Training a drone using ROS and OpenAI Gym

Conference Paper · April 2018

CITATIONS
0

READS
3,251

2 authors, including:

Gonzalo del Corral Tercero
Universitat Politècnica de Catalunya
1 PUBLICATION **0** CITATIONS

SEE PROFILE

Date of publication 25th March 2019.

Training a drone using ROS and OpenAI Gym

GONZALO DEL CORRAL TERCERO[1], MIQUEL MACÍAS[2]
Polytechnic University of Catalonia
Corresponding authors: gonzalo.del.corral@estudiant.upc.edu[1], miquel.macias@upc.edu[2]

ABSTRACT This paper explains how using a set of software libraries and open-source tools such as ROS and OpenAI it is possible to control drones and provide them with real autonomy by means of Reinforcement Learning. The paper will explain in a way that a non specialized or none technical audience would understand how those frameworks operate, the possible business applications and which drone manufacturers and models are compatible with the software mentioned before.

INDEX TERMS ROS, OpenAI, Gym, UAV, RPAS, Drone, Artificial Intelligence, Reinforcement Learning

I. INTRODUCTION

DRONES have demonstrated over the past few years that they have arrived to stay, UAVs are not only for military use anymore, now scientists, engineers and other civilians are starting to adopt these technologies. With this rise, drone manufacturers have emerged also enabling a wide variety of models in the market. The problem comes when drone and autopilot manufacturers have closed access to their APIs, or they are restricted to a specific framework that doesn't meet all the user requirements.

Fortunately, ROS (Robot Operating System), a software tool, is becoming popular and more drones are starting to be compatible with it, so users can easily program those drones without needing to adapt to their APIs. This change in becoming open to these frameworks is permitting that lots of interesting applications with drones to be developed. Artificial intelligence, computer vision, 3D mapping, are some of the possibilities that ROS, linked with other frameworks such as OpenAI, OpenCV allows us.

II. ROS

ROS stands for Robot Operating System, is a open source framework that started as a part of Stanford [1] University project STAIR. It provides programmers an easy way to work with all sorts of robotic platforms. It has multiple libraries that allow to code in different languages such as C++, Python and Lisp.

It's considered an operating system for robots because it gives similar functionalities to those you would expect from a conventional operating system, hardware abstraction, and low-level device control. In other words, the programmer doesn't need to care about implementing code for a specific hardware because ROS takes care of that. The same way a web developer is not coding how to set the pixels brighter or lower in the screen, because this is done by the operating system communicating with the screen (hardware).

Being open source helps the community to use libraries that fit best with the needs, in this paper we will integrate drone robots, with ROS. We will analyze the different drones in the market and check for projects and libraries that build on top of ROS to interact and control these drones.

A. NODES AND TOPICS

To understand better how ROS handles the processes we need to talk about ROS nodes. Each node is in charge of controlling a single process in the robot. In a drone, nodes could be the camera control, motors, inertial systems, range finder etc. By separating the processes in different and isolated nodes ROS prevents multiple nodes failing as a result of a chain reaction. It also simplifies the code allowing programmers specialized in different fields to access these isolated modules without compromising others far from their knowledge e.g., an image processing engineer who doesn't know about motors, can access easily the camera node and if there is a failure, motors and other nodes, should keep working properly.

ROS became so popular also because of the way it handles communications. It uses Topics to exchange message across nodes, with a publish/subscribe mechanism depending on nodes interest on a topic. A node sends a message by publishing it to a given topic, for example, a drone that uses range finder sensors for obstacle avoidance will publish the data collected by the range finder. A node interested in reading this data will be subscribed to this topic, the node that controls

[1] http://stair.stanford.edu/

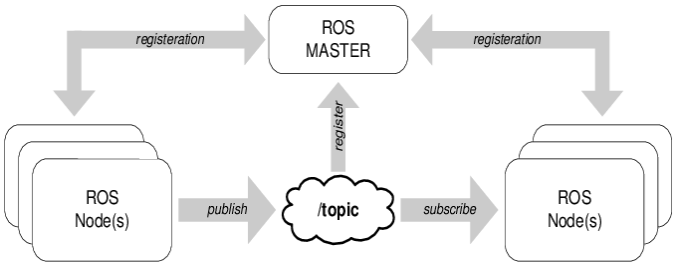

FIGURE 1. ROS Node and Topics scheme

the motors and movements of the drone should subscribe to this topic so it will know when an obstacle is too close to keep moving in a specific direction. The scalability of this system works incredibly well, allowing communication in the same way as seen before, also between different robots and middlewares.

1) Visualization and Monitoring

Another interesting feature ROS provides among Topics is the statistics, so programmers can easily identify when package contain errors or are getting lost due to high traffic or congestion in the network, enabling to adjust parameters to achieve the best communication between nodes.

Scripts can be written which subscribe to a particular topic to plot a particular type of data in quite an easy way. However, ROS provides a more powerful visualization program, *rviz* which is distributed with ROS as a plugin. Visualization dashboards can be dynamically instantiated to view a large variety of data types, images, point clouds, geometric primitives (such as object recognition results), render drone paths and trajectories, etc.

III. OPENAI GYM

Drones are aerial robots, and advanced robots should be able to work autonomously using AI (artificial intelligence). We know how to give instructions to handle communication between nodes in our drone through ROS, but how can these instructions be automated? here is where OpenAI comes in.

OpenAI is a non-profit organization created by Elon Musk, that works on implementing the OpenAI open source framework. To get a little bit of context, Elon is also the founder of Tesla, automobile company focused on electric and autonomous cars. This framework tries to simplify the complexity of AI algorithms by providing a user friendly framework that allows programmers to train their models. OpenAI has a really useful toolkit callen OpenAI Gym which is focused on a machine learning algorithm that is becoming really popular due to the amazing results these algorithms are getting when tested against classic computer games. The machine learning algorithms are called reinforcement learning algorithms.

A. REINFORCEMENT LEARNING

Reinforcement learning is basically how animals and humans learn, trial and error over hundreds or thousands of iterations make us learn. This is relatively easy to code. Every time the decision taken by the agent (robot) helps it on its target, it will receive a reward, otherwise it will receive no reward or a punishment. For example, if our target is to make a drone follow a target without crashing into the environment, we will have to reward the drone if it is close to the target, lets say between 3 and 10 meters. Every time the drone will be in this range it will get points as a reward. And if it crashes against something we will take points away. The target of the drone is to get as many points as possible.

To feed the machine learning algorithms, we just need ROS to send the values from the different sensors on the drone, camera, laser scanner, inertial systems etc. The other step is to set the rewards and punishments and that is it. Now OpenAI Gym should be able to learn on its own how to follow the target while avoiding crashes.

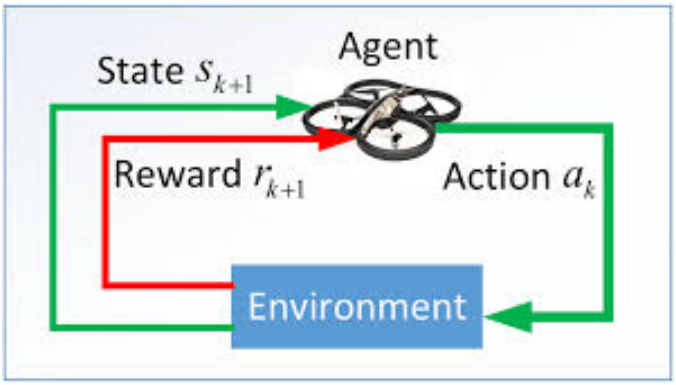

FIGURE 2. Reinforcement Learning workflow

We talked before about trial and error, and that's how the previous algorithm will get better and better after some iterations, it will start doing random actions, with no sense, but after some iterations it will try to follow some of the actions from previous iterations that ended up in getting a reward. The programmer will determine when the algorithm has learned enough and save the learning model created through the iterations.

But you may think, this method is really slow and it will break thousands of drones in each failing iteration. No! that's what makes Gym even more powerful. Gym counts on a simulation test bed, and it provides lots of different scenarios where the user can see what behavior the robot is taking for each iteration, this is really useful because it allows the programmer to improve the punish/reward system, and be confident that in real tests the drone will behave in a similar way, reducing the chances of failure in real scenarios.

IV. COMPATIBLE DRONES

Nowadays there are dozens or even hundreds of different drone models. Which can be divided into countless categories, according to weight, structure, type of controller

or even by the type of application that is carried out with them. For this ROS study we will focus on a drone software division, i.e., if the drone controller uses an open access software or if the access to the controller is restricted. We will focus only on ROS-compatible models for obvious reasons. In addition we will make a subdivision in each commented category differentiating the drone structure, i.e., if the drone is sold completely assembled, and is therefore a commercial product ready to fly as soon as it is purchased. Or if, on the other hand, the components of the drone have to be bought separately, so that from the purchase to the possible first flight there has to be a process of assembly and prior calibration. We should also specify that all the examples to be tested and shown in this document are both simple and good for getting initiated into the ROS drone control environment.

A. CLOSED SOFTWARE CONTROLLERS

With these type of models there are two possible options, the first one is that the drone is completely assembled so once purchased the model, it is already operational, or simply needs a small calibration process or software update to operate. On the other hand, it could be purchased the controller with all the systems isolated, and the framework to use can be selected by the users. Once everything is assembled, again, it is only necessary to run a small calibration to operate with the drone.

In the same way the initial use of ROS with these platforms is also simplified by using existing projects for these platforms. The open source concept on which ROS is based has helped to increase the number of examples available to work with drones of this type, so you can easily find different projects that only require compilation to start using ROS on these devices.

1) Assembled closed-software controllers

The first platforms to be shown in this report that offers full ROS support are those listed in Table 1.

TABLE 1. Assembled and closed software controllers in the market

Company	Models
Parrot	AR Drone BebopI BebopII
DJI	M100 M210 M600 Pro
AsTec	Pelican Hummingbird

These commercially available devices are ready to work with ROS at the time of purchase so they offer greater integration time and ease-of-deployment than all other platforms. Some examples of the ROS software that are available online for these devices are the following:

- ARDrone autonomy.
- Bebop autonomy.
- DJI Onboard SDK ROS.
- AsTec Drivers.

They show in detail the processes to install and compile the ROS section, and also which ROS frameworks are supported in these examples. Following these tutorials you can implement ROS and control the devices. The results of some of these implementations will be presented in the Testing ROS integration on a real drone section.

2) Disassembled closed-software controllers

In table 2 are shown the controllers that are available in the market disassembled. In order to be implemented in the desired framework of the operator.

TABLE 2. Disassembledd and closed software controllers in the market

Company	Models
DJI	N3 Controller A3 Controller

For this type of controllers, an assembly process is required prior to operation. You must choose a framework to install the controller and its relevant communication and control devices (which are included in the pack available on the market for these devices). Once the controllers are implemented, the process for using ROS is the same as in the subsection assembled closed controllers. Even the tutorials to use for the two devices presented in table 2 are the same as those shown above related to the DJI brand. In this case the example DJI Onboard SDK ROS.

This example in ROS implemented on the N3 controller will be presented in the section Testing ROS integration on a real drone.

B. OPEN-SOURCE SOFTWARE CONTROLLERS

Analogously to the closed software controllers we will divide the open-source in two types, those that are sold totally assembled in a platform and those that are sold as components and require other devices and framework to obtain the drone in its totality.

The implementation of these controllers in ROS requires more time and may become more complex as it may require knowledge in the protocols used by the controllers for communication. But in any case there are also examples and initial tutorials of ROS for this type of platform that require only compilation and a certain environment in the operating system used to run ROS.

1) Assembled open-source controllers

In table 3 are shown the controllers that are available in the market assembled and with open-source controllers. It is also shown the type of software/protocol used by the controller.

This type of controllers that are already assembled at the time of purchase offer a lower deployment time than the ones which have to be assembled, although higher than the ones

TABLE 3. Assembled and open-source software controllers in the market

Company	Models
Erle-Robotics	Plane Copter Hexa-copter
BitCraze	Crazyflie
GaiTech	Gapter

with closed software, since they normally require a longer calibration and preparation time for the flight.

In a similar way to closed software controllers there is a great variety of projects for ROS of different levels in order to start interacting with these type of devices. These examples, as well as the others mentioned above, are rapidly integrated, since, as it has been said several times, they are simple examples to get started in the knowledge and use of ROS.

Some of the examples of ROS software that are available online for these devices are listed below, among them, there are specific examples for some platforms or generic examples for the MAVlink protocol:

- Crazyflie
- Erle-Copter Robots
- ROS + MAVlink

In all of them there are detailed steps and commands to execute to run the examples and control drones by using ROS.

2) Disassembled open-source controllers

This last type of controllers is the one that implies a greater time both to assemble the drone platform, as well as to calibrate and prepare the drone for the flights. These controllers also requires a greater time to start performing some tests with ROS (only because a greater number of packages are required to install).

The advantage over closed software controllers is greater interaction with the drone, i.e. a greater number of different operations. They also provide access to all control parameters (stability, acceleration, etc.) of the controller. The fact that they are open-source controllers facilitates communication with the ROS platform regardless of which software is used, since in most cases a compatible communication protocol can be implemented. That being said, in this section we will focus on all the controllers that by default can support the MAVlink protocol, as this is one of the most used in the drone community. Table 4 shows the different controllers that fulfill the commented requirement.

All controllers shown in the table support all types of frameworks (quad-copter, hexa-copter, octo-copter, plane, etc). It is only necessary to indicate the type of framework of the drone when setting up all the parameters in the controller during the assemble phase.

For these controllers there are several examples to get started in ROS and control the drones using this platform.

TABLE 4. Disassembled and open-source software controllers in the market

Company	Models
Pixhawk Project	Pixhak Pixhawk (1, 2, 3, 4, 5) HKPilot 32 PixFalcon DroPix PixRacer

However, the one that has been found most relevant and easiest to implement is the one mentioned below:

- ROS + MAVlink

This tutorial shows the process for installing the ROS libraries needed to communicate via MAVlink and some examples of implementation. It is similar to the rest of the tutorials commented during the whole publication.

C. CONTROLLERS SUMMARY

TABLE 5. Summary of all the ROS compatible platforms available on the market

	Open-Source	Closed API
Assembled	Erle-Robotics (Plane, Copter, Hexa-Copter) BitCraze (Crazyflie) GaiTech (Gapter)	Parrot (AR Drone, Bebop I, Bebop II) DJI (M100, M210, M600 Pro) Astec (Pelican, Hummingbird)
Disassembled	Pixhawk Project (Pixhawk 1, 2, 3, 4, 5, Pixhak, etc)	DJI (N3 and A3 Controllers)

V. TESTING ROS INTEGRATION ON A REAL DRONE

A. TEST ON PARROT BEBOP I PLATFORM

After all the research we decided to test ROS on real drones, we chose Bebop I which is quite a stable drone and it doesn't need GPS location to fly, which is good to fly indoors. Another advantage is that bebop provides an open library available on Github called `bebop_autonomy` that allows you to control the drone through ROS commands and this way it's easier to get familiarized with them before starting a complex project. It basically explains you how to set up the workspace on a pc to connect with the drone. There are two ways you can set the connection with the drone and between the different drone nodes, Node or Nodelet configuration. Node will send all the subscribe and publish messages between processes through TCP/IP, but Nodelet will optimise the messages traffic, if the message travels within the same node manager (machine), only a C++ pointer will be modified to send the message. At the moment Nodelet only have a C++interface.

Once this is set up you can launch directly a few simple actions in the command window such us takeoff, land or

FIGURE 3. Parrot Bebop I platform

piloting, this last one allows you to control the drones flight by publishing geometry_msgs/Twist type messages in the cmd_vel topic with the direction you want the drone to follow in each of the axises.

Learning how to work with these commands it is easy, but we recommend users without experience to practice first in a simulated environment to avoid breaking the drone in the first attempt. After understanding how to build a ROS workspace, we decided to run another project. This project would allow us to work also with the camera on-board the Bebop and process some data in real-time to make our drone move autonomously.

This second test uses the previous one (`bebop_autonomy`) to communicate with the drone platform and a python script to process the video stream and not only detect an object of a specific color but maintain it in the center of the image performing different movements with the drone itself in Z and X axes. This color detector script comes from another library available on Github called `Bebop_ROS_Examples/bebop_color_follower`.

In this test appears two different windows when executing. In the first one appears a black and white image of the video stream, where the white space is the detected object and it is also circled with a red line. A control panel is also present in this window to select the desired HSV color values to detect (Fig. 4).

The second window shows the real video stream with two different dots. One of the dots is red and indicates the center of the image, the other one is a yellow dot indicating the detected object (Fig. 5).

To get the drone moving autonomously the object detected should be placed a few seconds in the center of the frame, making the two dots concur in the same point. After that, moving the object around and waiting some seconds, on the new position, for the platform to locate and react to the new position should result in a movement of the drone to center again the two dots. Analogously in the terminal where the python script has been executed should appear the commands (up, down, left and right) that the drone is executing.

B. TEST ON N3 DJI CONTROLLER MOUNTED IN A F450 FRAMEWORK

Another test that we have performed has been using a N3 controller from the DJI brand, mounted on a F450 framework

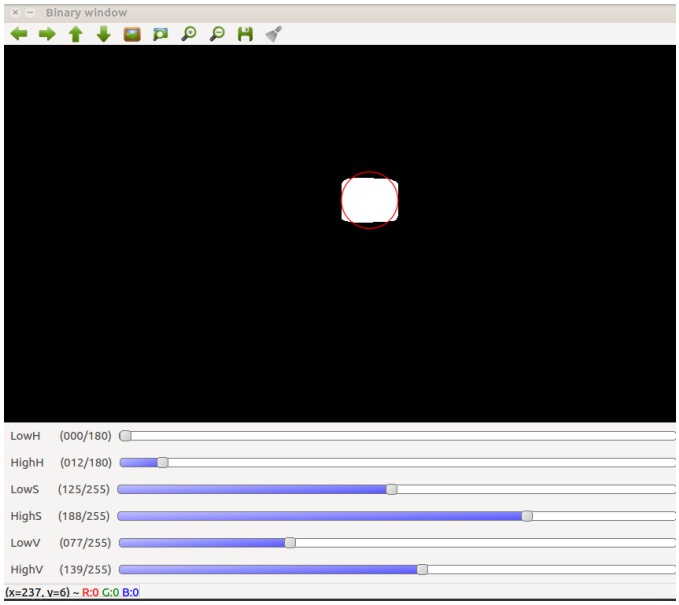

FIGURE 4. Black and white image and control panel from the example.

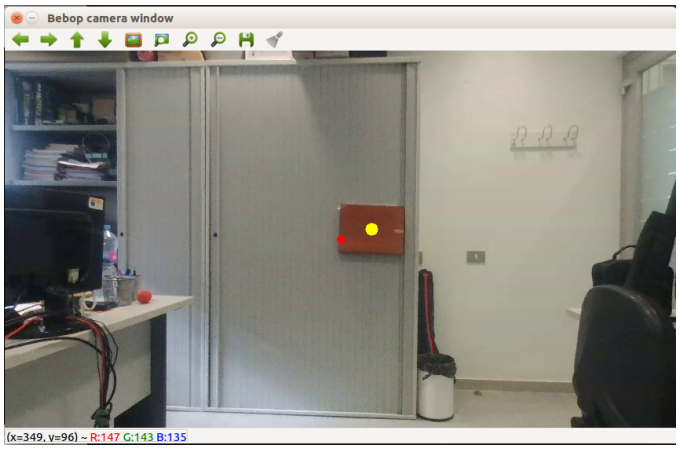

FIGURE 5. Real-time video stream from bebop camera.

(Fig. 6). This controller can use the DJI OSDK and as we said in the Disassembled closed-software controllers subsection is totally compatible with ROS.

This test has been performed following the tutorial called `DJI Onboard SDK ROS`. This test is similar to the first one presented with Bebop. Using ROS, commands can be sent to the drone platform. In this case this commands must be sent through a serial interface, so ROS and DJI OSDK must be executed from a Raspberry or other computer on board the platform to be able to execute the flight. There is also necessary to be outdoors with GPS coverage to execute this test.

VI. CONCLUSION

The whole research has empowered the idea that, the possibilities of the combination of drones and programming are huge. Open source tools are available and easy to use

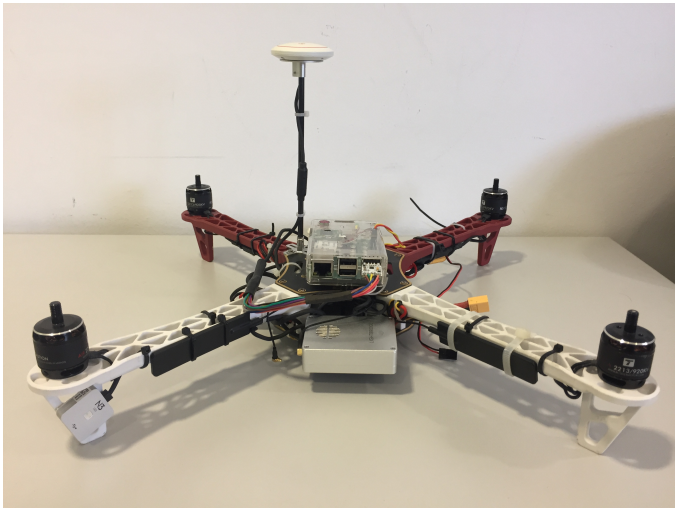

FIGURE 6. N3 mounted on a F450 Framework

with basic knowledge on the field. ROS is a really versatile and agile framework, accessible for beginers and powerful enough to run on complex projects. The greatest thing about ROS is the compatibility with external APIs such us OpenAI, OpenCV, DJI APIs.

For these reasons we understand the growth of this framework and the parallel development of projects all around the world using these tools. We believe more drones will start adopting an opensource mentality allowing thirdparties to build projects that will keep driving technology forward in the UAVs sector.

ACKNOWLEDGMENT

We want to thank our teacher Oscar, for introducing us to such an interesting topic and lending us one of his drones to experiment with it.

GONZALO DEL CORRAL TERCERO Received the B.Eng. degree in Electronics and Telecommunications specialized in Audiovisual Systems from Madrid's University Carlos III (2014) and M.Sc. degree in UAV technologies from the Polytechnic University of Catalonia (2019). From 2014 to 2018 he has been working in the private sector in different fields such as automotive, telecommunications, banking and R&D. During these years he had the chance to work in numerous environments and countries. From mid 2018s he decided to enter into aeronautics by getting enrolled into the UAV Technologies Master's degree, while working at SITEP.SL one of the leading companies in Spain using Drones for GIS (Geographic Information Systems).

MIQUEL MACIAS LOPEZ received 2 B.Eng. degrees one in aeronavegation engineering and the other in Telecommunication Systems from Polytechnic University of Catalonia (UPC) in 2018 and M.Sc. degree in UAV technologies also from the Polytechnic University of Catalonia (UPC) in 2019. He is currently working in a Research Group called Icarus also in the UPC at Castelldefels, Catalonia, Spain.

From 2017 to 2018, he was a Research Assistant at Icarus while he was finishing his Bachelors in the Polytechnic University of Catalonia. Once he finished he become a part of the Research Group and he decided to continue his studies by taking a Master Degree in UAV technologies.

REFERENCES

[1] (2019) The ROS website. [Online]. Available: http://http://wiki.ros.org/
[2] (2019) The OpenAI website. [Online]. Available: https://gym.openai.com/
[3] (2019) Ros: an open-source robot operating system. [Online]. Available: https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf
[4] (2015) Bebop-autonomy. [Online]. Available: https://bebop-autonomy.readthedocs.io
[5] (2018) Stair: Stanford artificial intelligence robot. [Online]. Available: http://stair.stanford.edu/
[6] (2018) Nodelet: Node vs nodelet. [Online]. Available: https://medium.com/@waleedmansoor/understanding-ros-nodelets-c43a11c8169e
[7] (2014) Ardrone-autonomy. [Online]. Available: https://ardrone-autonomy.readthedocs.io
[8] (2017) Dji onboard sdk ros project. [Online]. Available: http://wiki.ros.org/dji$_s dk$
[9] (2012) Astec drivers ros project. [Online]. Available: http://wiki.ros.org/Robots/AscTec
[10] (2015) Crazyflie ros project. [Online]. Available: http://wiki.ros.org/crazyflie
[11] (2016) Erle copters ros project. [Online]. Available: http://wiki.ros.org/Robots/Erle-copter
[12] (2019) Mavlink ros project. [Online]. Available: http://ardupilot.org/dev/docs/ros-install.html
[13] (2018) Bebop example, color follower. [Online]. Available: https://github.com/TOTON95/Bebop-ROS-Examples/tree/master/bebop-color-follower