A cloud robotics architecture for an emergency management and monitoring service in a smart cityenvironment

Original
A cloud robotics architecture for an emergency management and monitoring service in a smart cityenvironment / Ermacora, Gabriele; Toma, Antonio; Bona, Basilio; Chiaberge, Marcello; Silvagni, Mario; Gaspardone, M.; Antonini, R.. - ELETTRONICO. - (2013). ((Intervento presentato al convegno 2013 IEEE/RSJ International Conference of Intelligent Robots and Systems tenutosi a Tokyo (Japan) nel November 3-8, 2013.

A cloud robotics architecture for an emergency management and monitoring service in a smart city environment

G. Ermacora, A. Toma, B. Bona, M. Chiaberge, M. Silvagni, M. Gaspardone, R. Antonini

Abstract— **Cloud robotics is revolutionizing not only the robotics industry but also the ICT world, giving robots more storage and computing capacity, opening new scenarios that blend the physical to the digital world. In this vision new IT architectures are required to manage robots, retrieve data from them and create services to interact with users. In this paper a possible implementation of a cloud robotics architecture for the interaction between users and UAVs is described. Using the latter as monitoring agents, a service for fighting crime in urban environment is proposed, making one step forward towards the idea of smart city.**

I. INTRODUCTION

A new approach to robotics is coming up exploiting the emerging technologies of internet and cloud computing [1]. While in the past the robot was seen as a unique device that carried out onboard all the computation and storage processes, as Steve Cousins said “no robots is an island” [2], presently we are witnessing the dawn of cloud robotics. This bring us to a shift where “robot intelligence” once local for every single robot will now be managed by a higher and more powerful “centralized brain” located in the cloud architecture [3]. This breakthrough opens new scenarios where robots are seen as agents, relying on remote servers for most of their computational load and data storage, creating a network of devices where they can share knowledge and information [4].

The cloud robotics approach involves the software abstraction of each robot, abstracted from the hardware layer, and presenting ad hoc APIs to ease its management and the process of writing code on it. Even non robotics experts can now write programs without knowing the specific robot software architecture, simply calling the precise APIs from their code. Furthermore, once the API has been approved and tested, this approach facilitates the structure of the program and reduces the possibility of errors. In this way both beginners and experts that want to build a service do not need to know in details the dynamics and the technical features of the required robot.

This paper presents a *platform of cloud robotics* and its related *services*. The platform provides an abstraction layer of each robot accessible via APIs and offers support to services, available to final users [5].

Lots of applications are growing that see UAV as interesting devices for environment monitoring. Many services can be build fetching data from UAVs, such as telemetry or video streaming. Making available these information, users can have access to these services in many ways. As this paper illustrates, these services, part of the IT architecture, can be accessed via web or other devices, as smartphone applications.

Typically in search and rescue or emergency management when UAVs are required for monitoring intervention, users ask the service for intervention that builds a request to the platform of cloud robotics. The latter translates the required mission to a low level language organizing one or a swarm of UAVs to fulfill the request. In addition, the platform is designed to be resilient in case of failure. The mission configuration is prepared creating a message of instructions deployed from the platform to the UAV. In case the UAV loses the connection from the platform, this message is conceived as a buffer of data so that it can accomplish the minimum mission requirements. Additionally the platform accepts mission reconfiguration while the mission has already been submitted to UAVs.

In this paper the architecture previously outlined for emergency management and monitoring services is described. A real security problem in urban context inspired us as a test case proposed in this paper; urban spaces monitored by cameras are not an efficient way to decrease crime rates since criminal events e.g., theft, robbery, rape moves towards unmonitored zones. Thus the aim of this test case is to apply this cloud architecture, based on ROS [6], to crime prevention. In case of aggression the user requests the emergency service from the IT architecture, providing GPS coordinates and an identification number. The IT architecture organizes an UAV to reach him/her for offering monitoring and support. In the meantime a police officer will use the service to see the current position of the UAV, its telemetry and video streaming from its camera.

In this paper there is a first step towards a more complete idea of cloud robotics. In fact the future developments of this project aim to:

- Adding to the cloud architecture also robots of really low capabilities and not running ROS framework
- Adding a network and database repository (e.g. Roboheart [7]) to improve services for monitoring

This work is in collaboration with Telecom Italia Lab.

G. Ermacora, A. Toma and M. Silvagni are with DIMEAS, Politecnico di Torino, Torino, Italy, {gabriele.ermacora, antonio.toma, mario.silvagni}@polito.it

B. Bona is with DAUIN, Politecnico di Torino, Torino, Italy, basilio.bona@polito.it

M. Chiaberge is with DET, Politecnico di Torino, Torino, Italy, marcello.chiaberge@polito.it

M. Gaspardone and R. Antonini are with Telecom Italia Lab, Torino, Italy, {marco.gaspardone, roberto1.antonini}@telecomitalia.it

and emergency management. In this test case we can imagine to add capabilities of cloud computing and storage e.g. a face detection algorithm to recognize aggressors running on the cloud

This work will have to find a tradeoff between relying all the intelligence on the cloud and give robots resilient capacity. The cloud approach makes the robot dependent on the reliability of the Internet connection while, especially in this test case, it is important to respect mission constraints and maintain the control of UAVs in case of connection problem.

Roadmap of the paper. The paper is structured as follows: *Section 2* describes the service from the user and police officer side in. The cloud robotics platform structure and mechanism is explained in *Section 3*. The hardware and UAV implementation is presented in *Section 4*.

II. WEB SERVICES AND INTERFACES

The services described in this paper are the following:

- the first is user-side, for requesting help and assistance from the UAV
- the second is police-side, for monitoring the ongoing situation from the UAV

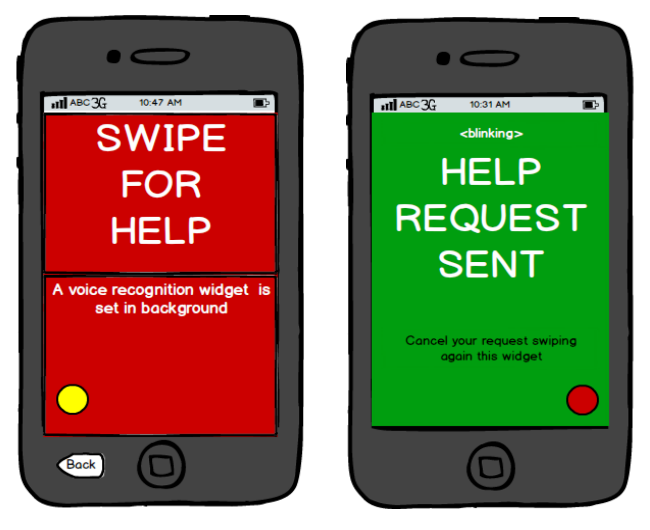
The first service requires a device for sending the help request. This needs a deeper study in the interaction between this device and users (Human Computer Interaction) in order to make such device easy to use and efficient in emergency situation. Following the theories of Donald Arthur Norman about Human Computer Interaction will be topic of future work also to distinguish from intention to action in the interaction with this device [8]. In other words whether the user wants to send a help request or it is just an error of interaction. But since the aim of this paper is not the study of human factors and ergonomics of the device, we are focusing more on the description of the IT architecture. Thus in this paper this device is seen just as a tool which is employed by the user to send an help request. With regards to the technical specifications, the device has to send a POST request, over HTTP protocol, to an ad hoc server listening on a predefined URL and port. In the POST request there are the GPS coordinates at the moment of the emergency call and an Identification Number. The web server is part of the IT architecture and uses the service to send these information to the platform and make the UAV get the user for monitoring and assistance.

In this test case, in order to have an easy way to implement the post request in an affordable and easy to program device, we have chosen an Android based smartphone application. There are possibly other devices that can accomplish the same goal or even better from the usability perspective, e.g., an ad hoc designed bracelet or a smart device but this is not the aim of this paper.

The usage of the Android application is as follows :

- the user knowing in advance that is going to enter in an harmful zone activates the application in his/her smartphone, holding it in background
- there are two ways to send the help request: the first is swiping the swipe button. The application start sending the POST request with GPS coordinates and user's phone number as Identification Number. The swipe button has been chosen in order to reduce errors in the interaction. This particular widget in fact is widely adopted for the incoming phone call or to activate the smartphone from sleeping mode. Another way to send the request is using voice recognition. A voice recognition listener is in background when the user activates the application. In this case the request is activated by saying the word “help”.
- This application can be used even when the user does not expect to be in a harmful situation, assuming he/she would have the possibility to activate the application and swiping over the help request button.
- In case of false positive or error of interaction the user can stop the help request swiping on a similar widget on the other side of the screen.
- While the application is sending the help request, it is designed to produce a loud alarm. This has been chosen for two reasons:
- a) to clearly show the system status to the user: in this way in emergency situation the user can know that the request has been sent correctly. This has been conceived with an audio feedback in addition to a message displayed on the screen since the user is supposed not to have the possibility to give attention to the smartphone.

  b) it is widely adopted in common alarm systems to produce a loud noise. This technique is used as deterrent against aggressors and also as a help request for other people that may passing by.



A voice recognition widget is set in background

Cancel your request swiping again this widget

Figure 1 : two screenshots of the application in wireframe. On the left the asking request activity and on the right the confirmation of the help request and the widget to delete the help request.

In this case we call the Android application *interface* and not *service* since it is not using the presented API from the IT architecture. It just send a POST request to a web server that will utilize the service.

From the police officer perspective a UAV monitoring and management service is created. The officer via web browser will have all the information about the UAV collected by telemetry and video streaming. In this way the he/she can know the actual position of the UAV, displayed on the web page map and see how far and how much time it will occur to get there. In addition, the video streaming from the UAV can offer assistance to the person in emergency. In this case the service is build exploiting the API presented from the IT architecture, so we call it a *service*.

III. THE CLOUD ROBOTICS PLATFORM

As explained above, the term "Cloud Robotics" is a new approach to robotics that takes advantage of the Internet as a resource for massively parallel computation and sharing of vast data resources in a robust environment; here a list of aspects to progressively approach the “Cloud Robotics” concepts mentioned above:

- abstracting the complexity of HW and SW at different levels to final developer,
- transferring the intelligence, that normally resides on robot, to the Internet, i.e., to a Cloud Robotics Platform (remote brain).
- exposing Application Programming Interfaces (APIs), which ease service developments and share the resources amongst different services
    - managing deployed applications where parallel computation typically resides, e.g. keep them alive, error management
    - APIs typically abstracting platform resources
    - developer access, e.g. account management
    - multi user access i.e. concurrent access to platform APIs
    - security, in our case for robot connection, e.g. certified VPN to connect robots

Starting from these concepts, we have built a robotics platform which mainly consists of three layers :

- Front End, containing APIs to build new services
- Application, containing all specific applications (the so called “remote brain”), supporting the above APIs
- Adaption, containing adapters and drivers to connect the different robots and their sensors, and abstracting their basics functionalities to the above applications and APIs .

Our platform is based on ROS framework [9], as showed in Figure 2, where gray boxes represent ROS nodes. The platform context is composed by two additional layers:

- Robots, containing all robots, which are connected to the platforms through specific ROS nodes, named drivers, and adapters (Adaption Layer).
- Services, containing all services, which exploit APIs exposed by the platform (Front End Layer).

All elements running in Application Layer, represent various applications each declined by a ROS node. APIs connect all ROS nodes to abstract their interfaces to service developer. The interface of a ROS node is a message that is typically conveyed by the three different communication processes: publish-subscribe, service-client, action-feedback. Therefore a ROS message is identified by its type and its communication process, namely a topic, service or action name; e.g. the ROS message “geometry-msgs/Twist” [9] and its topic can be abstracted by “move” API, where the name is uniquely related to a topic name (/cmd_vel) and the parameter is an object “Twist” having the same format of ROS message above. Once this API is called by a service, the ROS message is composed and published to ROS framework through the addressed topic.

Another example could be represented by NavData message abstraction, in this case the “get_feedback” API subscribes the related topic (/navdata) and registers a callback to be notified every time NavData message is published on this topic. The previous examples show the one-to-one relation between ROS messages (and their related topics or services) and APIs

Moving from ROS framework concept to ROS Container means introducing a management system, which traverses all layers where ROS nodes reside; so the ROS container includes the ROS framework and adds to it the following managing elements:

- WatchDog System (WDS): to manage ROS nodes
- Message Discovery Function (MDF): to enable or disable APIs according to ROS messages

which will be better explained in the following paragraphs. Other important issues to be considered are Security and Concurrency.

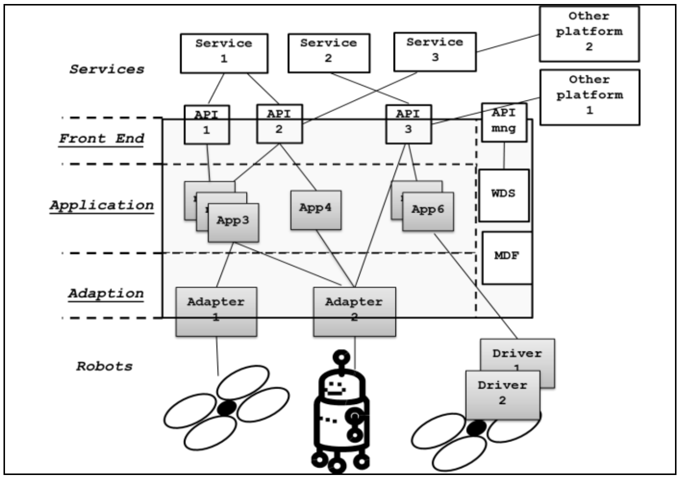


Figure 2 : The cloud platform architecture

Security needs some specific infrastructure and has to be assured at the platform gates, that are Front End and Adaption layer. APIs must be safely accessed after a registration phase, where developers are identified and specific security keys are provided. Robots are connected to the platform via-VPN, possibly certified, in order not to be easily accessed by hackers. Security keys are installed at robot side.

Concurrency, when two or more users access the same service, needs also to be managed at ROS level. If a service makes access to one of the above specified robotic API, a ROS message is exchanged with the ROS framework. Here a ROS node is deputed to manage such message. Therefore this node has to be designed to manage concurrency in a multithread architecture, e.g., the mission planner node (better explained in the following) has to manage two or more drones at the same time.

A. *WatchDog System (WDS)*

A ROS Node needs to be managed taking into account its life cycle, as showed in Figure 3. A ROS node life cycle (NLC) concept is introduced, represented as a simple Finite State Machine. In this representation the states represent node status and the arrows represent both expected (e.g., start and stop) and unexpected events (e.g., error). For managing such events dedicated APIs are introduced, which expose basic functionalities to support error checks and get or modify nodes status ("on demand" start and stop). Therefore a WatchDog System (WDS) is built to perform the following actions:

- enacting strategy to keep ROS nodes alive, e.g., in case of node break down due to unforeseen causes.
- periodically updating status of each ROS node, for the benefits of the Message Discovery Function (see next paragraph)

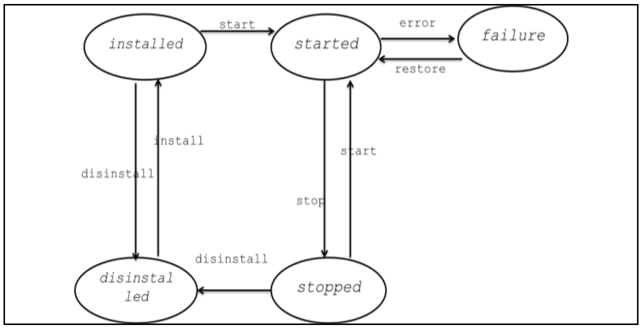


Figure 3 : Node Life Cycle in Finite State Machine representation

B. *The Message Discovery Function (MDF)*

The Message Discovery Function (MDF) enables or disables APIs basing on available ROS messages; as an example, the standard structure for accessing data from a digital camera in ROS is a "sensor_msgs/Image" message. Thus if this message is not present, the API abstracting this message is disabled. Furthermore ROS messages could be related each other, e.g., referring to Figure 4, the message *Msg 1* from Node 1 is related to *Msg 2* from Node 3, so API 1 abstracting message *Msg 1* is also disabled if *Msg 2* is not present. This complex relation between APIs and messages results in a tree structure, where ROS messages are the tree nodes (not to be confused with ROS nodes that contain actually these messages) and APIs are the tree leaves.

Hence MDF visits this tree and for each ROS node reads its current status, updated by WDS, to disable or enable APIs accordingly. For example if node 3 status is "stopped" or "failure", then API 1, 3, 5 and 9 are disabled, whereas if node 2 status is not "started", both API 3 and 10 are disabled.

In Figure 5 the management architecture is depicted: the WDS is a scheduled process to keep alive ROS node (in case of failure) and to update ROS node status. When something changes, MDF is triggered to read latest change and enable or disable APIs accordingly.

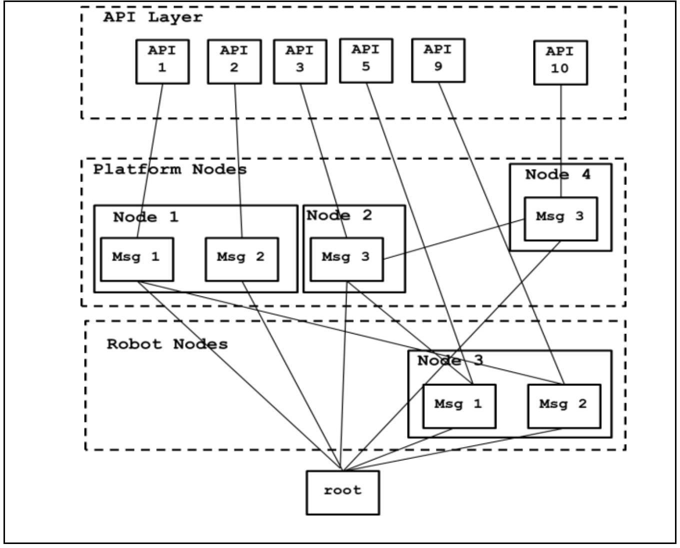


Figure 4: tree structure for Message Discovery Function

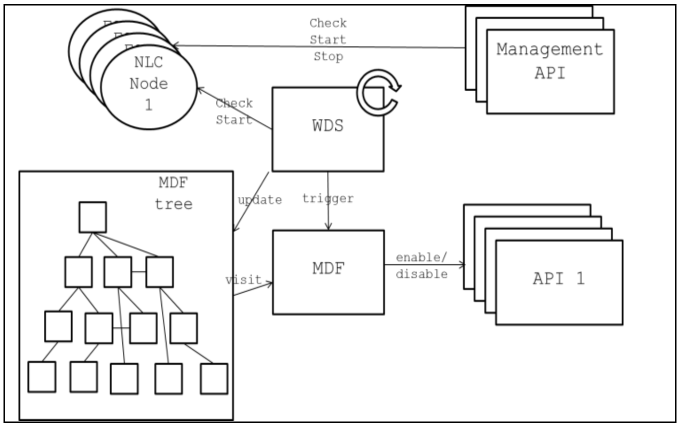


Figure 5 : Management architecture

C. *Emergency and monitor service*

The APIs and nodes supporting the service described in previous chapter, are depicted in Figure 6. The service core logic is implemented in mission planner ROS node, the ROS message and the service/mission are abstracted by **build_mission** API. Thus, in order to build a mission starting from the home GPS position of the UAV to the requested GPS position, a call to **build_mission** API is needed accepting as parameter an object with the following request message structure:

Header *header*
Coordinate *home*
Coordinate *target*

The mission planner receives the above message and optimizes the following cost functions, in order to choose a drone amongst the available ones in terms of:

- Distance to travel
- Drone battery consumption
- Drone battery autonomy

Once a drone is chosen, mission_planner publishes the FlightPlan message in specific topic to chosen drone, by addressing it on namespace basis, and returns that namespace in the following response message structure

String *drone_name*

where *drone_name* is the namespace related to the chosen drone. The drone name previously returned and, as a consequence, strictly linked to emergency request, is used to address following APIs:

- **get_feedback** collects telemetry and sensor data from chosen drone, e.g., GPS current position, in order to feed the drone position to a monitoring system tracking on a geo referenced map.
- **video_streaming** returns the camera video streaming, allowing a remote operator to watch emergency conditions.
- **move** used to remotely operate the drone.

In this service, intelligence is totally transferred from drone to platform. Indeed the mission is planned at platform side, the drones are simply actuators and are connected and managed concurrently by the platform itself. The brain (mission planner and drivers) is kept alive by the WDS platform component and its functionalities (APIs) enabled or disabled by MDF platform component.

IV. THE AGENTS

The first validation-tests of the overall system have been conducted using a quadrotor as agent. A quadrotor offers several clear advantages with respect to other possible choices (fixed-wing UAVs, terrestrial rover, etc.).

In particular a quadrotor is well-suited for surveillance and monitoring tasks because of its capability to hover above the target. The same is valid also for a standard helicopter architecture, but at the price of more complex mechanics and more difficult control scheme.

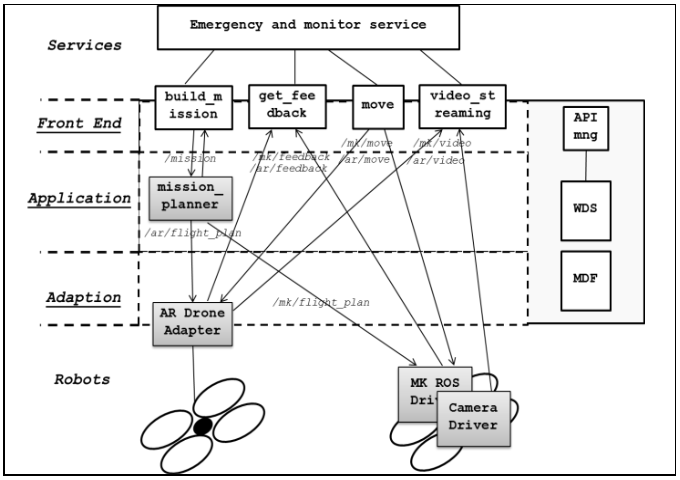


Figure 6: architecture platform for the service

On the contrary a quadrotor is easy to maintain and less expensive than a helicopter with similar features (in terms of autonomy and maximum payload weight). Unfortunately a quadrotor is an inherently unstable system [10], and for this reason it requires some electronics (autopilot) to guarantee its stability in standard flights [11].

Three different products are used in the validation of the proposed architecture:

Parrot AR.Drone: The AR.Drone [12] is a commercial low-cost quadrotor solution, fully equipped for remote control via smartphone. It features a front HD camera and the flight stability is ensured by a mother board (running a real-time linux-based operating system) and a navigation board interfaced with the on-board sensors (two cameras, ultrasonic range finders, gyroscopes and accelerometers).

The AR.Drone is mainly conceived for gaming applications, amusement and Augmented Reality videogames, but due to its low-cost, flexibility and the availability of an official SDK, it gained a very good popularity in the academic community.

Mikrokopter: Mikrokopter [13] is a complete auto-pilot designed for the control of generic multi-rotor platforms. It features two different boards: the Flight Control board guarantees vehicle inherent stabilization and altitude-hold function, the Navi Control board adds a set of GPS/Compass based autonomous navigation functions (waypoint navigation, come-home function, position hold mode). The Flight Controller relies on Atmel ATMEGA644 board running at 20MHz, and interfaces with the main inertial sensors (3-axis accelerometer, three gyros, one barometric sensor). Mikrokopter allows the user to take external control of the UAV (i.e., bypassing the radio controller) by means of a dedicated serial protocol.

Micropilot 2128: uPilot 2128 [14] is an auto-pilot board embedding all the peripherals needed for a stable and autonomous quad-rotor flight. This auto-pilot is specifically addressed to professional use and applications, this is reflected by its higher price and its market segment. Though Micropilot uses a completely closed-source software, it offers some tools allowing the user to write his own code. These functions come with an add-on product called “Xtender” [15]; Xtender provides a dedicated dynamic linking library that acts as a intermediate layer between the user code and the autopilot software. Using the functions encoded in the library the developer is able to get access to several low-level parameters of the auto-pilot and can modify their values.

Due to Micropilot's high price and to its relatively young support to multi-copters when compared to other solutions on the market, it is not so common to find academic works that use this hardware.

The three architectures offer growing functionalities, but also growing difficulties in implementation. Table 1 summarizes their main features and their integration status in ROS environment.

TABLE I. QUADROTORS FEATURES AND INTEGRATION STATUS IN ROS ENVIRONMENT

	Market	**Command**	**Telemetry link**	**Autonomous navigation**	**SDK**	**ROS support**
AR.Drone	Videogames /Hobby	Smartphone (via wifi)	Wifi (TCP/UDP packages)	☹	☺	☺
Mikrokopter	Hobby/ Photographer	Radio controller	UART (Custom Serial Protocol)	😐	☹	😐
Micropilot 2128	Professional applications	Radio controller	UART (Custom Serial Protocol)	☺	☺	☹

Notice that the only platform adequately supported in ROS is AR.Drone; a ROS node for Mikrokopter has also been written [16], but it requires flashing a software patch on the Flight Control board firmware and thus it has been excluded from this study, since we aim at maintaining the compatibility with the standard version of the cited auto-pilot; finally there is not any ROS node dedicated to microPilot support.

Therefore two different ROS interfaces have been written from scratch, one dedicated to Mikrokopter and the second to Micropilot. We choose to manage these nodes differently from the AR.Drone one. In fact the AR.Drone is well-suited for short-range mission and it is acceptable to maintain all its ROS interfaces in the cloud; on the contrary, the Mikrokopter and Micropilot are more likely to be used in long-range GPS-aided missions where a sudden loss of connection with the cloud must not interrupt the mission or – worse – exhibit dangerous behaviours or cause damages and injuries. For this reason, in these latter cases, the ROS driver node runs on a dedicated PC/104 board directly connected to the auto-pilot on the UAV. This choice allows to trigger specific emergency-management routines in case of missing link or communication issues. As depicted in Figure 2 the ROS interface takes the function of *adapter* in the Parrot AR.Drone case, while it should be considered a *driver* for Mikrokopter and Micropilot.

The three described solutions where specifically chosen in order to cover every possible segment of the market and to demonstrate the flexibility of the proposed service in adapting the mission schedule to very different families of available agents. Moreover they easily show the already highlighted difference between the *adapter* and *driver* modules in the cloud platform.

CONCLUSION AND FUTURE WORK

In this paper we propose a test case for cloud robotics for emergency management and monitoring service. We intend to exploit the emerging technologies of web services and mobile applications to use robotics in the proposed cloud architecture. In addition we want to leverage the power of cloud computing in terms of storage and computing e.g. adding the ad hoc cloud engine Roboearth [7]. Experimental results and data will be available in the next few months since the project is under development. A deeper and more complete study of Human Robot Interaction in emergency context will be also part of our future work.

REFERENCES

[1] Mell, Peter, and Timothy Grance. "The NIST definition of cloud computing (draft)." *NIST special publication* 800.145 (2011): 7.

[2] Ken Goldberg, Ben Kehoe , *Cloud Robotics and Automation: A Survey of Related Work.* Electrical Engineering and Computer Sciences University of California at Berkeley

[3] Sanfeliu, Alberto, Norihiro Hagita, and Alessandro Saffiotti. "Network robot systems." *Robotics and Autonomous Systems* 56.10 (2008): 793-797.

[4] Chibani, A., et al. "Ubiquitous robotics: Recent challenges and future trends." Robotics and Autonomous Systems (2013).

[5] Kamei, Koji, et al. "Cloud networked robotics." *Network, IEEE* 26.3 (2012): 28-34.

[6] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.

[7] Waibel, Markus, et al. "Roboearth." *Robotics & Automation Magazine, IEEE*18.2 (2011): 69-82.

[8] Norman, Donald A. "Categorization of action slips." *Psychological review* 88.1 (1981): 1.

[9] http://www.ros.org

[10] Hoffmann, Gabriel M., et al. "Quadrotor helicopter flight dynamics and control: Theory and experiment." *Proc. of the AIAA Guidance, Navigation, and Control Conference*. 2007.

[11] Bouabdallah, Samir, and Roland Siegwart. "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor." *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005.

[12] http://ardrone2.parrot.com/

[13] http://mikrokopter.de/en/home

[14] http://www.micropilot.com/

[15] http://www.micropilot.com/products-xtendermp.htm

[16] Sa, Inkyu, and Peter Corke. "Estimation and control for an open-source quadcopter." *Proceedings of the Australasian Conference on Robotics and Automation 2011*. 2011.