UAL: An Abstraction Layer for Unmanned Aerial Vehicles

Fran Real, Arturo Torres-González, Pablo Ramón-Soria, Jesús Capitán and Aníbal Ollero

Abstract—This paper presents a software layer to abstract users of UAVs (Unmanned Aerial Vehicles) from the specific hardware of the platform and the autopilot interfaces. The main objective of our UAV Abstraction Layer (UAL) is to simplify the development and testing of higher-level algorithms in aerial robotics by trying to standardize and simplify the interfaces with the UAVs. Our UAL can work seamlessly with simulated or real platforms and provides calls to issue standard commands such as taking-off, landing or pose and velocity controls. Even though UAL is under continuous development, a stable version is available for public use and it is currently being used in several European research projects by different academic and industrial entities.

I. INTRODUCTION

Nowadays there is a remarkable raise in the number of applications with Unmanned Aerial Vehicles (UAVs). In these applications, UAVs are expected to carry out a wide variety of tasks in different environments [1], developing certain levels of autonomy and cognition that are usually implemented by high-level algorithms.

Researchers working with UAVs use different aerial platforms, autopilots, or even different versions of the same autopilot. Such variability comes from the numerous possibilities and the specific constraints of each application. Thus, some applications may require certain platforms depending on, e.g., their payload capacity or their maneuverability; and each platform may require the use of a specific autopilot. However, high-level algorithms should be able to work regardless of the autopilot or platform used, since it becomes too complex maintaining different versions of the software depending on the particular communication protocols for each autopilot.

Several proprietary and open-source frameworks for UAVs have been developed over the past years [2]. Moreover, open-source organizations such as Dronecode [3] proposed a standard communication protocols for UAVs, such as MAVLink [3]. However, despite the increasing use of this protocol, there are still lots of autopilots that do not support it. Also, industrial leaders in UAV manufacturing such as DJI [4] still work with their own proprietary autopilot framework. Other authors proposed more elaborated and complex frameworks to deal with teams of multiple robots (aerial [4] or heterogeneous [5]) or described a whole solution including the hardware platform [6]. However, we focus on abstracting the autopilot to make easier the development of higher-level algorithms (e.g., swarm control algorithms).

Additionally, it is essential the integration of these software frameworks with simulated environments in order to switch between real and simulated platforms without a major effort. Due to the fact that UAVs are more sensitive and fragile than common ground vehicles, previous simulations to verify the correct functioning of the whole system become more critical. Thus, there are widespread simulators [7], [8], [9] to test most of the functionalities of UAV and multi-UAV systems, as for instance task allocation [10] or path planning algorithms [11]. Simulators with more realistic 3D engines can also be used for testing computer vision algorithms [7].

In this paper, we introduce the design and the current implementation of our software framework to abstract UAV users from the differences among autopilots. First, we define a sufficient set of functionalities that every UAV should implement. This set defines a common interface that we have called UAL (UAV Abstraction Layer). Then, for every autopilot we want to support, a specific back-end is coded taking into account autopilot specifics. This proposed framework eases the deployment of algorithms into the real world and promotes a safer development pipeline.

The main contribution of the paper is to expose the design of the UAL and its implementation, which is publicly available. We also depict how our UAL provides a simple way to simulate multiple UAVs and how its interfaces make the use of real or simulated robots transparent to the user. Finally, we describe some use cases to show the versatility of our proposed framework. In particular, UAL has been in a continuous development process for almost 2 years and it has been successfully tested in several experiments within the context of different European R&D projects. We describe the context of each project and how UAL is used to integrate UAVs into different high-level missions.

The remainder of the paper is structured as follows. First, Section II describes our UAL framework. Then, its advantages and specifics for simulation of UAVs are explained in Section III. Section IV shows some use cases on how UAL has been useful in the development of several research projects. Finally, conclusions and future work are in Section V.

II. UAL: A UAV ABSTRACTION LAYER

Our UAV Abstraction Layer (UAL) tries to abstract the user programmer from the platform’s autopilot. With that purpose, it defines a common interface with a collection of the most used information and functionalities of a UAV such as:
• Perform a take-off maneuver to a given height.
• Go from current position to a specified waypoint in global or in geographical coordinates.
• Set global linear velocities and yaw rate.
• Land on the current position.
• Recover from manual flight mode.
• Set home position to the current position.
• Get latest pose estimation of the UAV.
• Get latest velocity estimation of the UAV.
• Get latest transform estimation of the UAV.

UAL builds on top of the widespread Robot Operating System (ROS) [12], which provides libraries and tools to help software developers to create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. The main advantage of using ROS is that the communication between different processes and machines is easily solved. In particular, UAL has been developed and tested on ROS Kinetic Kame, even though it can be adapted to other versions easily.

The proposed framework consists of three layers (see Figure 1). First, the UAL itself. Second, the Backend class, which establishes a common interface to the UAL. Finally, in order to support a particular autopilot, a specific derived back-end for that autopilot must be implemented. This back-end communicates with the autopilot and handles specific details, offering a common interface in the user side. For instance, any autopilot that uses MAVLink as communication protocol (e.g., PX4 and Ardupilot), is currently supported via a MAVROS back-end. The MAVROS back-end communicates via MAVROS with the autopilot, and handles specific issues such as arming and switching mode before taking off.

In its current implementation, UAL offers a double interface (see Figure 2):

• Class: the developer can have an object of the class UAL and access data and functionalities via its class interface, directly calling its member functions.
• Server: at the same time, inside the instance of the class and in a separate thread, UAL can be continuously publishing data and responding to service calls as any other node inside the ROS network.

While the class interface is middleware independent, is always available and introduces no delay, the server interface depends on middleware (ROS in our case), is available only if server mode is enabled (it is by default) and may introduce network delays. Moreover, only one process may have one and only one class interface for a certain UAV. This may be an issue to handle multiple robots. However, the server interface can be reached from any host in the network and has been successfully tested with multiple robots.

These two interfaces are not exclusive in design nor in implementation (every function in UAL interface is thread-safe), and it might be convenient to use both of them, profiting from the advantages of each one. For example, delay-sensitive functionalities like velocity control are better suited to the class interface, whereas it is more useful to call the recover-from-manual service from any console using the server interface.

Until now, there are three supported back-ends for our UAL (see Figure 3). The first one for MAVROS (the ROS adaptation of MAVLink protocol), which is a very extended way of communicating with autopilots. Some well-known autopilots such as PX4 and Ardupilot support it. Second, we have a Light back-end only for simulation, as it will be detailed in the next section. Last, we have implemented a back-end using the ROS SDK that DJI provides to communicate with their own autopilots.
III. SIMULATION FUNCTIONALITIES

In addition to the main advantage of abstracting the user from the autopilot, UAL also provides tools that help the users to easily test their algorithms in simulation. In particular, UAL is totally integrated with the well known open-source robot simulator Gazebo\(^4\). This simulator allows for fast robot prototyping and creation of new scenarios, and it is already integrated within ROS. Besides, we have recently tested UAL with the Unreal Engine\(^5\) and its plugin Airsim\(^6\) for UAV simulation.

UAL comes with two possibilities for simulation in Gazebo: the light simulation and the PX4 SITL simulation. The first one uses a Light back-end that provides a simple model of the UAV, avoiding dynamics, and draws the simulated UAV in Gazebo. This is particularly useful to perform simulations with large numbers of UAVs, where the focus is on a higher-level behavior and not on having realistic dynamics for the UAVs, which may entail computational issues.

The second simulation option is based on the PX4 Firmware\(^13\), which is an open-source autopilot software. Along with the usual autopilot functionalities, PX4 Firmware comes with a Software-In-The-Loop (SITL) simulation environment based on Gazebo and RotorS\(^14\). This SITL has several Gazebo plugins that simulate the sensors (e.g., IMU, GPS, etc.) and the dynamics (e.g., rotor velocities and forces) of the UAV. UAL comes with the possibility to run SITL simulations with the PX4 Firmware. This feature allows the user to run in simulation the same software as in the real platform, replicating low-level behaviors of the autopilot (such as waypoint transformations and mode switching) in a more realistic fashion.

Apart from the above options, UAL provides some scripts to launch quite easily simulations with multiple UAVs. It also provides different UAV models, included in a ROS package called robots\_description.

IV. USE CASES

In this section we present several use cases where UAL is used to interface with real and simulated UAVs. In particular, we showcase the use of UAL for different applications with UAVs within the framework of European projects, where the collaboration between robotics labs has been eased by the usage of UAL. For further coding details, an example that explains the use of UAL for newcomers is publicly available in our github with the name of test\_ual\_interfaces inside the uav\_abstraction\_layer package.

A. Multiple Drones for Media Production

UAL is highly integrated in the architecture of the MULTIDRONE project\(^6\). This ongoing project aims at teams of multiple UAVs that cover outdoor sport events such as cycling, rowing races or football matches.

The MULTIDRONE European consortium is developing algorithms that will transform the ideas of the media production team into autonomous plans and control actions so that the UAVs can shot the event with their onboard cameras\(^15\). All partners have already adopted UAL to interface with the UAVs and they are using it to simulate the first examples developed in the project. Figure 4 shows a simulated mockup scenario to test autonomous shooting missions in the project. Next steps in the project will be using UAL to go from the simulation to real experiments.

![Simulated mockup scenario for MULTIDRONE where two UAVs follow a car taking different types of shots. Views from the two onboard cameras can be seen. This YouTube video\(^16\) shows the full mission simulation.](http://multidrone.eu)

In this project, UAL is used in its server interface. In the proposed architecture, a module in charge of the control of the drone, gimbal and camera communicates with UAL server to get the current pose and velocity and to send velocity control commands to it.

B. Autonomous Bridge Inspection

The European project AEROBI\(^7\) aims to automate the inspection of bridges’ concrete beams and piers by the use of flying unmanned robots equipped with manipulators driven by an intelligent control and a computer vision and sensing system.

For instance, measuring the deflection of bridges is a tedious operation in which an operator places a tool on the beam with a pole or aided with a crane. The tool is a prism which is used by a Total Station\(^8\) to accurately measure positions.

One of the objectives in the project is to use a UAV equipped with such a prism. As described in\(^17\), the UAV can use the drag forces generated by the propellers in the proximity to the ceiling to remain stuck to the beam. Therefore, the Total Station can measure any deformation of any beam at any bridge without putting into risk neither any human operator nor machine.

In that context, the UAL framework has been used to automate the control and movement of the aerial platform during the experimental missions. Figure 5 shows an experiment where the UAV follows a trajectory with three contact points chosen to measure the deflection of a bridge beam. In this particular example, the process of measuring

---

\(^4\)http://gazebosim.org

\(^5\)https://www.unrealengine.com

\(^6\)https://multidrone.eu/

\(^7\)http://www.aerobi.eu

\(^8\)https://leica-geosystems.com/products/total-stations
the deflection uses the class interface to make a collection of \texttt{goToWaypoint} calls to send the platform to the desired measuring points. Once there, the \texttt{velocity} method is used to trigger the beam-contact condition.

![Fig. 5. UAV measuring beam deflection in a bridge. Left, UAV trajectory during the mission with a point cloud captured by the Total Station. Right, snapshot of the UAV stuck to the beam.](image)

C. Aerial Manipulation

The proposed UAL has also been used in the European project AEROARMS \footnote{https://aeroarms-project.eu/} where a UAV equipped with a dual robotic arm is expected to operate in complex industrial environments for inspection and maintenance. Within the framework of the project, UAL has been extended with similar abstract interfaces to operate the robotic arms. In Figure \ref{fig:experiments}, some experiments in a mockup scenario are shown. In the experiments, a reactive navigation algorithm \cite{11} is used to avoid collisions while operating. The algorithm uses UAL for state estimation and velocity control, both in simulation and real experiments. In this particular case, the process in charge of the reactive navigation makes use of the class interface, calls the \texttt{pose} method to update the current pose of the platform (and hence the map of the perceived world) and calls the \texttt{setVelocity} method with the desired velocity from the obstacle avoidance algorithm.

![Fig. 6. Experiments for aerial manipulation in a mockup scenario emulating pipes in a plant. Real and simulated environments are depicted.](image)

D. Fast Development on Robot Competitions

The last use case concerns the Mohamed Bin Zayed International Robot Competition 2017 (MBZIRC) \footnote{http://www.mbzirc.com/} which took place in Abu Dhabi.

We participated as the \textit{Al-robotics} team in this challenge (see Figure \ref{fig:competition}), where a team of 3 UAVs had to perform a mission that combined exploration and picking and placing objects into a box. The participation in this competition, and the need for a common framework for both the actual UAVs and the simulation, motivated the development of a layer for the PX4 flight stack.

![Fig. 7. Left, competition arena for the MBZIRC. A multi-UAV team must find, pick and place a set of objects. Right, the Al-robotics team during the competition.](image)

Preliminary versions of the current UAL were designed during our participation in the MBZIRC. Later, these first approaches converged and generalized as a framework for interfacing UAVs in a standard fashion. In the competition, the UAVs performed a cooperative mission where they executed an area coverage algorithm (see Figure \ref{fig:map}) to search for the objects, and then a task allocation algorithm to get objects assigned for collection. UAL was used by these high-level algorithms to operate the UAVs, for instance, by sending waypoints or controlling them in velocity while collecting the objects. Given the multi-robot character of the application, the server interface was mostly used in this case.

![Fig. 8. Map of the MBZIRC arena. The scenario is split into three areas where the UAVs search for objects at the beginning of the mission. The routes assigned to the UAVs are depicted in different colors. The landing zone of the UAVs (LZ) and the dropping zone (DZ) to place the objects are also shown.](image)

V. Conclusions

This paper has presented UAL, a framework to abstract high-level software development in UAVs, allowing the user to work with different autopilots and platforms by means of common interfaces. After using UAL within the context of several R&D projects, we can conclude that it eases the development of higher-level algorithms. Researchers from different organizations have provided positive feedback and found useful UAL as an enhanced middleware. Moreover, the functionality to interface simulated or real UAVs in the same way has proved to be essential.

We presented a stable version that is publicly available. However, UAL is under continuous development, adding new features and fixing issues. This development would benefit from a more extended use of the UAL by the robotics community. We are currently working on adapting UAL
to newer versions of the platforms already supported and extending it to support new ones.

ACKNOWLEDGMENTS

This work was partially funded by the European Unions Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE). This publication reflects only the authors’ views. The European Commission is not responsible for any use that may be made of the information it contains.

REFERENCES


