with the user. However, if the framework is used from the ROS interface, the

[3] https://wiki.ros.org/mavros

generator will be a fully completely independent node even though the interface will be the same.

Table 1. Double user interface implemented. UAL provides the possibility of manage multiple aerial robots, for that reason, each one has a namespace (ns) and a path follower associated.

C++		ROS
Path	preparePath(path, mode look ahead, cruising speed)	Service preparePath
Void	updatePose(pose)	Service updatePose
Void	updatePath(pose)	Service updatePath
Velocity	getVelocity()	Topic /[ns]/velocity

The interface is simple, the user can set everything up just with the *preparePath* method. To read the velocity that the aerial robot must use at that instant the user can read the output of the method *getVelocity*. Before reading the velocity, the user should give the aerial robot pose to the follower using *updatePose* in order to calculate correctly the velocity. The method *updatePath* can be used to change the path during the flight. It will not affect the behavior of the path following because it calculates the velocity referenced to the pose given of that instant.

The proposed framework is under continuously development and publicly available in a stable version along with examples and a guide of how to use it. It can be found in the GitHub repository [4] under the MIT License.

5 Validation Results

This section presents different simulations results using the proposed system. As UAL integrates the robot simulator Gazebo [22], the developed path follower may be tested easily in simulation using different aerial robot models. All the simulations presented here have been performed based on the same aerial robot model, a simulated autopilot based on the PX4 firmware and assuming a maximum speed of 1.0 m/s. The path follower has used the maximum speed of the aerial robot model as cruising speed. UAL allows to provide sequentially a list of waypoints to the PX4 position controller (using the UAL method *setPose*). Thus, these simulations compare the proposed system based on the developed solution with the original position controller.

First, the behavior of both systems are compared for a straight path at the same altitude. Fig. 4a, Fig. 4b and Table 2 show a better performance using the path follower rather than the system based on the position controller.

[4] https://github.com/hecperleo/upat_follower/tree/robot19

On the other hand, both systems are compared for a straight path varying the altitude. The results illustrated in Fig. 4c and Fig. 4d show the main problem of the method based on the position controller. As the simulated aerial robot model has different maximum velocities and accelerations on different axes, the aerial robot behavior is different on each axis. Table 2 presents significantly differences stating the proposed framework as a better solution to follow a list of waypoints in three dimensional space.

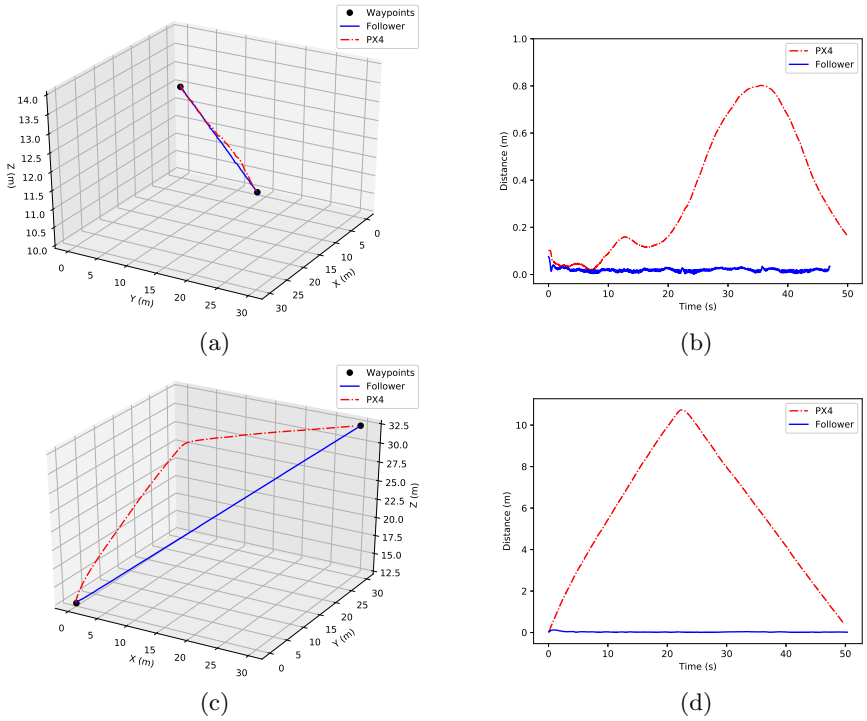


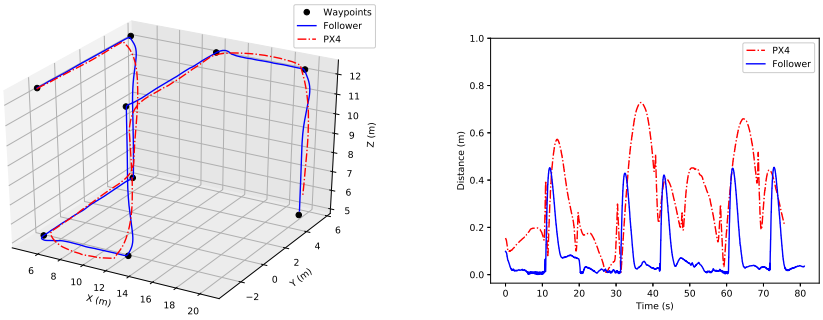
Fig. 4. Comparison between the path follower and the PX4 position controller going through two separate waypoints at same (a, b) and different (c, d) altitude. (a, c) Behavior of the comparison in a three dimensional view. (b, d) Detailed view of the values of the normal distance through the path.

Also, the behaviors of both systems to track more complex paths have been simulated, see Fig. 5a. The results show how the proposed path follower solution works better than setting waypoints using PX4 position controller, see Table 3.

With respect to the path generator node and the generator mode, previous simulations have been performed using the generator mode 0, which uses linear interpolation between waypoints. Although this configuration improves the behavior with respect to the original position controller, it still has some problems.

Table 2. Results of the normal distance of the path follower and the PX4 position controller going through two separate waypoints at same and different altitude.

	Same altitude		Different altitude	
Normal distance (m)	PX4	Follower	PX4	Follower
Mean	0.347	0.019	5.757	0.028
Maximum	0.802	0.076	10.733	0.124
Minimum	0.012	0.001	0.086	0.002
Variance	0.074	0.000	9.356	0.000
Standard deviation	0.271	0.008	3.059	0.018



(a) (b)

Fig. 5. Comparison between the path follower and the PX4 position controller following a list of waypoints. (a) Behavior of the comparison in a three dimensional view. (b) Detailed view of the values of the normal distance through the path.

For example, the results show a maximum value of the normal distance of 0.454 meters, which coincides with the peaks shown in Fig. 5b, because the inertia of the aerial robot prevents from changing the course quickly.

If generator mode 2 is used, the generated path is more curve-shaped with smoothed corners based on cubic splines. Also, if a less curve-shaped path with smoothed corners is needed, generator mode 1 can be used, see 6a and 6c. The difference between these two modes is that mode 1 has a joint between each pair of waypoints so the three dimensional interpolation results on a less curve-shaped path. The results show better behavior by having a smoother path without abrupt course changes, because it reduces the peaks and the mean values of the normal distance, see Fig. 6b, Fig. 6d and Table 3.

Several videos of these simulations are publicly available on web page [5]. For each simulation, a video is provided with the configuration of the proposed path follower and the visualization using RViz [6] (ROS visualization). Extra videos,

[5] https://grvc.us.es/robot19path
[6] https://wiki.ros.org/rviz

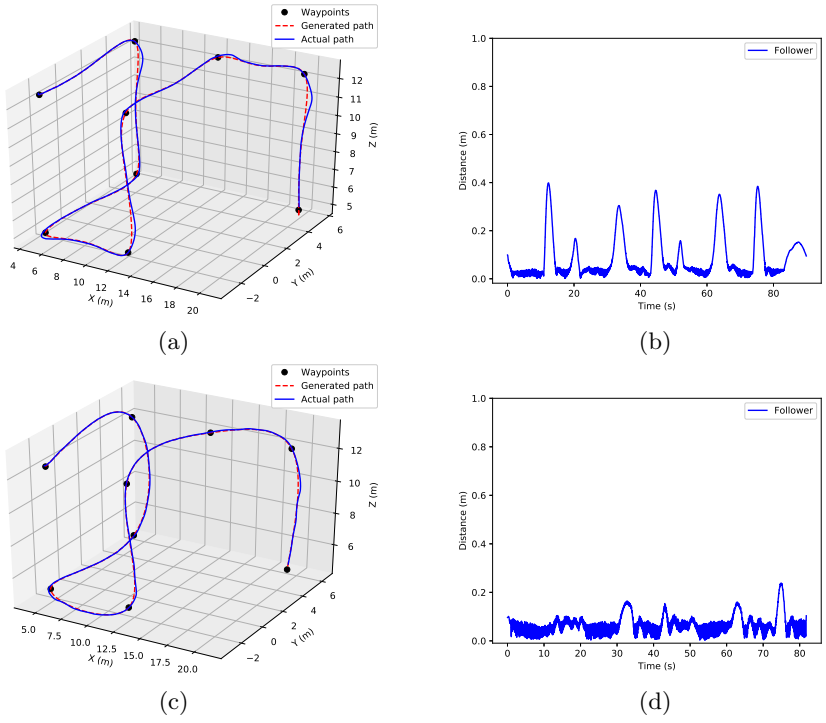


Fig. 6. Path follower using the generator mode 1 and 2. (a, c) Behavior of the path follower in a three dimensional view. (b, d) Detailed view of the values of the normal distance through the path.

Table 3. Comparative of the behavior of the path follower using different generator modes. Mode 0 uses lineal interpolation. Mode 1 and 2 use cubic interpolation.

	Mode 0		Mode 1	Mode 2
Normal distance (m)	PX4	Follower	Follower	Follower
Mean	0.323	0.088	0.086	0.064
Maximum	0.728	0.454	0.399	0.238
Minimum	0.010	0.000	0.002	0.002
Variance	0.015	0.015	0.009	0.001
Standard deviation	0.186	0.122	0.095	0.038

showing behavior of the aerial robot when too large or short look ahead distances are configured, are also provided.

6 Conclusions

This paper presents a path follower which improves the performance of the position controller of the PX4 autopilot when following a list of waypoints. It can be used without any configuration or based a list of parameters: look ahead distance, cruising speed, and the mode of the generator. Trying different values on simulation several times is recommended before going to fly in the real world because these values directly depend on the input desired path.

The proposed path follower and the PX4 position controller have been compared. The path follower presented better behavior than the original controller on every case, but the difference is larger if the waypoint list had variations on altitude. The solution has been compared with different modes, showing different advantages and disadvantages of each one.

The obtained results shows that at higher speeds the aerial robot may oscillate about the path if a short look ahead distance is settled. On the other hand, if the user sets a large look ahead distance the aerial robot will cut corners because the aerial robot tries to turn towards each new virtual point.

The presented path follower may be placed between a path planner and the UAL. The resulting waypoints of the path planner should be sent to the presented framework instead of directly to UAL to increase significantly the whole performance with respect to the waypoints tracking precision.

References

1. Basilico, N., Carpin, S.: Deploying teams of heterogeneous UAVs in cooperative two-level surveillance missions. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 610–615. IEEE (sep 2015), http://ieeexplore.ieee.org/document/7353435/
2. Acevedo, J.J., Arrue, B.C., Maza, I., Ollero, A.: A decentralized algorithm for area surveillance missions using a team of aerial robots with different sensing capabilities. In: Proceedings of the IEEE International Conference on Robotics and Automation. pp. 4735–4740 (May 2014), http://dx.doi.org/10.1109/ICRA.2014.6907552
3. Merino, L., Caballero, F., de Dios, J.M., Maza, I., Ollero, A.: An unmanned aircraft system for automatic forest fire monitoring and measurement. Journal of Intelligent and Robotic Systems 65(1), 533–548 (2012), http://dx.doi.org/10.1007/s10846-011-9560-x
4. Pham, H.X., La, H.M., Feil-Seifer, D., Deans, M.C.: A Distributed Control Framework of Multiple Unmanned Aerial Vehicles for Dynamic Wildfire Tracking. IEEE Transactions on Systems, Man, and Cybernetics: Systems pp. 1–12 (2018), http://arxiv.org/abs/1803.07926 http://ieeexplore.ieee.org/document/8331947/
5. Kondak, K., Ollero, A., Maza, I., Krieger, K., Albu-Schaeffer, A., Schwarzbach, M., Laiacker, M.: Unmanned Aerial Systems Physically Interacting with the Environment: Load Transportation, Deployment, and Aerial Manipulation, pp. 2755–2785. Springer Netherlands (2015)
6. Sanchez-Cuevas, P.J., Ramon-Soria, P., Arrue, B., Ollero, A., Heredia, G.: Robotic system for inspection by contact of bridge beams using uavs. Sensors 19(2) (2019), https://www.mdpi.com/1424-8220/19/2/305

7. Yoder, L., Scherer, S.: Autonomous Exploration for Infrastructure Modeling with a Micro Aerial Vehicle. In: Wettergreen, D.S., Barfoot, T.D. (eds.) Springer Tracts in Advanced Robotics, Springer Tracts in Advanced Robotics, vol. 113, pp. 427–440. Springer International Publishing, Cham (2016)
8. Micaelli, A., Samson, C.: Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. Ph.D. thesis, INRIA (1993)
9. Coulter, R.C.: Implementation of the pure pursuit path tracking algorithm. Tech. rep., Carnegie-Mellon UNIV Pittsburgh PA Robotics INST (1992)
10. Nelson, D.R., Barber, D.B., McLain, T.W., Beard, R.W.: Vector field path following for miniature air vehicles. IEEE Transactions on Robotics 23(3), 519–529 (2007)
11. Fossen, T.I., Breivik, M., Skjetne, R.: Line-of-sight path following of underactuated marine craft. IFAC Proceedings Volumes 36(21), 211–216 (2003)
12. Sujit, P.B., Saripalli, S., Sousa, J.B.: An evaluation of uav path following algorithms. In: 2013 European Control Conference (ECC). pp. 3332–3337 (July 2013)
13. Nunez, H.E., Flores, G., Lozano, R.: Robust path following using a small fixed-wing airplane for aerial research. In: 2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015. pp. 1270–1278. Institute of Electrical and Electronics Engineers Inc. (2015)
14. Xavier, D.M., Natassya Silva, B.F., Branco, K.: Comparison of path-following algorithms for loiter paths of Unmanned Aerial Vehicles. In: Proceedings - IEEE Symposium on Computers and Communications. vol. 2018-June, pp. 1243–1248. Institute of Electrical and Electronics Engineers Inc. (2018)
15. Kothari, M., Postlethwaite, I., Gu, D.W.: A suboptimal path planning algorithm using rapidly-exploring random trees. International Journal of Aerospace Innovations 2 (2010)
16. Real, F., Torres-Gonzalez, A., Ramon-Soria, P., Capitan, J., Ollero, A.: UAL: an abstraction layer for unmanned vehicles. In: 2nd International Symposium on Aerial Robotics (ISAR) (2018)
17. Millan-Romera, J.A., Perez-Leon, H., Castillejo-Calle, A., Maza, I., Ollero, A.: ROS-MAGNA, a ROS-based framework for the definition and management of multi-UAS cooperative missions. In: Proc. of the International Conference on Unmanned Aircraft Systems (ICUAS). pp. 1–10. IEEE (June 2019)
18. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. ICRA workshop on open source system (2009)
19. Meier, L., Camacho, J., Godbolt, B., Goppert, J., Heng, L., Lizarraga, M., et al.: Mavlink: Micro air vehicle communication protocol. [Online]. Tillgänglig: http://qgroundcontrol. org/mavlink/start.[Hämtad 2014-05-22] (2013)
20. Meier, L., Honegger, D., Pollefeys, M.: PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In: Proceedings - IEEE International Conference on Robotics and Automation (2015)
21. Meier, L., Tanskanen, P., Fraundorfer, F., Pollefeys, M.: Pixhawk: A system for autonomous flight using onboard computer vision. In: 2011 IEEE International Conference on Robotics and Automation. pp. 2992–2997. IEEE (2011)
22. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). vol. 3, pp. 2149–2154 vol.3 (Sep 2004)