CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Journal Paper

Dronemap Planner: A service-oriented cloud-based management system for the Internet-of-Drones

Anis Koubâa Basit QureshiMohamed-Foued SritiAzza AllouchYasir Javed Maram Alajlan Omar Cheikhrouhou Mohamed Khalgui Eduardo Tovar

CISTER-TR-190401

Dronemap Planner: A service-oriented cloud-based management system for the Internet-of-Drones

Anis Koubâa Basit QureshiMohamed-Foued SritiAzza AllouchYasir Javed Maram Alajlan Omar Cheikhrouhou Mohamed Khalgui Eduardo Tovar

CISTER Research Centre

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

https://www.cister-labs.pt

Abstract

Low-cost Unmanned Aerial Vehicles (UAVs), also known as drones, are increasingly gaining interest for enabling novel commercial and civil Internet-of-Things (IoT) applications. However, there are still open challenges that restrain their real-world deployment. First, drones typically have limited wireless communication ranges with the ground stations preventing their control over large distances. Second, these low-cost aerial platforms have limited computation and energy resources preventing them from running heavy applications onboard. In this paper, we address this gap and we present Dronemap Planner (DP), a service-oriented cloud-based drone management system that controls, monitors and communicates with drones over the Internet. DP allows seamless communication with the drones over the Internet, which enables their control anywhere and anytime without restriction on distance. In addition, DP provides access to cloud computing resources for drones to offload heavy computations. It virtualizes the access to drones through Web services (SOAP and REST), schedules their missions, and promotes collaboration between drones. DP supports two communication protocols: (i.) the MAVLink protocol, which is a lightweight message marshaling protocol supported by commodities Ardupilot-based drones. (ii.) the ROSLink protocol, which is a communication protocol that we developed to integrate Robot Operating System (ROS)-enabled robots into the IoT. We present several applications and proof-of-concepts that were developed using DP. We demonstrate the effectiveness of DP through a performance evaluation study using a real drone for a real-time tracking application.

Ad Hoc Networks 86 (2019) 46–62

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Dronemap Planner: A service-oriented cloud-based management system for the Internet-of-Drones

Anis Koubâa[a,b,e,h,*], Basit Qureshi[a], Mohamed-Foued Sriti[c], Azza Allouch[g,b,h], Yasir Javed[a,h], Maram Alajlan[a,d,e], Omar Cheikhrouhou[i,j], Mohamed Khalgui[f,k], Eduardo Tovar[e]

[a] Robotics and Internet-of-Things Lab (RIOTU), Prince Sultan University, Saudi Arabia
[b] Gaitech International Ltd., China
[c] Al-Imam Mohammad Ibn Saud Islamic University, Saudi Arabia
[d] King Saud University, Riyadh, Saudi Arabia
[e] CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Porto, Portugal
[f] LISI Laboratory, National Institute of Applied Sciences and Technology (INSAT), University of Carthage, Tunis 1080, Tunisia
[g] LISI Laboratory, Faculty of Mathematical, Physical and Natural Sciences of Tunis (FST), University of El Manar, Tunis, Tunisia
[h] Cooperative Networked Intelligent Systems (COINS) Research Group, Saudi Arabia
[i] Taif University, Taif, Kingdom of Saudi Arabia
[j] Computer and Embedded Systems Laboratory, University of Sfax, Sfax, Tunisia
[k] School of Electrical and Information Engineering, Jinan University (Zhuhai Campus), Zhuhai 519070, China

ARTICLE INFO

Article history:
Received 19 February 2018
Revised 21 July 2018
Accepted 19 September 2018
Available online 20 September 2018

Keywords:
Unmanned aerial vehicle
IoD
MAVLink
ROS
Cloud

ABSTRACT

[1] Low-cost Unmanned Aerial Vehicles (UAVs), also known as drones, are increasingly gaining interest for enabling novel commercial and civil Internet-of-Things (IoT) applications. However, there are still open challenges that restrain their real-world deployment. First, drones typically have limited wireless communication ranges with the ground stations preventing their control over large distances. Second, these low-cost aerial platforms have limited computation and energy resources preventing them from running heavy applications onboard. In this paper, we address this gap and we present Dronemap Planner (DP), a service-oriented cloud-based drone management system that controls, monitors and communicates with drones over the Internet. DP allows seamless communication with the drones over the Internet, which enables their control anywhere and anytime without restriction on distance. In addition, DP provides access to cloud computing resources for drones to offload heavy computations. It virtualizes the access to drones through Web services (SOAP and REST), schedules their missions, and promotes collaboration between drones. DP supports two communication protocols: (i.) the MAVLink protocol, which is a lightweight message marshaling protocol supported by commodities Ardupilot-based drones. (ii.) the ROSLink protocol, which is a communication protocol that we developed to integrate Robot Operating System (ROS)-enabled robots into the IoT. We present several applications and proof-of-concepts that were developed using DP. We demonstrate the effectiveness of DP through a performance evaluation study using a real drone for a real-time tracking application.

1. Introduction

1.1. Overview

Internet-of-Things (IoT) and Cloud Computing have been attracting a lot of interest in the last few years. In this paper, we address the use of Unmanned Aerial Vehicles (UAVs) to develop new Internet-of-Things applications (e.g. remote sensing, smart cities, surveillance, disaster management, patrolling, aerial survey, border security, to name a few) by leveraging cloud computing, web technologies, and service-oriented architecture (SOA). In the recent years, there have been a very few attempts to integrate drones

* Corresponding author.
E-mail addresses: akoubaa@psu.edu.sa (A. Koubâa), qureshi@psu.edu.sa (B. Qureshi), mfsriti@ccis.imamu.edu.sa (M.-F. Sriti), azza.allouch@coins-lab.org (A. Allouch), yasir.javed@coins-lab.org (Y. Javed), maram.ajlan@coins-lab.org (M. Alajlan), o.cheikhrouhou@tu.edu.sa (O. Cheikhrouhou), khalgui.mohamed@gmail.com (M. Khalgui), emt@isep.ipp.pt (E. Tovar).

[1] Video demonstrations and a video presentation providing a summary of the paper can be found at https://goo.gl/EDvxhk.

https://doi.org/10.1016/j.adhoc.2018.09.013
1570-8705/

with the Internet and IoT [1]. In [1], the authors defined the concept of Internet of Drones (IoD) as a layered control architecture that coordinates the access of drones to controlled airspace and that provides navigation services between locations of Interest. The authors proposed an abstract layered architecture and generic services for IoD that encourages openness, modularity and interoperability. It is based on broadcasting drones' information to ground stations located at Zone Service Providers (ZSPs). This work is only limited to a high-level description of the architecture and generic services and discussion of benefits; however, there is no concrete implementation to demonstrate its feasibility, nor performance evaluation study to illustrate the behavior of the system. In the more general context of the cloud robotics, coined by James Kuffner in 2010, Kehoe et al. [2] surveyed cloud robotics architectures and existing works [3–6] that attempt to integrate robot with the cloud through the Internet. Reference [3] is one of the first papers that specified the concept of Robot as a Service (RaaS). In [3], Yinong et al. proposed a cloud framework for interacting with robots in the area of service-oriented computing. In [4], the authors proposed the `DAvinCi` system for offloading computation from robots to cloud-based distributed computing system based on Apache Hadoop, but they did not address reliability and real-time control. In [5], the European project consortium proposed to build a World Wide Web for Robots for sharing knowledge about actions, objects, and environments between robots, and prototypes were implemented for service robots. In [6], the author proposed a SOAP-based service-oriented architecture that virtualizes robotic hardware and software resources and exposes them as services through the Web. However, the two latter works did not consider the specific use case of controlling drones over the Internet, as we propose in this paper.

1.2. Problem statement and motivation

This work garners motivation from the limitations of typical low-cost UAVs having strict processing and storage capabilities constraints. In fact, low-cost and battery-powered UAVs are unable to cope efficiently with the requirements of computation demanding applications (e.g. onboard image processing) encompassing real-time data and reliability constraints. Furthermore, taking into account the limitation of communication range of wireless-based drones even when using telemetry systems (with a range up to 5 Km), it is not possible to manage drones missions in large environments for example at the scale of a city, country or larger. For instance, in [7], the authors developed an interesting networked drone system for different types of applications. However, communication between drones was performed in an ad-hoc manner and managed by a central ground station within each other communication range. This restricts the deployment area of the drone network to the maximum communication range of the ground station. With Dronemap Planner, drones are connected to the Internet and thus there is not any kind of limitation with respect to the communication range. Obviously, the communication quality will become an issue as Internet connectivity may not be stable. This is further discussed in this paper in the experimental evaluation.

On other hand, wireless drone systems typically adopt point-to-point communication models with a ground station. This restricts the scalability of such systems and limits them to be used for managing a single drone. For example, nowadays commodities drones like 3DR Solo drone, Dji Phantom, Erle Copter, etc. use the MAVLink communication protocol [8] to interact with a ground station (e.g. QGroundControl, DroidPlanner, Tower) through a UDP, TCP, Bluetooth or USB network interfaces. These ground stations are standalone systems and allow to control only a single drone by a single user. Other ground stations like Universal Ground Control Station (UgCS) provide multiple drones' support, however, they share the limitation with other ground stations of being standalone applications with no option for sharing among multiple users nor promoting collaboration missions among drones. In contrast, Dronemap Planner has the objective to control multiple drones by multiple users by providing seamless access to drones' resources through cloud services, and dynamically manage the mission of drones.

In addition, Dronemap Planner offers the opportunity for developers to seamlessly develop their own cloud applications using APIs for high-level languages such as Java, Python and Web programming languages, through web services technologies. This provides new alternatives for educational framework on drones programming and applications development. Furthermore, we aim at using the cloud as a "remote brain" for the UAVs by providing computation and storage services remotely [9]. The UAV simply acts as a mobile sensor node and/or sink that collects data from points of interest and transfer it to the cloud, which, in turn, stores, processes and provides interpretation for the collected raw data. In addition, the services provided by the cloud allow clients to connect, coordinate missions as well as initiate commands to the UAVs for future missions.

1.3. Approach and contributions

Our approach leverages the use of cloud computing paradigm to address all the aforementioned requirements. There are two main benefits (1) **Virtualization:** the cloud infrastructure helps virtualizing UAV resources through abstract interfaces. It provides a mapping of the physical UAVs to virtual UAVs, so that end-users interact with virtual UAVs instead of the physical UAVs. This emphasizes the concept of the Internet-of-Drones (IoD) as a specific case of the IoT where the UAVs represent the things connected to the Internet through abstract interfaces. (2) **Computation offloading:** the cloud plays the role of a remote brain for the UAVs by providing storage and computation services [9]. This approach overcomes the computing and storage resources' limitations of the UAVs, as intensive computation is not performed on-board, but rather offloaded to the cloud. As such, the UAV will typically act as a mobile sensor and actuator decoupled from heavy computations. In this paper, we illustrate these two concepts through the design and development of the Dronemap Planner. In summary, the main contributions of this paper are four-folded.

- First, we specify the concept of Internet-of-Drones (IoD), its requirements, security considerations, challenges, and importance.
- Second, we propose Dronemap Planner, a cloud-based management of drones connected through the Internet with cloud integration.
- Third, we present the software architecture of Dronemap Planner.
- Fourth, we demonstrate the feasibility and effectiveness of the Dronemap Planner system in handling drones' missions using several experimental use cases and applications to control drones over the Internet and evaluate its performance.

Parts of this work was published in our conference paper [10].

2. Related works

There are several attempts in the literature to integrate drones with the cloud and IoT. A conceptual model for the Internet-of-Drones was presented in [1]. The proposed layered architecture covers three major networks: cellular network, air traffic control network and Internet. Several general services were provided for different UAV applications, namely surveillance, delivery, search and rescue. The paper did not present any implementation or realization of this architecture and only outlined general concepts of

the IoD. In our paper, we present both an architecture for IoD and its implementation which are validated through simulation and real world experimentation.

In [11], the authors proposed a SOA model for collaborative UAVs. A mapping between cloud computing resources and UAVs' resources was presented. Furthermore, essential services and customized services were proposed. The paper only provides a high-level description of the system architecture, components and services without any specific details on their implementation. The same authors extended that work in [12] and designed a RESTful web services model by following a Resource-Oriented Architecture (ROA) approach to represent the resources and services of UAVs. In addition, a broker that dispatches mission requests to available UAVs was proposed. The broker is responsible for managing the UAVs, their missions and their interactions with the client. A small prototype was implemented on an Arduino board that emulates a UAV and its resources. However, the experimental prototype is very limited as it does not demonstrate sufficient proof of concept on real drones, but on a simple Arduino board. Thus, the feasibility of the approach was not effectively demonstrated.

In [13], an experimental testbed for an emulated Arduino UAV system was used, and several sensors were used (including temperature and humidity, ultrasonic for distance measurements). RESTful web services were defined and implemented for manipulating each type of sensor through a Web interface. The authors evaluated the performance of their system. However, the experimental system remains limited in terms of validating the scalability issues, and also the experimental setup only applies to a small local network. In our paper, we consider real drones communicating with the MAVLink protocol [8] to validate our architecture.

Bona, Basilio [14] presented a cloud robotic platform called as FLY4SmartCity that is based on ROS. The proposed architecture contains basic features to create instances of drones as nodes where they are handled by platform manager in terms of planning and event management. The platform manager is supported by the service manager for the provision of services in case of events, while the rule manager handles the actions.

The work in [15], presents a cloud robotics platform for emergency monitoring based on ROS. It allows leveraging the advantages of the cloud to offload the data and computational capabilities. The layered architecture provides services built on API provided by applications built of drone capabilities and adaptation. Drones form the physical layer of the architecture.

Authors in [16] presented an IoT architecture named "VIRTUAL RESOURCES", which allows the developers to relocate slices of the application to any intermediate IoT device through the concepts of virtual sensor and virtual actuator using a RESTFUL interfaces. VIRTUAL RESOURCES provide several benefits to developers, including better utilization of network resources that results in higher energy efficiency and lower latencies, a simplification of the application logic at the Cloud, and better separation of concerns throughout the development process.

Aiming at making the drones available for surveillance round-the-clock, the work in [17], proposed an IoT-based command and control mechanism along with automatic landing system. The system composed of four types of software: 1) main console server and controller, 2) communication relay for drones, 3) firmware of the landing-field to control active gripper and wireless charging, 4) mobile users application which receives the drones video from the server in real-time. The drones can communicate directly via the internet with the main server and each of them has a low-cost LTE module. The server displays the received videos from the drones to the administrator and provides the image to the users mobile device at the same time.

The work in [18], proposed new bi-directional packet oriented communication protocol between the UAV and the base station named "UranusLink". UranusLink is designed for only small data flow such as control commands to the UAV and telemetry data from it. Also, it provides both unreliable and reliable transfer mechanism that allows secure connection and packet loss detection.

In the context of monitoring crops in agriculture, the work in [19] presented a vision-based technique that adapts low-cost commercial camera and can be used with rotatory wings or fixed-wing drones for visual analysis of the soil to recognize the fields plowing type. To achieve high-fidelity, the classic sensor fusion technique involving GPS integrated with Inertial Navigation Systems (INS) localization estimation adapted to be used along with standard Extended Kalman Filter solution for navigation.

We alert the readers who are not aware of the MAVLink and the ROSLink protocols that a detailed background description of both is available in the appendices of this paper.

3. Internet-of-Drones

3.1. Overview

The Internet of Drones (IoD) can be defined as an architecture for providing control and access between drones and users over the Internet. In fact, Drones are increasingly becoming commodity items widely available off the shelf, thus allowing their use by any user to fly various missions using multiple drones in a controlled airspace. Whereas technology is helping the miniaturization of a UAV's onboard components including processors, sensors, storage as well as improving the battery life, the limitations of these components hinder the performance and lower the expectations. IoD provides a vehicle for coupling of Internet of Things as well as cloud robotics technologies to allow remote access and control of drones as well as the seamlessly scalable computation off-loading and remote storage capabilities of the Cloud.

There are various challenges associated with the implementation of IoD. Reliable point-to-point communications, mission control, seamless wireless connectivity, effective utilization of onboard resources are just a few of the concerns. Furthermore, Quality of Service (QoS) stands-up as a crucial issue that must be considered in the design of the IoD. Security is also an important challenge as access to drones' resources must be authenticated and secured. In addition, the IoD system must be immune to attacks like drone impersonation, flooding, sniffing, etc. Another important aspect is to hide the underlying technical information from the user which is possible by using a service-oriented approach, typically implementing SOAP or REST Web services [20]. Users do not need to be technically savvy in order to program or develop missions, rather the web services based system would provide easy access to onboard resources through various APIs.

3.2. Objectives

In this paper, Dronemap Planner provides a solution to seamlessly accessing drones resources over the Internet and control their mission. We consider Dronemap Planner as a platform allowing users to access and control multiple drones over the Internet. Using a web based interface, a user can deploy a virtual drone with specific tasks including, (1) assigning a mission with a planned path over multiple waypoints, (2) allocating tasks at various waypoints including but not limited to capturing images, weather sensing information, etc., (3) transmitting captured data to the cloud for further processing in addition to (4) continuously updating the location and status of the drone to the user. At the time of mission initiation, each virtual drone is mapped to a physical drone available in a warehouse, which connects to the Dronemap Planner and performs the associated tasks. Using the Dronemap

Planner, several virtual missions can be planned ahead of time for a limited number of physical drones available. The IoD provides a platform for various applications in security, surveillance, disaster recovery, etc. and promises efficient use of limited resources available.

The main objectives of an IoD management system, like the Dronemap Planner that we propose in this paper, include:

- to provide seamless access to and real-time control and monitoring of drones for end-users
- to offload extensive computations from drones to the cloud
- to schedule the missions of multiple drones dynamically
- to provide a collaborative framework for multiple drones
- and to provide cloud-based programming APIs for developers to develop drones' applications through the cloud.

3.3. Motivating scenarios

To motivate the need for IoD and in particular for the Dronemap Planner cloud-based system, let us consider the following illustrative scenario: a team of multiple autonomous UAVs deployed in an outdoor environment in their depot waiting for the execution of certain missions. A user behind the cloud defines a mission (e.g. visiting a set of waypoints) and requests its execution. The user may either select one or more virtual UAVs from the list of available UAVs registered in the cloud, or may send his request to the cloud to auto-select one or more UAVs to execute the mission. Each virtual UAV is mapped to a physical UAV by the cloud using a service-oriented approach, typically implementing SOAP or REST Web services [20]. Once the mission request is received, the selected UAVs execute the mission and report in real-time the data of interest to the cloud layer, which in turn will store, process and forward synthesized results to the user.

Another use case is when the user selects locations of interest to visit on the map and sends them to the drone. By default, the drone will execute the mission by visiting the locations according to their sequence number. However, this might be not the optimal way to visit the waypoint. The problem becomes even more complex when there is a need to optimally assign multiple locations to multiple drones. By default, the drone will execute the mission by visiting the locations according to their sequence number. The sequence number is typically defined by the order of the waypoint selection of the end-user. However, this order might not be optimal in terms of energy, time or traveled distance. This problem can be seen as the traveling salesman problem (TSP), where an optimal tour must be determined to optimize the cost of the mission. This problem is known to be NP-Hard and the computational time grows exponentially with the number of target locations. Definitely, the execution of the TSP algorithm on the drone is not efficient, as low-cost drones have stringent computation capabilities and thus cannot execute such time-consuming algorithms. In addition, low-cost drones operate on batteries, thus, the execution of such resource-demanding and power-consuming algorithms will quickly consume available energy. The problem becomes even more complex when there is a need to optimally assign multiple locations to multiple drones, which maps to a multiple traveling salesman problem (MTSP) even harder than the TSP.

So, a cloud-based system will definitely help in offloading such extensive computation from the drone to optimize the mission execution, and thus extending the energy lifetime of the drone.

Furthermore, consider a mission being executed by multiple drones, where one or more drones fail to complete visiting their locations of interest. In this case, it is important to deal with such a fault and to make a recovery plan. Using a cloud-based management system, it is possible to detect these kinds of faults and to act accordingly by re-scheduling the missions of the drones in real-time to cover the waypoints not visited by the faulty drones. The same situation of re-scheduling happens when a new event occurs and requires the visit of new locations of interest by the drone. Using a cloud-based management system has the benefit to promote a collaborative environment between the drones through the cloud system so that to dynamically update their missions in real-time based on current status of the system.

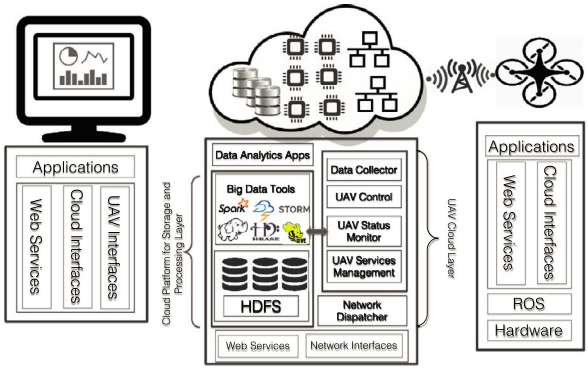

Fig. 1. DroneMap Planner system architecture: abstraction layers.

Dronemap Planner considers scalability of service allowing the system to scale with increased number of UAVs generating increasingly large sets of data. The use of cloud computing infrastructure will allow meeting this requirement by adopting an elastic computing model. Additionally, the correct operation of the integrated cloud-UAV system does not only depend on guaranteeing message delivery, but equivalently important is the delivery of messages within bounded end-to-end time frames. Excessive delays or jitters on the message may result in delaying decision making and subsequent actions on critical events, adversely affecting the QoS. Thus, QoS stands-up as a crucial issue that must be considered in the design of the integrated cloud-UAV system.

4. Dronemap Planner architecture

In what follows, we present the general system architecture, then the software architecture of Dronemap Planner, and we discuss in details its different components. For background information about the MAVLink and the ROSLink protocols, the reader may refer to the appendices of this paper.

4.1. General system architecture

Fig. 1 presents the architecture of Dronemap Planner cloud system addressing the above functional and non-functional requirements.

- **The UAV Layer:** The UAV represents a set of resources exposed as services to the end-user. The UAV has several layers of abstractions. On top of the hardware, ROS and MAVLink provide both two different means as the first layer of abstraction that hides hardware resources (i.e. sensors and actuators). Robot Operating System (ROS) is one of the widely used middleware to develop robotics applications and represents an important milestone in the development of modular software for robots. In fact, it presents different abstractions to hardware, network and operating system such as navigation, motion planning, low-level device control, and message passing. On the other hand, MAVLink is a communication protocol built over different transport protocols (i.e. UDP, TCP, Telemetry, USB) that allows to exchange pre-defined messages between the drones and ground stations, which provide a high-level interface for applications

developers to control and monitor drones without having to interact with hardware. These two alternatives allow software developers to focus more on the high-level development without having to deal with hardware issues.

- **Cloud Services Layer**: Three sets of components are defined:(1) *Storage components:* This set of components provides storage for streams of data originated from UAVs and captured by this layer. Each UAVs environment variables, localization parameters, mission information, and transmitted data streams including sensor data and images with time-stamps are stored in the cloud either in regular SQL database or in a distributed file system (i.e. HDFS, NoSQL database such as HBase), depending on the application's requirements. Storage in distributed file systems helps to perform large-scale batch processing on stored data using tools like Hadoop Map/Reduce.
  There are two types of data processing on cloud computing infrastructure: (*i.*) *Real-time stream processing*: the cloud processes incoming streams of data for detecting possible critical events or threats that require immediate action or performs dynamic computation in a distributed environment. Examples of real-time stream processing could include processing sample images received from the UAV to detect possible threat (i.e. intruder into an unauthorized area) or also processing sensor data (e.g. high temperature value in case of fire). Another possible application real-time processing might be required when new events are detected and require the dynamic re-scheduling of drones' missions to ensure the optimality of the missions' executions after considering the new events. (*ii.*) *Batch processing*: Incoming data is stored in the HDFS distributed file system for increased reliability as well as post-processing using a distributed parallel computing approach. Batch processing can be used to look for particular events into the log file, for example, how many intruders detected in an unauthorized area over a certain period of time. The cloud services layer implements a cluster of compute nodes running `Hadoop HDFS`. All data is stored in various tables using NoSQL in Hbase. (2) *Computation components:* Various computation intensive algorithms are deployed in the cloud. Image processing libraries process stored data available in HBase to detect possible event. In addition, Map/Reduce jobs running on the Yarn cluster allow applications to run in parallel reducing the processing time, therefore improving performance. Additionally, Data Analytics algorithms can be executed on the stored set of large scale data. (3) *Interface components:* We defined three sets of interfaces as part of this component. (*ii.*) *Network interfaces* implement network sockets and Websockets interface on the server side. These provide listening to JSON serialized messages sent from UAVs. In particular, Websockets are the most appropriate protocol to reliably handle streaming applications. In the context of Dronemap Planner, MAVLink messages are received from the drones through network sockets (UDP or TCP), and then forwarded to the client application through Websockets. The reason behind this strategy design choice is that Websockets are supported by all programming languages (e.g. Java, Python, C++) including Web technologies. The use of UDP or TCP sockets for streaming to the client will induce more restrictions to the development of Web clients applications. (*ii.*)*The Web Services interfaces* allow clients to control the missions of the drones and their parameters. Both SOAP and REST Web Services are used to provide the end-users and clients applications different alternative to control and monitor the drones through invocation of Web Services. While network interfaces are used to mostly handle continuous streams, Web Services are used for sending control commands to the drones and getting information from the cloud.

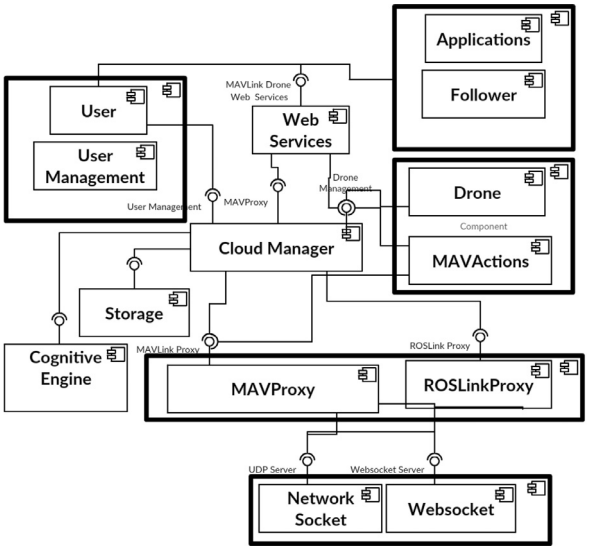

Fig. 2. Dronemap Planner software architecture: component diagram.

- **Client Layer**: The client layer provides interfaces for both end-users and drones' applications developers. For end-users, the client layer runs `Dronemap client` side Web applications, which provide interfaces to the cloud services layer as well as the UAV layer. Users have access to registering multiple UAVs, defining and modifying mission parameters and decision making based on data analysis provided by the cloud. The application allows users to monitor and control the UAVs and their missions remotely. Front-end interfaces provide the functionalities to the user to connect/disconnect, use available physical UAVs and their services, configure and control a mission and monitors the parameters of the UAV. For developers, the client layer provides several APIs for different programming languages to easily develop drones' applications and interact with their drones.

4.2. Software architecture

In this section, we present the Dronemap Planner software architecture. We adopted a modular component-based software promoting, where components are loosely coupled and each component implements a specific behavior of the application. In our architecture, we refer to *agent* as a drone, user or a cloud.

4.2.1. Architecture Components

Fig. 2 shows the component diagram of the software architecture. The software system is decomposed into four main subsystems (or layers), each of which contains a set of components. These subsystems are:

- **Communication:** This subsystem represents the basic building block for network communications between the drones, users and the cloud. There are two main components, namely (*i.*) Network sockets and (*ii.*) Websockets. On the one hand, Network sockets allow agents (drones, users, and cloud) to exchange JSON serialized messages between each other through the network interface using sockets. The use of JSON message format is beneficial for interaction between heterogeneous systems as it is platform-independent and less verbose than XML. On the other hand, Websockets interfaces are used to handle data streaming between the cloud and the user applications. As explained above, we opted for the use of Websockets technology because it is supported by different programming languages including Web technologies.

- **Proxy:** This layer acts on top of the communication layer and incorporate all the protocol-related operations including message parsing, dispatching, and processing. This layer supports two protocols. The first protocol is the MAVLink communication protocol, which is the de-facto standard for the communication between ground stations and drones. The MAVLink protocol is based on binary serialization of messages and operates on different transport protocols, namely, UDP, TCP and serial. The second protocol is the ROSLink protocol, which is a proprietary protocol that we developed to ensure interaction between a robot and a ground or control station. ROSLink is very much inspired from the MAVLink protocol in the sense that it implements a client in the robot that keeps sending JSON serialized messages to the ground station.
- **Cloud:** The cloud layer is responsible for managing all the computing, storage and networking resources of Dronemap Planner. It is composed of four components, namely (*i.*) Cloud Manager, (*ii.*) Storage, (*iii.*) Web Services components and (*iv.*) Cognitive intelligence. The central of the cloud layer is the `Cloud Manager` component, which orchestrates all the processes in Dronemap Planner and links all other components together. It uses the interfaces provided by `MAVProxy` and `ROSLinkProxy` components, in addition to the storage component. On the other hand, it provides interfaces to the Drone and Users components, so that they do have access to `MAVProxy`, `ROSLinkProxy` and `Storage` components. The main role of the `Storage` component is to provide interfaces to store data in different storage media including SQL/NoSQL databases and distributed file storage, i.e. HDFS. Different types of data need to be stored, retrieved and accessed. For example, SQL databases may be used to store information about users, and their credentials, or also information about drones and their missions. NoSQL databases (e.g. MongoDB) are used for more unstructured data such as data collected from the drones' sensors for further analysis. HDFS storage can be used to store data that requires further batch processing using distributed computing techniques, like Map/Reduce. For example, data related to drones' missions can be dumped from SQL or NoSQL databases to HDFS to process it either with batch processing system like Map/Reduce or real-time processing systems like Storm, and extract useful information for dumped data.
  The `Cognitive Engine` (CE) component aims at performing computations on cloud data to reason, plan and solve problems using artificial intelligence techniques. For example, the CE component may include algorithms for assigning multiple targets locations to multiple drones or to a single drone to optimize their missions. This is known as an instance or the typical traveling salesman problem (TSP). Another example would be to process received images or sensor data from drones using real-time processing systems (e.g. Apache Storm) to detect possible events or threats. In general, the CE component will contain intelligent applications to provide smart functionalities and reasoning.
  The `Web Services` (WS) component is the main interface between the Dronemap Planner cloud and the client applications (i.e. users). It provides platform-independent interfaces to end-users and leverages the use of the service-oriented architecture (SOA) paradigm. Both SOAP and REST Web Services are defined. The REST API was designed to allow accessing cloud public resources through simple HTTP requests. The SOAP API was designed for a more formal and structured service-orientation for remote procedure invocation, which is basically used to send commands to the drone from the client application. There have been long discussions about the pros and cons of REST and SOAP Web Services and the reader may refer to [20] for more details. In our architecture, we opted for providing both types of Web Services as interfaces with end-users to give more flexibility to users.
- **Drone:** The `Drone` subsystem contains all information related to drones and actions that could be performed on them. The `Drone` component represent *resource* in the Dronemap Planner cloud. This resource is basically accessed by client applications through Web Services. In addition, the `MAVAction` component represents all the MAVLink protocol actions that could be executed on the drone including taking-off, landing, waypoint navigation, getting waypoints list, changing operation mode, etc. The `Drone` component maintains the status of the drone, which is updated whenever a new MAVLink message is received. In addition, it provides an interface to access and modify the parameters of a drone. Note that the cloud manager maintains a list of drones into a map data structure, as mentioned above.
- **User:** The `User` subsystem contains information about users that access the Dronemap Planner cloud. Each user should be registered to the Dronemap Planner cloud to have access to drones based on his profiles and privileges. A user might be able to control a single drone, or multiple drones or all drones based on his privileges. The mapping between drones and users is made through the Cloud Manager during the registration of the user to the system, and based on approval of the cloud administrator. There are different possible strategies of mapping between users and drones, namely: (*i.*) Single User / Single Drone, where one user is allowed to access and control a single physical drone, (*ii.*) Single User/Multiple drones, where one user is allowed to access and control multiple physical drones, (*iii.*) Single User / Virtual Drone(s), where one user is not allowed to control a physical drone, but sends its request to the cloud, which will decide on which drone(s) to execute the mission of the user. Each user should have an access key that allows him to access a certain drone resource over the cloud or to develop applications for a particular drone resource. The access to drone resources on the cloud is given to the users either through SOAP and REST Web Services to execute commands, or through Websockets to receive drones' MAVLink data streams.

4.3. MAVProxy/ROSLink

The MAVProxy is responsible for (*i.*) processing MAVLink related messages received from the drones, (*ii.*) dispatching messages to users through the Websockets protocol, (*iii.*) updating the information of drones objects of the `Cloud Manager`. It is a multi-threaded server that was designed to effectively handle MAVLink data streams and messages. At the reception of a MAVLink message, a new thread will be created to process that individual message and extract related information, depending on the message type.

The `ROSLinkProxy` has the same functionalities as MAVProxy, but it processes ROSLink messages instead of the MAVLink message.

If the message received is the `Heartbeat` message for a new drone (first message received from a drone), the MAVProxy adds this drone into the map data structure of drones maintained in the cloud manager. In the drones' map structure each drone is identified by a key that is composed of (*i.*) the IP address of the drone, (*ii.*) the port number, (*iii.*) and its system ID. This composite key will make sure that every drone is identified uniquely in the cloud, as it might happen that two drones of different users use the same system ID. In fact, the system ID is encoded in 8 bits only which allows a total of 255 different system ID for drones, which limits the scalability of the system, or will induce conflict in recognizing a particular drone using the same system ID of another drone. The use of IP address and port number will definitely eliminate

this problem. However, the main drawback of this technique is that the same drone will have a different key in the drones' map each time it connects as it will likely have a new different IP address and/or port number. If the drone already exists, the MAVProxy updates the drone's attributes and parameters in the drones' list. In fact, MAVLink messages periodically carry information about different drone's attributes including GPS location, heading, battery level, altitude, air speed, ground speed, etc. Thus, every type of MAVLink message will update a particular set of attributes of the corresponding drone of the Cloud Manager.

In addition to processing messages, MAVProxy forwards incoming messages received from its network interface to the Websockets interface, which in turn forwards the message to the corresponding client, with which a connection is open. When a client connects onto the Websockets server of the MAVProxy, a new session is defined and messages related to the drone of interest will be forwarded to the client through that open session. The use of Websockets is very useful in reliable streaming of MAVLink packets between the MAVProxy and the clients. Not only it provides a unified interface for different programming languages, but also Websockets have the advantage of handling clients' sessions effectively, natively supported in the Websockets protocol.

4.4. Cognitive Engine

The Cognitive Engine embeds the intelligence of the cloud of drones. It is the core component that reasons, plans and solves problems. It can be seen as a big "remote brain" for the drones. At abstract level, the CE is responsible for (*i.*) performing intelligent mission planning for drones, (*ii.*) extensive computations on the cloud, like image processing, (*iii.*) real-time monitoring of events, like critical events detections with Apache Storm, (*iv.*) batch processing and data analytics on logged data.

In what follows, we present the design of a dynamic mission planner (DMP) application and how it is integrated as a module in the Cognitive Engine. Consider the case of multiple drones to be assigned to multiple target locations (i.e. waypoints) to be visited and then go back to their original locations. This problem is an NP-Hard problem and is known as the multiple depot multiple traveling salesman problem (MD-MTSP) and several approaches were proposed in the literature to solve it [21,22]). The Cognitive Engine provides the user with the service that determines the most optimal allocation of waypoints based on his requirement, while implementing one or some of the solving algorithms on the CE component of the cloud. Current mission planners such as QGroundControl, DroidPlanner, Tower and Universal Ground Control Station (UgCS) do not incorporate a similar intelligence functionality. Since the cloud has global knowledge of the drones characteristics including their locations, battery levels, speed, and the set of waypoints to visit (received from the user), the dynamic mission planner can determine the optimal assignment for such a configuration.

More importantly, the dynamic mission planner incorporates fault-tolerance and dynamically prepares a recovery plan in case of a failure in executing the mission. By keeping track of drones' missions during execution, the dynamic mission planner detects any anomaly and acts accordingly. For example, a drone may crash and run out of energy during the execution of its mission, this requires a real-time re-planning of the missions of the other drones to visit the waypoints assigned to the failing drone and not yet visited, and then the new waypoints assignments will be sent to drones.

5. Security consideration

In this section, we will present security threats for Internet-of-Drones that can affect Dronemap Planner cloud-based drones management system and we will highlight possible solutions to them.

Table 1 summarizes the identified security threats against our Dronemap Planner system, with the corresponding countermeasures techniques. When drones are connected through Internet, they are exposed to a multitude of threats and attacks that may target the drone, the flight controller, the Ground Control Station GCS, the wireless data link or any combination of them [23]. Several risk factors that target the data integrity, authentication, network availability and information confidentiality can complicate their operations and can further prevent the accomplishment of their missions [24].

Dronemap Planner cloud involves different components. Thus, attacks can occur at different layers: (1) The Proxy Layer, (2) The Cloud Layer, (3) The Drone Layer.

5.1. Proxy layer

The Proxy Layer in the Dronemap Planner software architecture supports the MAVLink protocol. It allows the exchange of predefined messages between the drones and ground stations. The MAVLink Protocol does not provide any kind of security. There is no confidentiality, nor authentication mechanisms. The protocol communicates with drones over an unauthenticated and unencrypted channel. Anyone with an appropriate transmitter can communicate with the drone. This means it is possible to inject commands into an existing session. Thus, the MAVLink protocol is exposed to different attacks, including:

- **Spoofing**: sending a false wireless control command, using the data link, that appears to be from the drone or from the ground control station. The attacker blocks the communication between the UAV and the GCS and begins commanding the drone herself. In order to mitigate the risk of C2 data link spoofing, mutual authentication is important.

- **Jamming**: An adversary disables the reception of control signals from the ground control by the drone. The communication between the drone and the GCS is blocked and the aircraft to go into a lost link state. Implementing fail-safe mechanisms can help mitigate the risks.

- **Data interception**: the signal messages are able to be read by unauthorized parties. The data collected by drones is transmitted back to the user or the cloud. Since the connection used is not always secured. A hacker can easily intercept and steal the data. Authentication and encryption should be used on the link to mitigate this risk and to guarantee the confidentiality and integrity of the exchanged data.

In fact, the MAVLink protocol is not secured and can be hacked quite easily. It is crucial to design secure mechanisms for authentication and encryption of MAVLink data streams to avoid harmful attacks.

5.2. Cloud layer

The Cloud Layer (refer to Section VI.B) contains several components. (*i.*) *Storage component* provides storage services for streams of data originated from UAVs. Data is stored in the cloud either in regular SQL database or in a distributed file system (HDFS, NoSQL database). Database injection attacks can target the cloud layer.

- **SQL injection**: Hackers exploit the vulnerabilities of these databases and inject a malicious code in order to bypass login and gain unauthorized access to backend databases. If successful, hackers can manipulate the contents of the databases, retrieve confidential data, remotely execute commands, or even take control of the web server.

Table 1
Security threats against Dronemap Planner and countermeasures.

Categories	Threats	Countermeasures
Attacks on Confidentiality	Eavesdropping	Encryption of the data link, AES
	Man-in-the-middle attack	End-to-end encryption
Attacks on Integrity,availability	GPS spoofing and jamming attack	Anti-spoofing techniques,Anti-jamming antenna
	C2 Data link jamming	Fail-safe mechanisms, authentication
	Denial of service	Anomaly-based intrusion detection systems, fail-safe mechanisms
	C2 Data link spoofing	Mutual authentication, (PKI) certificates
Attacks on availability	Obstacles and civic challenges	Collision avoidance techniques, air traffic management
	Environmental conditions	AI approaches
Attacks on Confidentiality,availability,authenticity,integrity	Vulnerabilities of MAVLink	Authentication and encryption mechanisms
Attacks on integrity	Hijacking	Hash functions, MAC, Authentication, encryption
Attacks on authenticity	UAV spoofing or jamming	Authentication and encryption of the data, password
Attacks on confidentiality and integrity	C2 data link interception	Authentication and encryption

- **NoSQL injections** target "big data" platforms. These platforms are vulnerable to threats because of the lack of authentication, no support for transparent data encryption and no support for secure communications. Data is sent in clear and could easily be sniffed on the network.

(*ii.*) *Web services (WS) component* (REST and SOAP) allow clients to control the missions of the drones and their parameters. Web services are also potentially exposed to underlying vulnerabilities including: DOS attack, Message monitoring and copying, Message source spoofing.

5.3. Drone layer

Attacks on drones can potentially lead to taking control of or crashing them. Examples of attacks are:

- **Physical challenges:** Drones are vulnerable to a multitude of physical threats that can complicate their operation and further prevent the accomplishment of their missions. Their presence supposes an extra risk that should be considered. The interference (animals, humans), the environmental conditions (wind, temperature) affect the operation of UAV and may cause accidents. Dynamic obstacles or the presence of civic constituents such as trees, electric cables, and buildings are important examples of such challenges. These threats can be evaded using technology for recognizing adjacent air traffic and collision avoidance techniques.

- **GPS spoofing attacks**: The basic idea in GPS spoofing is transmitting fake GPS signal to the control system of the drone, usually more powerful than the original GPS signal coming from the satellite. As a result, the victim might use the faked signals instead of the original ones. GPS Spoofed signals are providing different wrong location to the drone and let it changing its trajectory in order to mislead the drone. GPS enables a drones navigation, and due to non-encryption of the signals they can be easily spoofed, which directly influences the operator commands. This can possibly result in a drone crash. Anti-spoof algorithms can help mitigate GPS spoofing attacks.

- **GPS jamming attacks:** A drone which uses GPS could be attacked by jamming the GPS signal, making it unable for the drone to fix its position. Jamming aims to disrupt all communication. Anti-jamming antenna selection and orientation can help mitigate jamming attacks.

- **Eavesdropping:** A successful man-in-the-middle attack against a drone allows an attacker to know all of the commands sent from the GCS to the drone, and enables the attacker to monitor all telemetry sent by the drone because of the violation of the confidentiality of the drone.

- **Hijacking:** A successful man-in-the-middle attack against a drone would also enable an attacker to transmit unauthorized commands to the drone and take control of it from the GCS (i.e., hijacking) due to the integrity violation of the drone.

- **Denial-of-Service:** A Denial-of-Service (DoS) attack against a drone results in the UAV becoming unresponsive to the GCS, or vice versa, due to the violation of the system's availability.

Finally, these attacks can be classified into four general categories: Interception (Attacks on confidentiality), Modification (Attacks on integrity), Interruption (Attacks on availability) and Fabrication (Attacks on authenticity). Interception can be achieved by eavesdropping on channels. This facilitates traffic analysis and disclosure of message contents. The Dronemap planner system should employ mechanisms to mitigate unauthorized disclosure of the telemetric and control information. Different encryption standards such as AES [25] can be used for encryption of the data link. Modification means replacing an original message to a particular service with a modified one. Integrity can be checked by the use of hash functions, message authentication code and Authenticated encryption cryptographic primitives [26]. Interruption means that a message from/to a particular service is blocked. It includes Routing attack, Channel jamming, Denial of service. Countermeasures against these type of attacks are Strong authentication and incident detection and reporting mechanisms [26]. Fabrication includes message forgery, UAV spoofing, and base station spoofing. Authenticity is ensured by the use of passwords, which may involve a certification agent [24].

6. Applications with Dronemap Planner

Dronemap Planner supports different types of applications of drones/robots over the Internet. We have developed several real-world applications with Dronemap Planner to demonstrate the effectiveness of DP in enabling new IoT applications.

For all applications that follow, DP was deployed into a DreamCompute cloud instance provided by DreamHost service provider. It is also possible to deploy it in any other cloud or public IP server like Amazon AWS or Azure. We used a minimal instance of the DreamCompute with 80 GB of storage, 4GB of RAM, and 2 Virtual CPU. As Dronemap Planner is launched, all of its services are deployed and are accessible from a public IP address with at a predefined port number.

6.1. Web-based control and monitoring of MAVLink drones

The primary objective of Dronemap Planner is to control drones and monitor them through the Internet. A video demonstration of Web-based drone mission control and planning is available in [27].

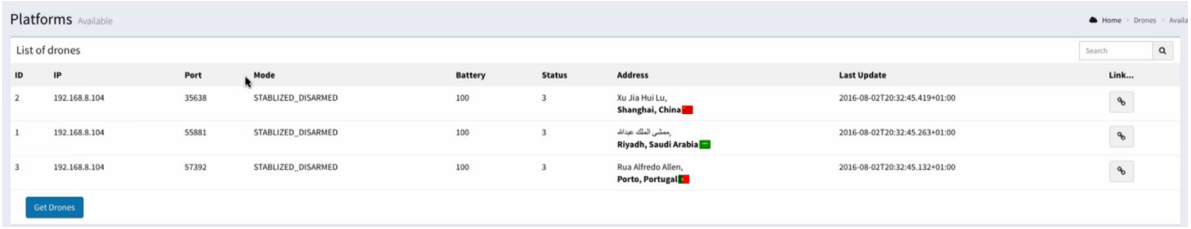

Fig. 3. List of drones on web ground station.

We developed a Web client interface that interacts with DP using Web Services and Websockets interfaces. Web Services calls were used to invoke remote services on the cloud, like getting the list of available drones connected to the cloud. Fig. 3 presents the list of active drones for an administrator user. The figure shows three active drones that the user can control and monitor. The drone list interface provides the user with information of every connected drone including the IP address, port number, and the MAVLink System ID. In addition, the physical address of each drone is shown based on its GPS coordinates. Google Geolocation Web Services API were used to map the physical location addresses with GPS coordinates. The following video demonstration [28] shows the drones' automatic detection and removal feature of DP. When a drone connects to the Internet, it is automatically displayed in the web interface that periodically invokes the SOAP web service method `getDronesList()`. On the cloud side, the `Watchdog` process is keeping track of connected drones by checking their `heartbeat` messages. If a certain number of consecutive `heartbeat` messages do not reach the cloud from a drone, the latter is considered as disconnected and is automatically removed from the drone list. The time interval for listening to consecutive lost `heartbeat` messages is set by the DP administrator in the DP configuration file to decide about whether the drone is still active or disconnected. We used 5 seconds in our experiments.

Once the user connects to a drone from the list of available drones, he will be redirected to the drone control interface illustrated in Fig. 4. Fig. 4 depicts the Dronemap Web graphical user interface.

The web interface contains all information about the drone, including altitude, air/ground speeds, heading, battery level, location address based on GPS coordinates, and GPS fix status. These real-time data are received through the JavaScript Websockets client interface that connects to the Websockets interface of the DP. This ensures real-time updates of the Web interface with the status of the drone, and a reliable bi-directional communication between the Web ground station of MAVLink streams and the MAVProxy through the Websockets protocol. A comprehensive JavaScript/Ajax library was developed to parse and process incoming MAVLink messages and update the Web interface in real-time.

In what concerns control commands, the Web ground station allows the user to change the flight mode, arm/disarm the drone, take-off and landing, navigate to a waypoint in a guided mode, load and save a mission, execute a mission in autonomous mode, and return to launch. A mission refers to visiting a set of waypoints. The Web ground station allows to add and remove waypoints to a mission and save it to the drone, through DP. All these control actions are performed through the SOAP Web Services interface using remote method invocation. For example, to take-off, a Web service client invokes the take-off method of the DP Cloud Manager Web Services called `MAVLinkControllerService`. Fig. 5 presents an excerpt from the WSDL (Web Service Description Language) document. The use of Web Services provides much flexibility considering that it is platform-independent. In fact, it is possible to access DP cloud resources through Web Services using any programming language such as Java, C++, Python, etc.

Technical note. For security purposes, current browsers do not allow to use JavaScript for directly invoking remote SOAP Web Services methods. This is a known issue that prevents from directly invoking Web Services methods through client-side scripting. We overcome this problem by developing using PHP a proxy SOAP client interface that invokes the SOAP Web Services of the DP Cloud Manager. As such, the parameters of the remote methods are sent from the browser in JavaScript using a typical Ajax GET request to the PHP SOAP client page, which prepares the corresponding SOAP message using the received parameters, invokes the Web service and returns the result to the browser through Ajax response.

The Web ground station allows to define missions for the drones in real-time, change the mission dynamically by adding and/or removing waypoints as required, navigate to a particular waypoint in `Guided` mode.

It also to be noted that Dronemap Planner is able to track the communication quality between the drone and the cloud and visualizes it in the web interface. Two types of link quality estimators are used, namely, short-term and long-term link quality estimators, using an exponentially weighted moving averages over different time windows. The link quality information is important to make sure about the reliability of the drone-cloud connection before starting any mission and allows the cloud to select drones with good communication qualities, and also to take preventive actions if a drone communication quality drops over time and more generally ensures contingency management.

6.2. DroneTrack: real-time tracking using drones

DP cloud offers great potential for developing new Internet-of-Drones applications thanks to its open software interfaces and service-oriented architecture design.

DP was used to develop, DroneTrack, a follower application which performs real-time tracking of moving objects through the cloud using their GPS coordinates. The application consists in a moving target object (e.g. person, vehicle, etc.) that is being tracked in real-time by a drone. The moving object periodically sends GPS coordinates using a mobile application to the DP server, which forwards it to the drone.

The DroneTrack follower application was developed as an extended software component into DP cloud architecture. The `Follower` component defines software Web service interfaces to expose its functionalities as SOAP Web Services to the client application from which it receives commands and GPS coordinates of the target object. In addition, it interacts with the `Drone` and `MAVProxy` components of DP to send GPS navigation commands to the drone. The Follower Web Services methods are exposed in the `ApplicationWS` component, which as a SOAP Web service module that exposes all applications services, including the Follower application.

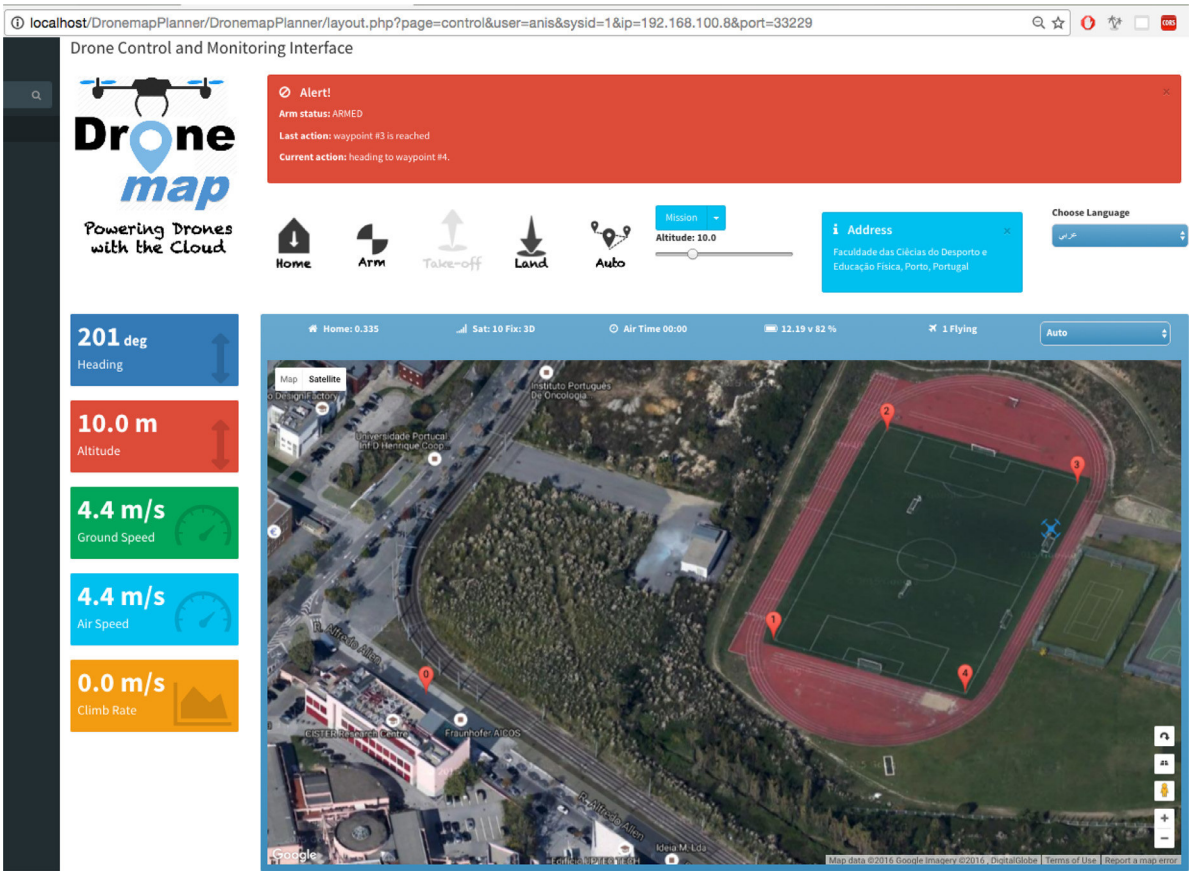

Fig. 4. Dronemap web-based ground station.

```
▼<binding xmlns:ns1="http://cloud.dronemap.org/wsdl"
 name="MAVLinkControllerWSImplPortBinding"
 type="ns1:MAVLink Action Web Service">
   <soap:binding
   transport="http://schemas.xmlsoap.org/soap/http"
   style="document"/>
 ▼<operation name="takeoff">
     <soap:operation soapAction=""/>
   ▼<input>
       <soap:body use="literal"/>
     </input>
   ▼<output>
       <soap:body use="literal"/>
     </output>
   </operation>
```

Fig. 5. Dronemap web-based ground station.

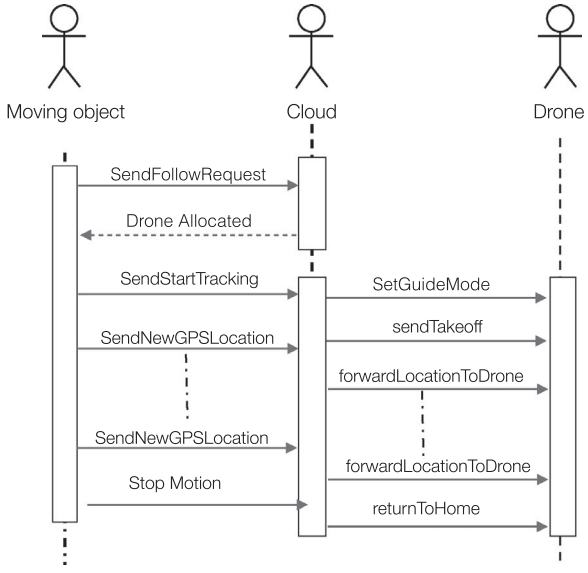

Fig. 6. Follower application sequence diagram.

Fig. 6 presents the DroneTrack Follower application sequence diagram, and Fig. 7 shows the follower client mobile application.

More specifically, the follower application has five Web Services methods available for the end-user client applications, which are as follow:

- *Follow Request web service method:* this method allows a user to send a follow request to the cloud. The request has as parameter the current location of the user to track. Once the request is received, the follower cloud application will search for an available drone among all drones registered in the cloud and select the one that will minimize the cost of the mission and with sufficient energy. In our current implementation, we consider the cost as the distance to the person to follow; i.e. the closest drone to the person will be selected and allocated for the tracking mission. It is possible to consider other metrics for the cost to select a drone namely the remaining energy, the type of drone, availability, ...
- *Cancel Follow Request web service method:* The user can cancel the request at anytime before the tracking is started, which release the allocated drone and make it available.
- *Start Tracking web service method:* When the user starts the tracking, the drone switches to the `GUIDED` flight mode, arms its motors (if not), and flies to the specified altitude, then heads

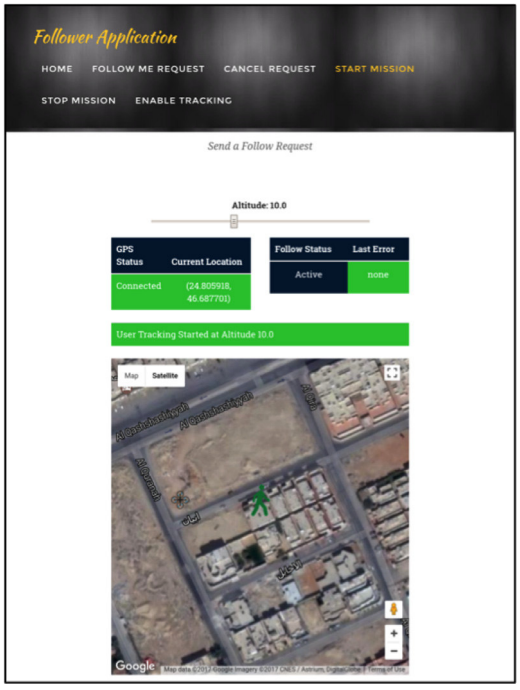

Fig. 7. Follower client application (android device).

towards the location of the moving object. As long as the user moves, new GPS coordinates are sent to the drone in real-time to track it, if the tracking is enabled. Technically, the mobile follower client application opens a Websockets connection with DP cloud to exchange the GPS coordinates in real-time. On the follower client application side, the Websockets inbound stream receives the GPS location of the drone in real-time, and updates it in the Web interface using Google Maps so that the user can track the location of the drone.

- *Stop Tracking web service method:* When the user cancels the tracking, the drone will stop the tracking mission and then returns to its home location and becomes available for other missions.
- *Enable/Disable Tracking web service method:* this command allows the user to enable or temporarily disable the tracking without completing the mission. When the mission is disabled, the drone will still be flying and allocated to the user, but it will not track its new GPS locations until the tracking is enabled again.

6.3. Drone programming API

The development of drones' applications is typically restricted to embedded systems' engineers who should have sufficient knowledge of drones hardware, drivers, and their low level APIs. Some development frameworks like Robot Operating System (ROS) and DroneKit offer high-level libraries for developing drone's control applications. These frameworks, while effective, they require to embed these programs into the drone onboard computer. As such any change in the program will require uploading again the program to take effect, which may become tedious if changes must occur too often. There are no means of online programming of the drone with existing frameworks. (Listing 1)

Dronemap Planner addresses this gap and provides software developers with new and effective approaches for developing programs for drones at much higher-level than existing frameworks. First, with DP, software developers do not need to have full knowledge of the drone system, as the cloud will make a full abstraction to it. Second, they can change the programs to control the drones without having to upload it onboard. The cloud will be the interface between the software developer programs and the drone, through the use of Web services.

The advantage of the cloud-based drones' missions control is to allow developers to develop applications and interact with drones leveraging the use of cloud Web Services. In fact, most programming languages fully support SOAP and REST Web Services and provide appropriate APIs to interact with them.

In this section, we demonstrate how DP makes it simple to develop applications of drones through cloud Web Services APIs. In what follows, we provide an example written in Java, which can also be written in any other language

This definitely helps in providing pragmatic educational tools for teaching and learning drones programming for students at undergraduate or even high-school levels, as no prior background on drones, robots or MAVLink is needed to develop programs. The program below defines an example of mission for a drone written in Java using the drone client API developed to interact with the drones through the Dronemap Planner cloud. The `DroneClient` UML class diagram is presented in Fig. 8. A similar API could be developed for other programming languages like Python.

In Line 3, a new `DroneClient` object is created specifying the IP address of the Dronemap Planner server. In Line 4, the drone client object attempts to connect to a drone with a system ID equal to 1. The connection is successful if the client is able to connect to all Web Services and Websockets servers, and a drone with the specified system ID exists. If the drone exists, the mission will be executed. In Line 6, the user can query DP about the remaining battery level of the drone. Lines from 7 to 15 define a new mission that includes changing the flight mode to guided, arming the drone, taking-off for an altitude of 20.3 m, going to a pre-defined location, and finally returning to launch. The program above is rather illustrative and more sophisticated event-driven programs could be written based on the API.

6.4. Controlling a drone using ROSLink

In this section, we demonstrate how DP is able to control and monitor ROS-enabled robots and drones.

In [29], we used DP to control the Turtlesim, the simulated turtle robot in ROS. We considered an open-loop control application of the motion of Turtlesim robot to follow a spiral trajectory. We analyzed the impact of network and cloud processing delays on the generated spiral trajectories with varying the linear velocity steps. It was observed that network delays resulted in slight discrepancy of trajectories obtained from sending commands over the cloud. However, the jitters remain in the order of tens of microseconds, which is acceptable and does not compromise the real-time control of the robot through the cloud.

In this section, we will illustrate the integration of Gapter EDU drone [30], using ROSLink with Dronemap Planner, and its control over the Internet. The concept can be applied to any drone or robot that operates with ROS.

Gapter EDU is a quadcopter designed for education and research. It has a Pixhawk autopilot hardware using the Ardupilot software, and re-enforced with an embedded single board computer Odroid XU4, with 2GB of RAM, and Samsung Exynos5422 Cortex-A15 2Ghz and Cortex-A7 Octa core CPUs. ROS Indigo is installed on Odroid XU4 and can interact with Pixhawk through the `mavros` ROS package. `Mavros` is a ROS package that represents an interface layer between ROS and Ardupilot systems using the MAVLink communication protocol. The `mavros` package provides required ROS topics and nodes that makes it possible to control Ardupilot systems using ROS commands.

```
1  public static void main(String []args) throws ↩
       InterruptedException{
2  Gson gson = new Gson();
3  DroneClient drone = new DroneClient("192.168.1.102");
4  if (drone.connect(1)){
5      System.out.println(drone.getDroneHostID(droneID));
6      System.out.println(drone.getRemainingBatteryLevel(↩
           droneID));
7      System.out.println(drone.getDroneHostID(1));
8      drone.flightMode(Config.MAVLINK_SET_MODE_GUIDED);
9      Thread.sleep(5000);
10     drone.arm();
11     Thread.sleep(5000);
12     drone.takeoff(20.3);
13     Thread.sleep(10000);
14     Location3D location = new Location3D(24.734840, ↩
           46.698421, 25.0);
15     drone.goToGoal(gson.toJson(location));
16     Thread.sleep(40000);
17     drone.flightMode(Config.MAVLINK_SET_MODE_RTL);
18 }
```

Listing 1. Sample drone client mission control Java program.

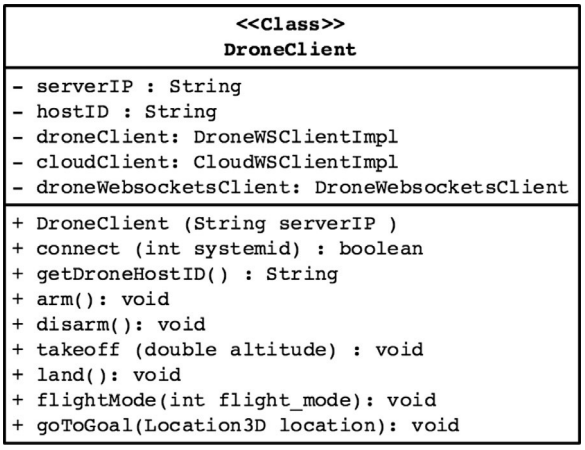

Fig. 8. DroneClient UML class diagram.

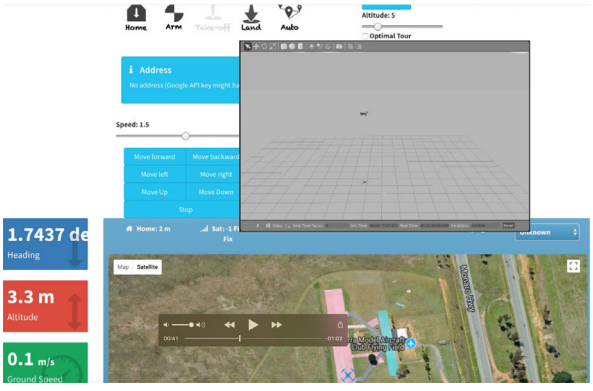

Fig. 9. Control of Gapter using ROSLink with SITL and Gazebo Simulators.

We developed a simulation model for Gapter EDU drone using Software-In-The-Loop (SITL) and Gazebo simulator. SITL is a simulation platform of Ardupilot system, which allows to simulate Ardupilot drones. Gazebo is a 3D robotics simulation platform that supports ROS. The interaction between ROS and SITL is also based on the `mavros` package.

In order to enable a user to control Gapter EDU through Dronemap Planner, we developed the `ROSLinkBridge` node for the Gapter EDU drone, which maps `mavros` commands and topics to ROSLink messages. These messages are exchanged between the `ROSLinkBridge` and Dronemap Planner cloud. The `ROSLinkProxy` software component in Dronemap Planner is responsible for receiving ROSLink messages, parsing them and forwarding them to the user's Web application. The latter parses incoming ROSLink messages and updates the monitoring Web interface accordingly. In addition, the user can send control commands such as takeoff, land or motion commands through the Web interface using ROSLink messages. These messages are received through a Web Services in Dronemap Planner, which will forward them to the drone to execute the commands.

Fig. 9 shows the Web interface for the control and monitoring of Gapter EDU drone using ROSLink and Dronemap Planner. The gray rectangle represents the Gazebo-SITL simulator of Gapter. A video demonstration is available in [31]. The user can send control commands such as takeoff, land, and also control the motion of the drone by sending velocity command through the directions buttons. In addition, the interface allows the user to monitor the status of the drone in real-time, similarly as with MAVLink drones.

Fig. 10 shows the sequence of messages exchanged between ROSLink and Gapter. When Gapter EDU drone is started, its `ROSLink bridge` sends its messages through UDP sockets to the Dronemap Planner. Then, DP will forward them through the Websockets interface to JavaScript Websockets client interface of the Web client application. As with MAVLink, we developed a comprehensive JavaScript/Ajax library to parse all incoming ROSLink messages and update the Web interface accordingly. The command messages are issued from the Web interface, and eventually will be sent through Websockets to the `ROSLink bridge`, which is responsible for parsing them and publish to the appropriate topic. For example, when a user clicks on the Take-off button of the Web interface, a ROSLink Take-off command message is issued and sent to DP, which will forward to `ROSLink bridge` of Gapter EDU. When this ROSLink Take-off command message is received by `ROSLink bridge`, it will be parsed and converted to a ROS topic command. In this case, it will publish a message of type `CommandTOL` on the ROS topic `takeoff`.

7. System deployment and validation

In this section, we present an experimental study for the performance evaluation of the Dronemap Planner cloud. We first evaluate the real-time performance of the DroneTrack tracking application in terms of tracking accuracy; Then, we investigate the impact

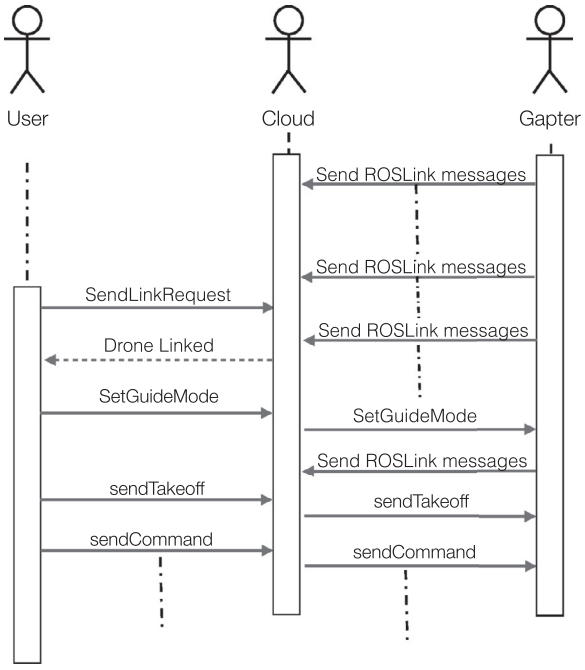

Fig. 10. ROSLink and Gapter.

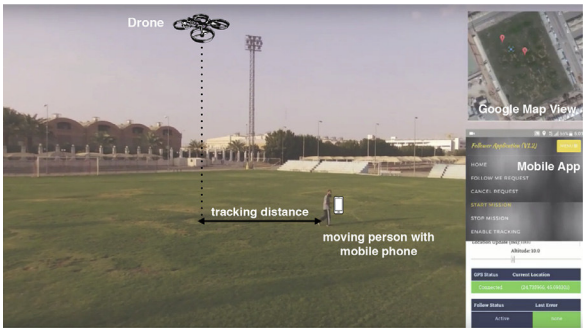

Fig. 11. Experimental environment.

of network and cloud delays on the performance of open-loop control applications of ROS-enabled robots.

7.1. Experimental set-up

We conducted experiments with a real drone to demonstrate a proof-of-concept of the follower application and evaluate its performance. The experimental scenario of the tracking application consists in tracking a moving person with a real drone at the Football field in Prince Sultan University as shown in Fig. 11.

We used our custom built drone, a 450 mm quadcopter with DJI F450 Frame equipped with a Navio2 autopilot on top of a Raspberry PI 3 single board computer. Raspberry PI 3 has Quad Core 1.2GHz Broadcom 64-bit CPU, with 1GB of RAM and MicroSD card for storage. It has an embedded WiFi and Bluetooth interfaces. Navio2 autopilot board is a drone controller hardware that is equipped with the UBlox NEO-M8N embedded GNSS receiver to track GPS signals with an external antenna, while allowing the connection of external GPS devices in its UART port. It has a dual IMU with two 9 degree-of-freedom IMUs, namely the MPU9250 and LSM9DS1. Each IMU contains an accelerometer, a gyroscope, and a magnetometer which are fused together to estimate drone acceleration and speed. The video demonstration of this proof-of-concept is available at [32].

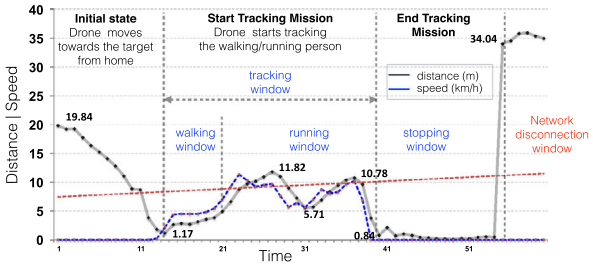

Fig. 12. Tracking distance vs. time.

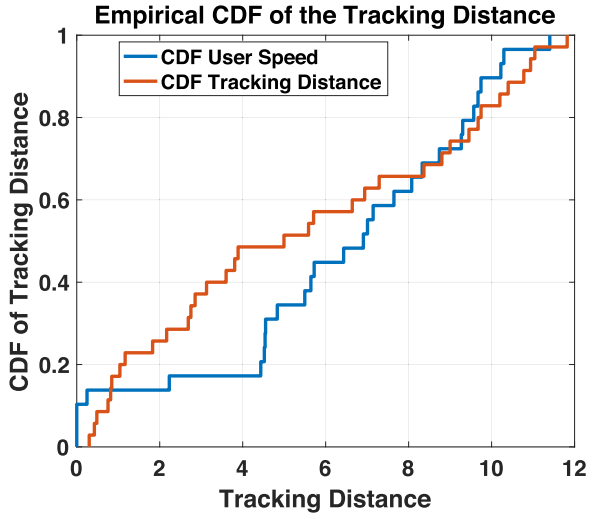

Fig. 13. CDF of the tracking distance.

The drone connects to Dronemap Planner through a 3G WiFi router to which its streams its status and receives command through MAVLink messages. The user is connected to the cloud using a mobile device with 3G connection, sends tracking requests, and updates its GPS locations at 1Hz frequency.

It has to be noted that commercially-available drones typically create their own ad-hoc network. In this case, the client application needs a point-to-point connection with the drone, and this prevents it to connect to the Internet. In our case, we have changed the network configuration of the Navio2 autopilot (similar to other autopilots like Erle Brain) so that it connects to an infrastructure network (i.e. a 3G/4G WiFi router), to guarantee Internet connectivity. This is one of the reasons why we used Navio2 as autopilot since it enables connection with the Internet. For a comprehensive tutorial on how to set the network configuration of the autopilot, the reader may refer to our tutorial in [33].

7.2. Results

Fig. 12 shows the results of the experiments. Initially, the drone was located at home location 20 m away from the moving person. Then, when it received the tracking request, it moved towards the moving person until flying on top of him at the same location. When the moving person was walking at slow speed, the tracking distance was below 4 m in average, and during the running time window, it increased to 11.82 m. It is observed that speed (blue curve) and tracking distance (black curve) are correlated. The loss of connection in the stopping window was due to the mobile phone going into a sleep mode. This resulted in interrupting the tracking mission and tracking distance reached 34 m.

Fig. 12 shows the tracking window of Scenario 1 involving a walking person and a real drone. Fig. 13 shows the cumulative dis-

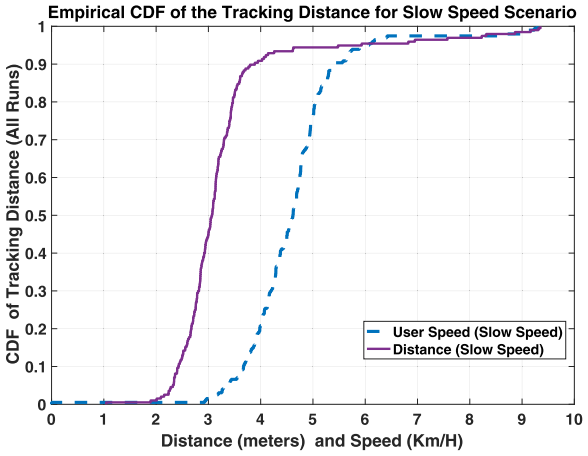

Fig. 14. CDF of the tracking distance at slow constant speed.

tribution function of the tracking distance during the tracking window shown in Fig. 12.

The correlation between the speed and the distance can be clearly observed when comparing the blue curve of the speed and the black curve of the tracking distance in Fig. 12.

At first, the person starts walking in the football field, he abruptly starts running for a short period and then resumes walking. The running window in Fig. 12 shows an increase of the distance up to 11.82 m, then abruptly, the distance decreases to 5.71 m as soon as the moving person starts walking back, finally the distance goes back to 10.78 m when running. The gap between the two curves represents the network delay as the drone will respond to new location and move towards it as soon as a new GPS location is received by the drone. The coefficient of variation of the tracking distance is as high as 0.86 which is due to the variation of the speed in walking and running during the experiments.

We conducted other experiments with a person moving with a slow constant speed (below 5 km/h) and results are illustrated in Fig. 14, which presents the CDF of the tracking distance. It can be clearly observed that the tracking distance is strictly lower than 4 m in 90%, with an average tracking distance of 3 m.

In summary, the results demonstrate that Dronemap Planner is a cloud robotics solution that is able to provide an acceptable quality of service for soft real-time applications. The tracking accuracy can further be improved by increasing the frequency of updating GPS coordinates of the moving person and controlling more the end-to-end delays.

We note that it is possible to improve the accuracy of tracking application and compensate more the delays by adding additional sensors like cameras and using filtering techniques (like extended Kalman filters) for sensor fusion to reduce the measurement noise and use predictions in addition to measurements.

7.3. Discussions on real-time challenges of the Internet-of-Drones

Controlling any real-time system over the network will impose that the network provides a reliable quality of service, otherwise, the real-time system will not operate correctly. This is a common fact for all networked control systems, including the Internet-of-Drones.

When it comes to controlling drones over the Internet, the same challenges remain open for such a control, but we differentiate between two situations: (1) hard real-time control, (2) soft real-time control. Hard real-time control will require zero fault and zero deadline misses, otherwise the system may crash. These constraints impose the usage of a very reliable and high quality-of-service network. For example, controlling a drone using a joystick over the Internet may be very risky as missing one command, or having a command with an excessive delay may result in crashing the drone against a wall. This can be overcome by implementing a smart onboard autonomous system on the drone to take appropriate actions to avoid crashes when a command is not received (e.g. collision avoidance systems).

On the other hand, the control of drones over the Internet may impose soft real-time constraints or even no real-time constraints. For example, sending a list of waypoints to the drone, and then flying it in autonomous navigation mode with passive monitoring, does not impose any real-time constraints, as the Internet will be used simply to deliver offline commands to the drone. Another example, would be the tracking application that we presented in the paper which involves soft real-time constraints, as missing some messages will just result in a temporary increase of the tracking error, but will not cause any harm or crash to the drone system. In summary, the impact of the latencies and delays on the performance of the Internet of drones depend on different parameters including the strictness of the real-time constraints, the type of application, the quality of service provided by the network, and the cloud processing delays. With the exponential evolution of networking infrastructure, the use of cloud robotics for hard real-time control applications will become possible in the future. This area is an open research topic which requires more investigations.

8. Conclusions

In this paper, we proposed Dronemap Planner, a cloud-based management system for robots and drones over the Internet-of-Things. Dronemap Planner is a comprehensive framework that over the horizon for multiple IoT applications using drones. We presented several applications, namely real-time tracking, and controlling and monitoring drones and robots over the Internet. We also discussed the concept of Internet-of-Drones and its security aspects. The performance of Dronemap Planner was also evaluated in terms of real-time guarantee. It was demonstrated that Dronemap Planner exhibits good performance for real-time applications.

One of the potential usages of Dronemap Planner is in the context BVLOS (Beyond Visual Line of Sight) operations of drones which represents the next major challenge for commercial applications using drones, such as delivery, surveillance. In fact, Dronemap Planner allows to remotely control and monitor the drones anywhere and anytime without having to be in the visual light of sight with the drone. However, this induces security threats and safety risks that must be taken into account. One of the initiatives in this respect is that NASA has established UAS Traffic Management (UTM) system to ensure safe operation of drones. Dronemap Planner helps in implementing these UTM regulations to ensure the safe operation of drones. In fact, Dronemap Planner maintains real-time status data about drones during their missions and can apply law enforcement and regulations on drones while flying. For example, it is possible to control the airspace where the drone is allowed to fly and apply geofencing constraints on the drone mission in real-time. It is even possible to adopt adaptive geofencing based on the feedback received from drones and current circumstance during flight operations.

Furthermore, the integration of blockchain technology with drones for the authentication of drones is a promising research direction as authentication and privacy of drones messages is still an open problem, in particular since Dronemap Planner operates on cloud premises.

We are currently working towards extending Dronemap Planner in several aspects. First, we aim at using DP for multi-drone task allocation by coordinating the tasks of multiple drones during critical missions. We are also investigating how to secure the

access to drones and to the Dronemap Planner cloud to mitigate security threats that might compromise the safety of operations of drones during their mission. In fact, the current specification of the MAVLink and ROSLink protocols are totally insecure and this imposes a serious problem towards the deployment of drones in real applications.

Acknowledgments

This work is supported by the Research and Initiative Center at Prince Sultan University. It is also supported by Gaitech Robotics. This work was initially supported by the Dronemap project entitled "DroneMap: A Cloud Robotics System for Unmanned Aerial Vehicles in Surveillance Applications" under the grant number 35-157 from King Abdul Aziz City for Science and Technology (KACST).

Appendix

A.1. The MAVLink protocol

Micro Air Vehicle Link (MAVLink) is a communication protocol for small unmanned vehicles. MAVLink was released by Lorenz Meier in 2009 under LGPL license. It specifies a set of messages that are exchanged between a small unmanned vehicle and a ground station. One of the main reasons for the wide use of MAVLink by many ground stations and autopilots is that it has a stable message format. One of the most advanced and reliable autopilots that support the MAVLink is Ardupilot. This autopilot is capable of controlling any vehicle system, from airplanes, multirotors, and helicopters, to boats and even submarines. Ardupilot supports several flight modes for drones, including:

- **STABILIZE (MANUAL):** allows to control the vehicle manually using an RC controller.
- **ALT HOLD:** known as `altitude hold` mode, where the drone maintains a fixed altitude, while the user controls the roll, pitch and yaw.
- **LOITER:** attempts to maintain the current location, heading and altitude. Loiter means "stand or wait around idly or without apparent purpose". This mode requires either a GPS (for outdoor navigation) or optical flow sensor (for indoor navigation) to get and maintain the position.
- **LAND:** allows the drone to land to ground.
- **RTL:** also known as `Return-To-Launch`. It navigates the drone from its current location to hover above the home location, and then to land.
- **GUIDED** operates **only with a GPS** and allows to send the drone to a GPS coordinate dynamically through the ground station.
- **AUTO:** in `Auto` mode the drone follows a pre-programmed flight path stored in the autopilot, which consists of a set of waypoints to visit.

All MAVLink messages contain a header appended to every data payload of the message. The header contains information about the message while the payload contains the data carried out by the message. The total MAVLink message has a length of 17 bytes, which includes 6 bytes for the header, 9 bytes for the payload and 2 bytes for the checksum. The checksum is intended to verify the integrity of the message and that it was not altered during its transmission. The MAVLink message header format is as follow:

- **Start-of-frame:** The header contains a packet start sign encoded into 1 byte, which indicates the start of the packet.
- **Payload-length:** The byte with index 1 corresponds to the payload length, which is encoded on one byte for a value between 0 and 255.

```
{"type": uint8_t,
 "autopilot": uint8_t,
 "base_mode": uint8_t,
 "custom_mode": uint32_t,
 "system_status": uint8_t,
 "mavlink_version": uint8_t,}
```

Listing 2. MAVLink heartbeat message structure.

```
{"onboard_control_sensors_present": uint32_t,
 "onboard_control_sensors_enabled": uint32_t,
 "onboard_control_sensors_health": uint32_t,
 "load": uint16_t,
 "voltage_battery": uint16_t,
 "current_battery": int16_t,
 "battery_remaining": int8_t,
 "drop_rate_comm": uint16_t,
 "errors_comm": uint16_t,
 "errors_count1": uint16_t,
 "errors_count2": uint16_t,
 "errors_count3": uint16_t,
 "errors_count4": uint16_t, }
```

Listing 3. MAVLink system status message.

- **Packet sequence:** one Byte with index 2 refers to the packet sequence, which indicates the incremental sequence number of the packet by a value between 0 and 255. If the sequence number reaches 255, it is reset back to zero.
- **System ID:** The system ID is an important parameter, which identifies the system that can be a drone or any vehicle running the autopilot. The fact it is encoded in 8 bits restricts the total number of vehicles on one ground station to 256, including the ground station. The MAVLink protocol imposes that the ground station has a system ID equal to 255. This is how it is recognized in the autopilot running in the monitored vehicle.
- **Component ID:** The component ID is the identifier of the component sending the message inside the system. For the moment, there is no subsystem or component, so it will be of no use.
- **Message ID:** The message ID is an important field. It indicates the type of message being carried out in the payload. For example, if the message ID is 0, this means that the message is of type `HEARTBEAT`, which is periodically, every 1 second, sent to indicate that the system is active. Another example, if the message ID is 33, this indicates that the payload carries out GPS coordinate of the system.
- **Payload:** The data into the message, depends on the message ID.
- **CRC:** Check-sum of the packet. To ensure message integrity and to ensure the sender and receiver both agree in the message that is being transferred.

The payload from the packets are MAVLink messages. Every message is identifiable by the ID field on the packet, and the payload contains the data from the message. Several `control` and `state` messages are defined in MAVLink protocol. In what follows, we present a sample of the main MAVLink messages. For a complete set of messages, the reader may refer to [34].

The `HEARTBEAT` message is the most important message in MAVLink (refer to Listing 2). It indicates the presence of the vehicle system and that it is active. A vehicle should send the `HEARTBEAT` message periodically (generally every second) to the ground station to tell the latter that it is alive. This is a mandatory message. The `System Status` message (refer to Listing 3) with a message ID equal to 1 carries information about the onboard control sensors present in the drone, and which one are enabled or disabled. It also provides information about the battery status and the remaining voltage. In addition, it provides information about the

```
{"time_boot_ms7": uint32_t,
 "lat": int32_t,
 "lon": int32_t,
 "alt": int32_t,
 "relative_alt": int32_t,
 "vx": int16_t,
 "vy": int16_t,
 "vz": int16_t,
 "hdg": uint16_t,}
```

Listing 4. MAVLink global position message.

communication errors. The `Global Position` message (refer to Listing 4) with an id equal to 33 represents the filtered GPS location given by the GPS. The important information carried in this message is the latitude, longitude, and the altitude, all of them encoded into 4 bytes (32 bits). The values of latitude and longitude are multiplied by 10^7, so to get the real value we divide by 10^7. The altitude is expressed in millimeters. It also provides information about the speed of the drone around the 3 axes in addition to orientation referred to as heading.

A.2. The ROSLink protocol

ROSLink [29] is a lightweight communication protocol that allows the control of ROS-enabled robots through the Internet. The integration of robots with the Internet is essential to advance new sorts of cloud robotics applications where robots are virtualized and controlled through the Internet. For example, in our previous work [35], we developed a personal assistant robot for providing home and office services using ROS. However, the communication between the robot and the end-user was constrained to the scope of a local area network (LAN), which presented a limitation. We designed ROSLink to allow Internet connectivity of ROS-enabled robots.

ROSLink makes a complete abstraction of ROS by providing all information about the robot through ROS topics/services without exposing ROS ecosystem to the users. It defines a set of interfaces for the users to interact with the robot, and a set of messages to exchange between them. ROSLink provides asynchronous communication as 3-tier architecture between the robots and the end-users through the cloud. It is composed of (1) `ROSLink Bridge` client in the robot side, (2) `ROSLink Proxy` acts as a server in the ground station, and (3) client application at the user side which interacts with the robot through the ROSLink protocol. ROSLink Bridge is a ROS node that accesses all topics and services of interest in ROS, and sends selected information in ROSLink messages, serialized in JSON format.

`ROSLink` messages contain information about the command and its parameters. To standardize the type of messages exchanged, we specified a set of ROSLink messages that are supported by the ROSLink Proxy. These messages can be easily extended based on the requirements of the user and the application. There are two main categories of ROSLink messages: (*i.*) *State messages*: these are message sent by the robot and carry out information about the internal state of the robot, including its position, orientation, battery level, etc. (*ii.*) *Command messages*: these are messages sent by the client application to the robot and carry out commands to make the robot execute some actions, like for example moving, executing a mission, going to a goal location, etc.

The most basic `ROSLink` message is the **`Heartbeat`** message, which is sent periodically from the robot to the ROSLink proxy server, and vice-versa. Every `ROSLink Bridge` should implement the periodic transmission of the `Heartbeat` message. The objective of the `Heartbeat` message is for the proxy server to ensure that the robot is active and connected, upon reception of

```
{"roslink_version": int8,
"ros_version": int8,
"system_id": int16,
"message_id": 0,
"sequence_number": int64,
"payload": {"type": int8
"name": String,
"system_status": int8,
"owner_id": String,
"mode": int8}
}
```

Listing 5. ROSLink heartbeat message structure.

```
{"roslink_version": int8,
"ros_version": int8,
"system_id": int16,
"message_id": 1,
"sequence_number": int64,
"payload":{"onboard_control_sensors_present":uint3↩
    2,
"onboard_control_sensors_enabled": uint32,
"voltage_battery": uint16,
"current_battery": int16,
"battery_remaining": int8, }
}
```

Listing 6. ROSLink robot status message structure.

```
{"roslink_version": int8,
"ros_version": int8,
"system_id": int16,
"message_id": int8,
"sequence_number": int64,
"payload":{"time_boot_ms":uint32
"x": float64,
"y": float64,
"z": float64,
"vx": float64,
"vy": float64,
"vz": float64,
"wx": float64,
"wy": float64,
"wz": float64,
"pitch": float64,
"roll": float64,
"yaw": float64,}
}
```

Listing 7. ROSLink global motion message structure.

that message. In the same way, a robot that receives a `Heartbeat` message from the `ROSLink Proxy` server ensures that the server is alive. This message increases the reliability of the system when it uses a UDP connectionless protocol, such that both ends make sure of the activity of the other end. Failsafe operations can be designed when the robot loses reception of `Heartbeat` messages from the user such as stopping motion or returning to start location until connectivity is resumed.

The `Heartbeat` message structure is defined in JSON representation in Listing 5. In the ROSlink protocol, the `message_id` of the `Heatbeat` message is set to zero. The **`Robot Status`** message contains the general system state, like which onboard controllers and sensors are present and enabled in addition to information related to the battery state. Listing 6 presents the`Robot Status` message structure, which has a `message_id` equal to 1. The **`Global motion`** message represents the position of the robot and its linear and angular velocities. This information is sent to the ROSLink client at high frequency to keep track of robot motion state in real-time. An instance of the `Global motion` message structure is expressed in Listing 7.

```
{"roslink_version": int8,
"ros_version": int8,
"system_id": int16,
"message_id": int8,
"sequence_number": int64,
"payload":{"time_usec":int64
"angle_min": float32,
"angle_max": float32,
"angle_increment": float32,
"time_increment": float32,
"scan_time": float32,
"range_min": float32,
"range_max": float32,
"ranges": float32[],
"intensities": float32[],}
}
```

Listing 8. Range finder data message structure.

Listing 8 presents the **Range Finder Datamessage**, which carries out information and data about laser scanners attached to the robot. The Range Finder Data sensor information enables to develop control application on the client through the cloud, such as obstacle avoidance reactive navigation, SLAM, etc.

References

[1] M. Gharibi, R. Boutaba, S.L. Waslander, Internet of drones, IEEE Access 4 (2016) 1148–1162, doi:10.1109/ACCESS.2016.2537208.

[2] B. Kehoe, S. Patil, P. Abbeel, K. Goldberg, A survey of research on cloud robotics and automation, IEEE Trans. Autom. Sci. Eng. 12 (2) (2015) 398–409, doi:10.1109/TASE.2014.2376492.

[3] Y. Chen, Z. Du, M. Garca-Acosta, Robot as a service in cloud computing, in: Proceedings of the 2010 Fifth IEEE International Symposium on Service Oriented System Engineering, USA, 2010, pp. 151–158, doi:10.1109/SOSE.2010.44.

[4] R. Arumugam, V.R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F.F. Kong, A.S. Kumar, K.D. Meng, G.W. Kit, Davinci: a cloud computing framework for service robots, in: Proc. Int. Conf. Robot. Autom.(ICRA), 2010, pp. 3084–3089, doi:10.1109/ROBOT.2010.5509469.

[5] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Gálvez-Lôpez, K. Häussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schiele, M. Tenorth, O. Zweigle, R.V.D. Molengraft, Roboearth, IEEE Rob. Autom. Mag. 18 (2) (2011) 69–82, doi:10.1109/MRA.2011.941632.

[6] A. Koubâa, A service-oriented architecture for virtualizing robots in robot-as-a-service clouds, in: International Conference on Architecture of Computing Systems ARCS, Springer International Publishing, Lübeck, Germany, 2014, pp. 196–208.

[7] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, C. Bettstetter, Drone networks: communications, coordination, and sensing, Ad Hoc Netw. 68 (2018) 1–15, doi:10.1016/j.adhoc.2017.09.001.

[8] The MAVLINK Protocol, [Online]. Available: http://qgroundcontrol.org/mavlink/start.

[9] E. Guizzo, Robots with their heads in the clouds, IEEE Spectr. 48 (3) (2011) 16–18, doi:10.1109/MSPEC.2011.5719709.

[10] A. Koubâa, B. Qureshi, M.-F. Sriti, Y. Javed, E. Tovar, A service-oriented cloud-based management system for the internet-of-drones, in: 17th IEEE Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC), 2017, pp. 329–335, doi:10.1109/ICARSC.2017.7964096.

[11] S. Mahmoud, N. Mohamed, Collaborative uavs cloud, in: Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS '14), Orlando, Fla, USA, 2014, pp. 365–373, doi:10.1109/ICUAS.2014.6842275.

[12] S. Mahmoud, N. Mohamed, Broker architecture for collaborative uavs cloud computing, in: 2015 International Conference on Collaboration Technologies and Systems (CTS), 2015, pp. 212–219, doi:10.1109/CTS.2015.7210423.

[13] S. Mahmoud, N. Mohamed, J. Al-Jaroodi, Integrating uavs into the cloud using the concept of the web of things, J. Rob. 2015 (2015) 10pages, doi:10.1155/2015/631420.

[14] G. Ermacora, Advances in Human Robot Interaction for Cloud Robotics applications, Politecnico di Torino, 2016 Ph.D. thesis. doi: 10.6092/polito/porto/2643059.

[15] 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, in: IEEE/RSJ International Conference of Intelligent Robots and Systems, Tokyo (Japan), 2013.

[16] A. AzzarÃ, L. Mottola, Virtual resources for the internet of things, in: In Proceedings of the IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 2015, pp. 245–250, doi:10.1109/WF-IoT.2015.7389060.

[17] H. Chae, J. Park, H. Song, Y. Kim, H. Jeong, The iot based automate landing system of a drone for the round-the-clock surveillance solution, in: IEEE International Conference on Advanced Intelligent Mechatronics (AIM), Busan, 2015, pp. 1575–1580, doi:10.1109/AIM.2015.7222767.

[18] V. Kriz, P. Gabrlik, Uranuslink-communication protocol for uav with small overhead and encryption ability, 13th IFAC IEEE Conf. Programm. Devices Embedd. Syst. 48 (4) (2015) 474–479, doi:10.1016/j.ifacol.2015.07.080.

[19] P. Tripicchio, M. Satler, G. Dabisias, E. Ruffaldi, C.A. Avizzano, Towards smart farming and sustainable agriculture with drones, in: Proceedings of the 2015 International Conference on Intelligent Environments IE, Prague, Czech Republic, 2015, pp. 140–143, doi:10.1109/IE.2015.29.

[20] C. Pautasso, O. Zimmermann, F. Leymann, Restful web services vs. "big" web services: Making the right architectural decision, in: Proceedings of the 17th International World Wide Web Conference (WWW 2008), ACM, New York, NY, USA, 2008, pp. 805–814, doi:10.1145/1367497.1367606.

[21] A. Koubâa, O. Cheikhrouhou, H. Bennaceur, M.-F. Sriti, Y. Javed, A. Ammar, Move and improve: a market-based mechanism for the multiple depot multiple travelling salesmen problem, J. Intell. Rob. Syst. 85 (2) (2017) 307–330, doi:10.1007/s10846-016-0400-x.

[22] S. Trigui, O. Cheikhrouhou, A. Koubaa, U. Baroudi, H. Youssef, Fl-mtsp: a fuzzy logic approach to solve the multi-objective multiple traveling salesman problem for multi-robot systems, Soft Comput. 21 (24) (2017) 7351–7362, doi:10.1007/s00500-016-2279-7.

[23] R. Altawy, A.M. Youssef, Security, privacy, and safety aspects of civilian drones: a survey, ACM Trans. Cyber-Phys. Syst. 1 (2) (2016) 7:1–7:25, doi:10.1145/3001836.

[24] D. He, S. Chan, M. Guizani, Drone-assisted public safety networks: the security aspect, IEEE Commun. Mag. 55 (8) (2017) 218–223, doi:10.1109/MCOM.2017.1600799CM.

[25] J. Daemen, V. Rijmen, The design of rijndael: AES - the advanced encryption standard, Information Security and Cryptography, Springer Berlin Heidelberg, 2013.

[26] N.M. Rodday, R. d. O. Schmidt, A. Pras, Exploring security vulnerabilities of unmanned aerial vehicles, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2016, pp. 993–994, doi:10.1109/NOMS.2016.7502939.

[27] Autonomous Mission with GoPro Camera view on drone using Dronemap Planner, 2017, [Online]. Available: https://www.youtube.com/watch?v=QqtnLHKJI2k.

[28] Drones'auto-detection in Dronemap Planner, 2017, [Online]. Available: https://www.youtube.com/watch?v=MjW01sRdDds.

[29] A. Koubaa, M. Alajlan, B. Qureshi, ROSLink: Bridging ROS with the Internet-of-Things for Cloud Robotics, vol. 2, Springer International Publishing, Cham, pp. 265–283. doi:10.1007/978-3-319-54927-9_8

[30] Gapter EDU drone, 2018, [Online]. Available: http://edu.gaitech.hk/gapter/.

[31] Gapter EDU ROSLink control video, 2018, [Online]. Available: https://www.youtube.com/watch?v=eGYOH7akp-Q.

[32] Video demonstration of the follower application the football field of prince sultan university with a real drone, 2017, [Online]. Available: https://www.youtube.com/watch?v=9a8Gn0BDe8U.

[33] Gapter EDU, autopilot network configuration, 2018, [Online]. Available: https://www.youtube.com/watch?v=eGYOH7akp-Q.

[34] MAVLink Common Message set specifications, (https://pixhawk.ethz.ch/mavlink/). Accessed: 2017-06-03.

[35] A. Koubâa, M.F. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze, A. Mahmoud, Turtlebot at office: A service-oriented software architecture for personal assistant robots using ros, in: Proceedings of 2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC), Bragança, Portugal, 2016, pp. 270–276, doi:10.1109/ICARSC.2016.66.

Anis Koubâa received his B.Sc. in Telecommunications Engineering from Higher School of Telecommunications (Tunisia), and M.Sc. degrees in Computer Science from University Henri Poincare (France), in 2000 and 2001, respectively, and the Ph.D. degree in Computer Science from the National Polytechnic Institute of Lorraine (France), in 2004. He was a faculty member at Al-Imam University from 2006 to 2012. Currently, he is a Professor in the Department of Computer Science, Advisor to the Rector, and Leader of the Robotics and Internet of Things Research Lab, in Prince Sultan University. He is also R&D Consultant at Gaitech Robotics in China and Senior Researcher in CISTER/INESC TEC and ISEP-IPP, Porto, Portugal. He becomes a Senior Fellow of the Higher Education Academy (SFHEA) in 2015. He has published over 190 refereed journal and conference papers, and one patent. His research interest covers mobile robots, cloud robotics, robotics software engineering, Internet-of-Things, Robot Operating System (ROS), cloud computing and wireless sensor networks. Dr. Anis received the best research award from Al-Imam University in 2010, and the best paper award of the 19th Euromicro Conference in Real-time Systems (ECRTS) in 2007. He is the head of the ACM Chapter in Prince Sultan University. His H-Index is 31.