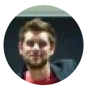
Leveraging open data for supporting a cloud robotics service in a smart city environment



Gabriele Ermacora
Politecnico di Torino

Leveraging open data for supporting a cloud robotics service in a smart city environment

G. Ermacora[1], A. Toma[1], R. Antonini[2] and S. Rosa[3]

[1] Politecnico di Torino, DIMEAS, Torino, Italy
{gabriele.ermacora, antonio.toma}@polito.it
[2] Telecom Italia Lab, Torino, Italy
roberto1.antonini@telecomitalia.it
[3] Politecnico di Torino, DAUIN, Torino, Italy
stefano.rosa@polito.it

Keywords: cloud robotics · cloud computing · open data · smart city · robotics · service · UAV

Abstract. At the dawn of cloud robotics one of the biggest challenges is to successfully exploit the power offered by the internet for acquiring and sharing information, in order to build a common knowledge base among the agents. In this paper we propose a cloud robotics service for emergency management in a smart city scenario. The application is capable of collecting significant open data over the internet. Then it uses them in the mission planning phase, in order to autonomously define a set of waypoints towards the target coordinates.

1 Introduction

The emerging technology of cloud computing [1] is revolutionizing the robotics industry, opening new scenarios where robots are seen as agents and are managed by a high-level platform [2]. In this perspective cloud computing allows robots to share knowledge and perform demanding real-time data processing [3].
The cloud robotics approach involves the concept of "robot-as-a-service" [4], referring to robots that, abstracted from the hardware layer, can be dynamically combined to give support to the execution of specific applications [3], [5], [6].

This paper presents a service, supported by open data, for emergency management in a smart city environment within a cloud robotics architecture. An actual security problem inspired us in the definition of a test case, which was described by the authors in a preliminary version of this work [7]. Surveillance cameras are not an efficient way to decrease crime rates in urban spaces, considering that criminal events such as theft, robbery, rape, etc. may occur in unmonitored zones.

We imagine the following workflow: the user requests the emergency service from the cloud, by providing GPS coordinates and an identification number via a smartphone application. The cloud platform chooses an UAV to be sent to the emergency site in order to provide monitoring and support. In the meantime, a police officer can access the service via a web browser to monitor the current position of the UAV, its telemetry, and to receive a video stream from the UAV's camera.

A special block of the cloud robotics platform, called in this work ODOMI (Open Data Oriented MIssion planner), computes a suitable path to send to the UAV from its initial position to the emergency zone. ODOMI calculates the best path for the UAV based on the knowledge of possible obstacles and on the availability of communication between the platform and the UAV. This knowledge is acquired from open data accessible over the Internet.

Finally the platform transforms the path computed by ODOMI into a mission, encoding it in a lower level language message to be deployed to the UAV.

The architecture is based on the Robot Operating System (ROS) [8].

Roadmap of the paper. The paper is structured as follows: *Section 2* describes the service from the user and police officer point of view. The cloud robotics platform structure and its implementation are explained in *Section 3.* The ODOMI block is presented in *Section 4,* while *Section 5* draws some conclusions.

2 Interfaces

In this work two interfaces are implemented to interact with the services: a user-side interface to request help and assistance from the UAV and a police-side interface for the emergency monitoring and management.

The first interface is a smartphone application that sends a GET request, over HTTP protocol, to an ad-hoc server listening on a predefined URL and port. The GET request contains the GPS coordinates at the moment of the emergency call and an Identification Number.

The second interface is designed for the police force. The officer accesses all the information collected by telemetry and the video stream via a web browser. In this way the officer can know the actual position of the UAV, displayed on a map embedded in the web page. Information about estimated remaining time and distance for mission accomplishment are also made available. In addition, the video streaming from the UAV camera can be used by the police officer to offer assistance to the person in emergency.

3 The cloud platform

Existing cloud robotics platforms [19] implement custom protocols in order to connect robots to containers residing in Virtual Machines. In our implementation we want to maintain the standard ROS protocol for communication between Endpoints instead. In this way users are not forced to create new robot drivers or ”bridge” node between the ROS framework and the containers; but they can simply use standard ROS nodes and interfaces.

3.1 The basic elements

The basic elements composing the robotics platform presented in this paper, are the following (Figure 1):

- **Node (N)**: it represents the "building block" of a platform service. A node is *installed* if it resides in an instance, while if it resides in a service container, it is *started*.
- **Endpoint**: two types of endpoints are possible:
  - **Internal (IE):** it belongs to a node and its purpose is connecting this both to other nodes in the current service container and to an external endpoint (EE) in its container.
  - **External** (**EE**): it belongs to a service container. It is the link to an endpoint belonging to a node residing in another service container. In this way nodes belonging to different service containers can be connected each other.
- **Service Container** (SC): it is in charge of organizing services as a set of connected nodes. It contains (*started*) nodes. As mentioned above, a service container has external endpoint, in order to connect nodes which are *started* in other service containers.
- **Instance**: it is the object where the platform manager (PM) and the elements described previously reside. The instance can be:
  - **Normal (NI)**: it contains Service Containers and (*installed*) nodes.
  - **Simple (SI)**: it does not contain service containers but only (*installed*) nodes. For starting them, the Simple Instance needs to address the Service Container within a Normal Instance.

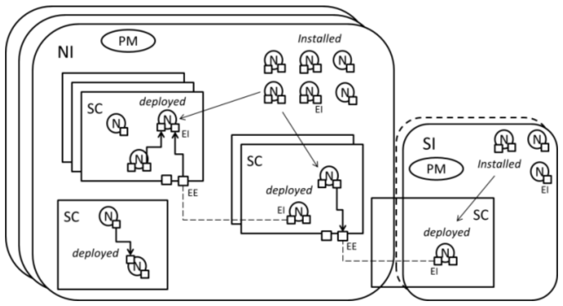


Fig. 1. The platform objects and their relationships

The main purpose of these objects is to build services for enabling robotics applications. These can be enacted by starting nodes (N), which are *installed* in instances (SI, NI), into a service containers (SC). The platform services can be accessed by *service APIs* and can be built by *management APIs*. Metaphorically speaking, connected service containers represent the "remote intelligence" of a robot, the nodes inside them are the "neurons" which need to be connected each other; consequently the couple of endpoints, when connected each other, can be considered the “synapsis” and, finally, APIs are the “senses” enabling the relationships with external world.

3.2 The Platform Manager

The *Platform Manager* is designed to handle the basic objects described before. It:

- sends and receives commands through the *Command Manager*;
- listens and creates events through *Event Manager*, triggering a set of controlled counteractions when an event occurs through *Event Engine*.

In Figure 2 is depicted a logical architecture of Platform Manager. As shown in the figure, events and commands are accessed by the *Platform API Manager*, basically to build and manage platform services.

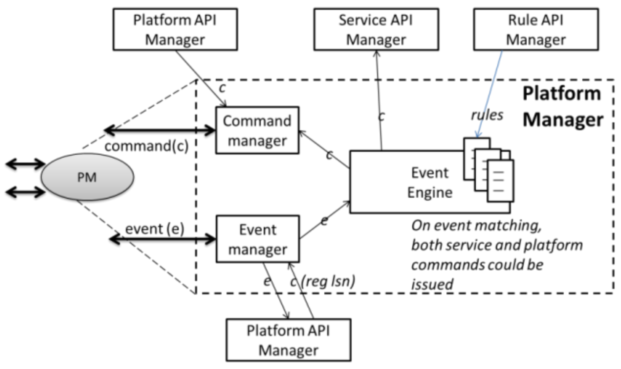


Fig. 2. Platform Manager logic architecture.

The *Event Engine* has been conceived to make a platform service more robust and resilient. The counteractions can be both service commands (e.g., publish a message) and platform commands (e.g., create a service container). The first ones are accessed by *Service API Manager* while the second ones are directly accessed by *Platform API Manager*. The counteractions can be created, read and deleted from users and applications throughout *Rule API Manager*.

3.3 API managers

External elements, such as robotic applications and users, use APIs to access the platform. In particular (Figure 3):

- *Service API Manager*, it is a special node that needs to be *started* in a service container. It exposes APIs to external world for managing the service commands and events.
- *Platform API Manager*, it exposes APIs to external world to manage the platform commands and events.

- *Rule API Manager*, it exposes APIs to external world to manage the counteractions for the event engine.

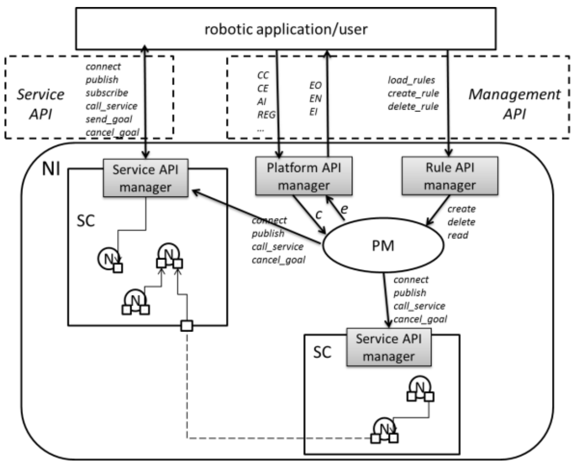


Fig. 3. The logical architecture of API managers

3.4 The cloud oriented Robotic Platform

In this work we implemented the concepts outlined in previous chapters as follows:

- *Normal Instance* (NI): it is a real or virtual machine with high performance.
- *Simple Instance* (SI): it is a real or virtual machine with low performance.
- *Service Container* (SC): it is a ROS container [8] identified by its ROS master. The multi-master robot concert technology [9] enables ROS container multiplicity.
- *Node* (N): it is a ROS node.
- *Internal Endpoint* (IE): ROS topic, ROS service or ROS action.
- *External Endpoint* (EE): ROS topic, ROS service or ROS action of a node in another Service Container.

Figure 4 depicts our implementation of the cloud robotics platform. In the picture sharp edges rectangles stand for NIs, round edges rectangles stand for SCs, dotted edges rectangles stand for SIs. Any small circle represents a ROS node and its endpoints are symbolized as small squares. At the platform level every drone connected to the cloud constitutes a *Normal* or a *Simple Instance*. In the case of Drone 1, the *Normal Instance* contains a *Service Container* running the driver *node* (sending and retrieving data to and from the aircraft); the Drone 2 is represented by a *Simple Instance*, since its service container is shared with the normal instance named "*Virtual Machine".* Finally a third possible configuration is presented with Drone n. In this case the *Service Container* (named Adn) runs in a Gateway (ground station). Here both adapter and driver *node* are placed on the same physical machine.The fourth *Normal Instance* presented in the figure, named "Virtual Machine", cotains both the ODOMI (*Open Data Oriented MIssion planner*) and the Adapter (Ad1 and Ad2) *Ser-*

vice Containers. Adapter SCs translate ROS messages coming from the drones in standard platform messages (i.e. from */sensor_packet_i* endpoints to */sensor_packet* endpoint) and viceversa (i.e. from */drone* EE to */flight_plan_i* endpoints).
ODOMI is the core *Service Container* of the whole platform, and it runs four *nodes*: the rosbridge *node* (the Service API manager described in previous section), the Mission Planner, the Path Planner and the Open Data Driver (described in the next section).

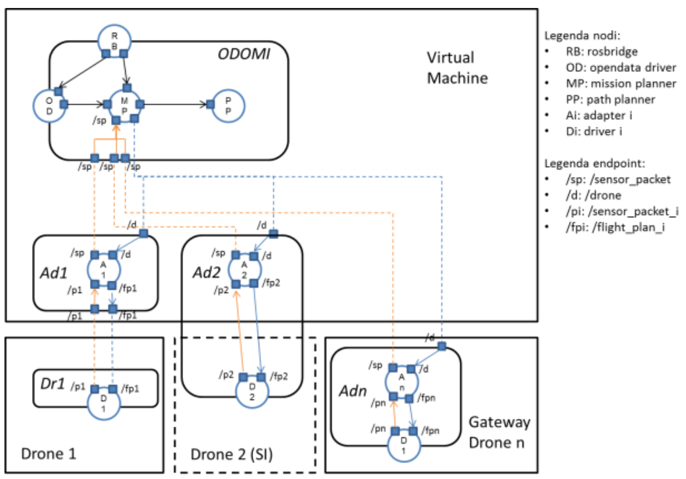


Fig. 4. The Cloud Robotics implementation

Security for the whole platform is assured at the platform gates: API access and instance connection. APIs must be safely accessed after a registration phase, where developers are identified and possibly specific security keys are provided. Instances are connected to the platform via Virtual Private Network in order not to be easily accessed by hackers. Another main requirement is concurrency. ROS framework provides multiple-access management with a specific queue mechanism: any ROS node can either consume a message queue (subscriber), according to computational complexity of consumer algorithm, or produce messages to be queued before sending (publisher), according to available bandwidth.

4 Open Data Oriented MIssion Planner (ODOMI)

As introduced in the previous paragraph ODOMI is the *Service Container* in charge of organizing the mission and calculating the path for the UAV. It is the only *Service Container* that has privileges to START, STOP or ABORT the mission.
It is organized in four *nodes*:

- Open Data Driver (ODD): The module that retrieves and organizes available information about the mission scenario. It retrieves available open data and serialize them in a well-structured and standardized ROS message

- RosBridge (RB): it provides JSON API to ROS interface [10]. This *node* is the interface between the smartphone application and the robotics platform, namely the *Service API Manager* that exposes the APIs for sending the target coordinates that the drone has to reach to the Mission Planner.
- Mission Planner (MP): This module takes as input the position of the drone (home), the target coordinates and the message from ODD module. The MP sends a Bounding Box (BB) to the ODD in order to define accurately the area of intervention according to the initial position of the drone and the position that has to be reached. It outputs a map in the proper format for the Path Planner module.
- Path Planner (PP): It plans the path (set of waypoints) that the UAV has to follow in order to successfully accomplish the mission exploiting the data provided by the MP module. Once defined, the waypoint set is returned to the MP.

The input data for the ODD module are continuously provided by retrieving information on the Internet by calling the appropriate web-service, packaging them in a standard payload independent message on topic */open_data* and forwarding them to the MP module.

The message **Open_data_msgs/Open_data** contains four fields:

- `Int type (static, dynamic):` a *type* field that specifies if the retrieved Open_data are static or dynamic (a data is considered *dynamic* when the information associated to it varies considerably during the mission. e.g. road traffic);
- `String label`: a human-readable label;
- `Parameter[] attributes`: some attributes (e.g. height of the nearest buildings);
- `Polygon[] area` : a finally an array of polygons, where each polygon is intended as a collection of connected geographic coordinates.

4.1 Open Data supported providers

The ODD module is able to parse information coming from several Open Data providers.

Provider	Data	Response	License
OpenStreetMaps	2D Map		Open Data Commons Open Database License (ODbL).
Geoportale Torino	2D Map,	GML(XML)	Creative Commons public licence" Attribuzione - 2.5 (ITALIA).
	Height	GML(XML)	
	Pedestrian areas	GML(XML)	
5T Torino	Traffic	XML	Creative Commons - CC0 1.0 Universal

Table 1.

In addition to data listed in Table 1 some internet applications or websites provide additional geo-referenced data useful to add even more information to the Mission

Planner (Table 2). Since they belong to private companies and are usually subject to license and access restriction, these data cannot be defined “Open” in a strict sense [11], however they are typically retrievable under some limitation (e.g., maximum number of daily/monthly API calls).

Provider	Data Type	Response	License
Google Maps	Digital Elevation Map	XML/JSON	Google Maps API licensing*
OpenSignal	Average RSSI	XML/JSON	OpenSignal API licensing**
	Tower Info	XML/JSON	

* https://developers.google.com/maps/terms
** http://developer.opensignal.com/terms/

Table 2.

4.2 Path planning

Path planning is a well-known problem in robotics, and it has been recently applied to UAVs. UAVs in particular present problems due to their dynamics, three-dimensional environments, disturbed operating conditions. Most of the approaches rely on a two-stage procedure: first they solve the path planning problem, then they use a control loop to follow the found trajectory [15]. Some work has been carried out for path planning in presence of uncertainties and with signal constraints in [12], [13], [14].

Since we do not have to worry about dynamics, we simplify the problem to the 2D case (we slice the 3D city map at a certain height) and apply a well-known state of the art graph search algorithm, A*.

Given an area of operation, we build a cost map where the cost of traversal for each cell depends on the presence of tall structures. In particular, structures taller than a certain threshold are considered obstacles.

We create another cost map based on LTE signal heat-maps. Information about obstacles and LTE signal are gathered by the ODD module. The cost of each cell is based on the RSSI for that area; if the RSSI is below a threshold, the cell is marked as obstacle, since we cannot afford to lose connectivity; otherwise the traversal cost for the cell depends on the quality of the signal.

The two cost maps are merged and a path is found from the current position of the agent to the destination using A*. We then extract a series of waypoints from this trajectory using a simple procedure. For each waypoint we expand a circle around it with the radius corresponding to the nearest obstacle. The next point of the trajectory that lies on the circumference is taken as the next waypoint. The process is then repeated. Figure 5 shows a simulated case, in which the agent has to fly from initial GPS to the target position. The two cost maps have been built by the ODD module automatically. We can see how the trajectory tries to avoid areas with low signal strength.

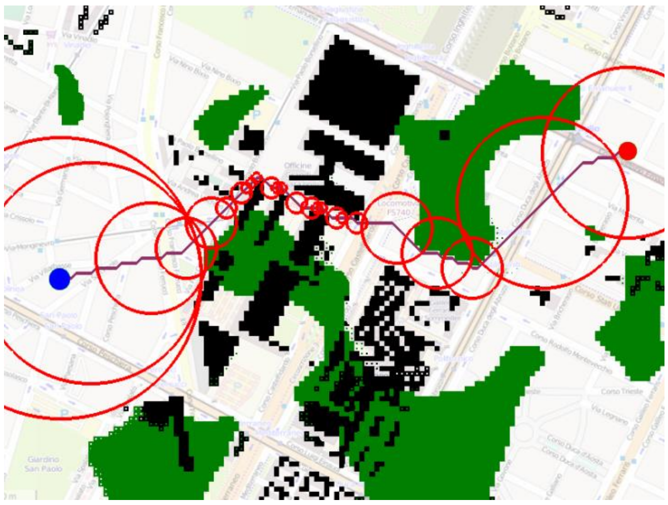


Fig. 5. Left: path creation. Buildings are shown in black; areas with low signal strength are shown in green; starting point is shown in blue, the goal is shown in red. Red circles show the minimum distance from obstacles for each waypoint.

5 Conclusion

In this work we present a service within a cloud robotics platform for emergency management in a smart city environment. In particular we show how an automatic exploitation of Open Data information coming from several heterogeneous internet providers allows a consistent planning of an UAV mission in a smart city scenario. The system has been validated using a simple test case in which the cloud platform manages a request for help, received from a smartphone, sending a drone monitoring a particular area of the city.

Although the architecture presented in this paper supports every ROS-compatible robot, the proposed test case involves the use of different kind of UAVs, in particular quadrotors. Three different products have been selected for the validation of the cloud architecture: Parrot AR.drone [16], Mikrokopter [17] and Micropilot [18]. Table 3 presents an overview of the different architectures. For Mikrokopter and Micropilot 2128 the ROS support has been found to be poor or totally absent. Therefore their ROS interfaces have been written in order to include them in the cloud platform.

This work is part of the project Fly4SmartCity, in collaboration with Telecom Italia S.p.A. and it is currently under development. As future work we intend to test the performances of the platform and to further explore the range of possibilities offered by cloud robotics in a smart city environment, mainly focusing on drone to drone knowledge sharing, remote processing and teleoperation in urban environment.

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

Table 3.

References

1. Mell, Peter, and Timothy Grance. "The NIST definition of cloud computing (draft)." NIST special publication 800.145 (2011): 7.
2. Sanfeliu, Alberto, Norihiro Hagita, and Alessandro Saffiotti. "Network robot systems." Robotics and Autonomous Systems 56.10 (2008): 793-797.
3. Chibani, A., et al. "Ubiquitous robotics: Recent challenges and future trends." Robotics and Autonomous Systems (2013).
4. Chen, Yinong, Zhihui Du, and Marcos García-Acosta. "Robot as a service in cloud computing." Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on. IEEE, 2010.
5. Kamei, Koji, et al. "Cloud networked robotics." Network, IEEE 26.3 (2012): 28-34.
6. Waibel, Markus, et al. "Roboearth." *Robotics & Automation Magazine, IEEE*18.2 (2011): 69-82.
7. G. Ermacora, A. Toma, B. Bona, M. Chiaberge, M. Silvagni, M. Gaspardone, R. Antonini. “A cloud robotics architecture for an emergency management and monitoring service in a smart city environment”, IROS - Cloud Robotics Workshop (2013).
8. Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009. Crick, Christopher, et al.
9. http://www.robotconcert.org/wiki/Main_Page
10. "Rosbridge: Ros for non-ros users." Proceedings of the 15th International Symposium on Robotics Research. 2011.
11. Open Knowledge Foundation Blog. Defining Open Data. http://tinyurl.com/qh722vn
12. Grøtli, Esten Ingar, and Tor Arne Johansen. "Path planning for UAVs under communication constraints using SPLAT! and MILP." Journal of Intelligent & Robotic Systems 65.1-4 (2012): 265-282.
13. Grancharova, Alexandra, and Tor Arne Johansen. "Distributed MPC-Based Path Planning for UAVs under Radio Communication Path Loss Constraints." Embedded Systems, Computational Intelligence and Telematics in Control. No. 1. 2012.
14. "Civil UAV Path Planning Algorithm for Considering Connection with Cellular Data Network." Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on. IEEE, 2012.
15. Goerzen, Kong, Mettler. "A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance", J Intell Robot Syst, 2009
16. http://ardrone2.parrot.com/
17. http://mikrokopter.de/en/home
18. http://www.micropilot.com/
19. Hunziker, Dominique, et al. "Rapyuta: The roboearth cloud engine." *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013.