Control Engineering Practice 106 (2021) 104657

Contents lists available at ScienceDirect

Control Engineering Practice

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

Cloud-based mission control of USV fleet: Architecture, implementation and experiments☆

Zhao Wang [a], Shaolong Yang [a,b,c], Xianbo Xiang [a,b,c,*], Antonio Vasilijević [d], Nikola Mišković [d], Đula Nađ [d,**]

[a] School of Naval Architecture and Ocean Engineering, Huazhong University of Science and Technology, Wuhan 430074, China
[b] Hubei Key Laboratory of Naval Architecture and Ocean Engineering Hydrodynamics (HUST), Wuhan 430074, China
[c] Collaborative Innovation Center for Advanced Ship and Deep-Sea Exploration (CISSE), Shanghai 200240, China
[d] Laboratory for Underwater Systems and Technologies, Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

ARTICLE INFO

MSC:
00-01
99-00

Keywords:
Unmanned surface vehicles (USVs)
Cloud-based mission control
Decentralized architecture
Robot Operating System (ROS)
Sea trials

ABSTRACT

In this paper, a cloud-based mission control architecture is proposed to achieve flexible remote access and coordinated mission control among a fleet of unmanned surface vehicles (USVs). First, a cloud-based mission control architecture that renders easy, timely and prioritized remote access to the USVs regardless of the remote operator's location is proposed. It is achieved by leveraging remote cloud-based technology and local Operating onboard System. Decentralized property of the architecture accomplishes scalable monitoring, remote control, data acquisition and missions sharing for an USV fleet. Second, the related software interfaces are required for this task: the user interface of the remote client that is used for mission control/planning and data visualization and that is applicable across mobile robotic systems; and the back-end interface for the local USVs that bridges robotic and cloud server and provides seamless integration with the well-established Robot Operating System (ROS). ROS is nowadays, the most widely used framework for robotics developments. Furthermore, the proposed cloud-based mission control architecture is implemented on a fleet of real vehicles, H2Omni-X USVs, and the performance of the remote experimentation is demonstrated during sea trials at the Adriatic coast, Croatia, representing the practical contribution of this paper.

1. Introduction

Internet and cloud-based services and related robot platforms are becoming attractive in recent years (Doriya, Chakraborty, & Nandi, 2012; Karimi, Duffie, & Dashkovskiy, 2010). Compared with integrated industrial computers (Shen & Shi, 2020; Zhang, Zhang, Chemori, & Xiang, 2018), cloud computing achieves the distributed computing to reduce the computing payload caused by complicated missions in advanced robotic systems, which depends on the heavy processing performance, such as intelligent environment modeling (Mohanarajah, Usenko, Singh, D'Andrea, & Waibel, 2015), object recognition (Ali, Hammad, & Tag Eldien, 2018; Krupinski, Desouche, Palomeras, Allibert, & Hua, 2015; Sivcev et al., 2018), real-time multi-sensor data sampling (Peng, Wang, & Han, 2019; Wang, Liu, & Meng, 2015), cloud-based robotic localization (Ansari, Pal, Shukla, Nandi, & Chakraborty, 2012; Lai & Cheng, 2014), robotic navigation (Salmeron-Garca, Inigo-Blasco, Daz-del-Ro, & Cagigas-Muniz, 2015; Sun, Zhu, Tian, & Luo, 2019), and robotic control (Chu, Xiang, Zhu, Luo, & Xie, 2018; Wang, Karimi, Li, & Su, 2019; Wu, Lou, Chen, Hirche, & Kuhnlenz, 2013; Zheng, Sun, & Xie, 2018). In addition to the support for complicated robotic missions, cloud technology also enables new concepts for other robotic services, such as a flexible motion planning and control of industrial robots instead of a closed monolithic architecture (Vick, Vonásek, Pěnička, & Krüger, 2015), cloud-based multi-robot communication system (Hartanto & Eich, 2014) and video analysis for intelligent robot (Guo, Wu, & Li, 2018). It is obvious that cloud technology brings a new paradigm to design and implements the robotic framework and service, and it also enables the breakthrough of some robotic technical bottlenecks (Kaliski, 2008).

With increasing focus on the exploration and exploitation of ocean resources in recent decades, the marine community has witnessed

☆ The research leading to these results was partially funded by the European Union Horizon 2020 research and innovation programme under the project EUMarinerobots (grant agreement No 731103), Contract CC-MICRO "Cloud-based coordinated mission control of fleet of ASVs", China-Croatia Bilateral Project CC-MARS "China-Croatia Collaboration on Marine Robotic System" , National Natural Science Foundation of China under Grant 51579111 and the Croatian Science Foundation IP-2016-06-2082 CroMarX project.

* Corresponding author at: School of Naval Architecture and Ocean Engineering, Huazhong University of Science and Technology, Wuhan 430074, China.
** Corresponding author at: Laboratory for Underwater Systems and Technologies, Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia.
E-mail addresses: xianboxiang@hust.edu.cn (X. Xiang), dula.nad@fer.hr (Đ. Nađ).

https://doi.org/10.1016/j.conengprac.2020.104657
Received 3 April 2020; Received in revised form 28 July 2020; Accepted 8 October 2020
Available online 1 November 2020
0967-0661/

remarkable growth and development in marine robotics. Unmanned surface vehicles (USVs) have become common in marine robotics, executing missions such as source seeking (Song, Gupta, Hare, & Zhou, 2013), environmental monitoring and ocean sampling (Vasilijević, Nad, Mandić, Mišković, & Vukić, 2017) in a coordinated way with multi-vehicles (Wang & Han, 2016; Zhang, Lapierre, & Xiang, 2013). With the limited range of commonly used communication technologies (WiFi and/or UHF), traditional monitoring and control platforms for marine robots are often co-located with the deployed robotic systems. Assuming constant monitoring is desired, this heavily reduces the operating area of the vehicles. From another perspective, if one of the vehicles is lost, the operator will be unable to recover data from the lost vehicle. Similarly, if the control platform is corrupted the fleet operation will be compromised. With improved connectivity and cellular networks providing bandwidths similar to WiFi, the current technological trend, as seen in Lorencik and Sincak (2013) and Saha and Dasgupta (2018), is towards off-loading processing from local platforms onto the cloud.

In this article, a dedicated mechanism to build connections between robotics and remote access in back-end of cloud server is proposed, in which the cloud-robotics architecture leverages cloud technologies to better distribute monitoring and control tasks, thereby removing the bottleneck encountered in traditional approaches. The cloud-computing benefits (high processing capabilities, robust data storage and server distribution/redundancy) allow mission commands and data to be transmitted and processed without limitation of distance or demanding hardware requirements (Hu, Tay, & Wen, 2012; Xiang, Yu, Zhang, Wilson, & Xu, 2020; Zhang et al., 2017; Zheng, Deng, Zhu, & Deng, 2014). Once vehicles are equipped with mobile communication hardware and a CPU running the cloud client interface, the proposed distributed cloud architecture can be provided as a common solution for mobile robots. Similar cloud-robotics efforts have been widely applied to unmanned ground vehicles and unmanned aerial vehicles such as DronTrack (Koubaa & Qureshi, 2018), RoboEarth (Riazuelo et al., 2015) and Rapyuta platform (McGillivary & Zykov, 2016; Mohanarajah, Hunziker, D'Andrea, & Waibel, 2015; Mohanarajah et al., 2015). To the best of the authors' knowledge, the application to marine robotics is still limited to non-existent, especially in the case of considering coordinated mission control of a fleet of unmanned surface vehicles.

Another advantage of our approach is focus on mission control platform that will ensure an interoperability of variety of systems and vehicles. While the user interface of the remote client is applicable across heterogeneous vehicle types, the back-end client interface is seamlessly implemented within the Robot Operating System (ROS) framework (Quigley, Conley, Gerkey, Faust, Foote, Leibs, Wheeler, & Ng, 2009). The framework is already well-established in control system of land robotics (Silva, Moita, Nunes, Garrote, Faria, & Ruivo, 2012) and continuously gaining popularity in the marine sector (Cai, Wang, Wang, Wang, & Tan, 2019; Gallimore, Stokey, & Terrill, 2018; Mancini, Frontoni, & Zingaretti, 2015). ROS is also available for oriented robotics software such as simulator (Mendonça, Santana, Marques, Lourenço, Silva, & Barata, 2013). Therefore, this implementation enables the use of the proposed cloud-based framework with many different types of robots with ROS onboard, among which are the H2OmniX surface vehicles used for field demonstrations in this paper.

As mentioned above, the first contribution of this article is a cloud-based mission control architecture which renders easy, timely and prioritized remote access to the USVs without the limitation of the operator's location, it leverages cloud technologies to better distribute monitoring and control tasks. The second is that the required software interfaces are designed in a plugin modular way, this cloud-based system can be easily ported to any kind of ROS-based robotics. Furthermore, the practical contributions of this paper are the implementation of the proposed cloud-based mission control architecture on a fleet of real vehicles and remote at sea experimentation, demonstrated during sea trials in Croatia.

The remainder of this paper is organized as follows. Section 2 introduces the applied cloud-based mission control architecture for USV fleet, and centralized and decentralized architectures are listed and compared with each other. System components involved in the cloud-based architecture are described in detail in Section 3. Experiment results on simulation, pool test and sea trail are presented in Section 4. Finally, the paper is concluded with the recap of main concepts and planned future work for extending the proposed cloud-based control architecture.

2. System architecture

The proposed system architecture for marine vehicles is designed within the classic client/server framework. The minimum system consists of a cloud server and at least two clients in remote and local fields respectively as shown in Fig. 1. The remote client connects the remote user and the cloud-server, while the local client connects the vehicle in-field and the cloud-server. It can be easily scaled to multiple local in-field nodes when a fleet of autonomous vehicles is operating. However, there are two possible architectures to integrate the fleet of vehicles with the cloud and consequently with the remote client, and they are introduced in the following section.

2.1. Centralized architecture

In this architecture, a bus-based approach is taken to bridge USVs and local client as shown in Fig. 2(a). There is only a single local client module (L module) in this architecture, hence, a dedicated workstation is required for the module. The module connects to the vehicle network over WiFi by using the local vehicle communication protocols. During communication between USVs and remote client each vehicle sends data to the interface client. The interface client combines the received messages into a joint data packet and sends it to the cloud. The cloud processes the data packets into individual vehicle parameters (e.g. position, orientation, etc.) and stores them in the cloud database. The cloud also handles routing of the data packets and forwards them to the remote control client as needed. Similarly, the packets are unpacked on the remote client for purposes of display and local logging.

Communication is possible from remote clients to USVs in form of mission commands. Depending on the mission and command type, command parameters are serialized and sent to the cloud server which in turn relays them to the local client module. The local client module unpacks incoming packages for individual vehicle and uses the native vehicle communication protocols to execute requested commands.

With this centralized interface, data is transmitted between the vehicles and the remote client in a synchronized style. Easy data synchronization from multiple sources can be beneficial for some scenarios, e.g. a scenario where time consistency during parameter collection is need. However, this layout enforces co-location of the interface client with all USVs since they are required to be on the same local network. Therefore, only distance between users/researchers is expanded, but the USV swarms is still limited to near shore applications.

2.2. Decentralized architecture

The alternative to the previous architecture layout is a decentralized type of the interface shown in Fig. 2(b). The local interface module is moved from a dedicated workstation on the USVs. Each USV establishes a connection with the cloud and performs individualized communication. The overall data exchange is similar to the previously described interface layout. However, there is no extra synchronization of the data when communication towards the cloud. Similarly, commands are directly encoded and executed on each vehicle.

This decentralized architecture does not provide centralized synchronization. Communication load is distributed between vehicles and is less sensitive to scaling. Decentralization also improves overall flexibility concerning the used network layer without affecting the vehicle interface design. This allows for a more heterogeneous approach, as

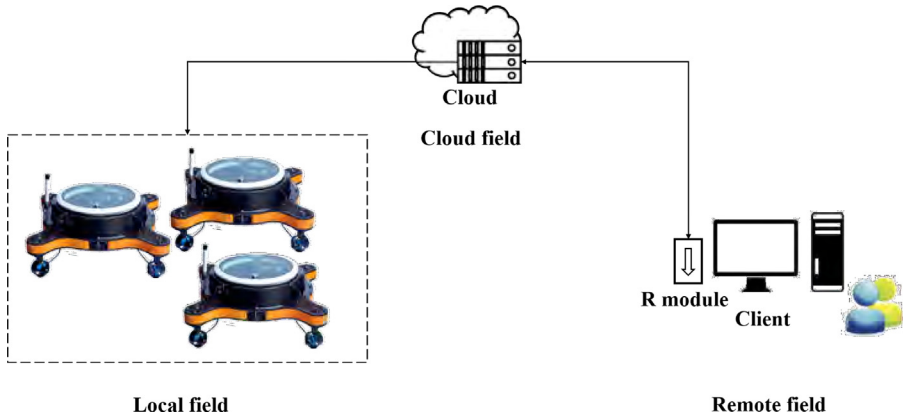


Fig. 1. Abstract architecture of the minimize remote control system.

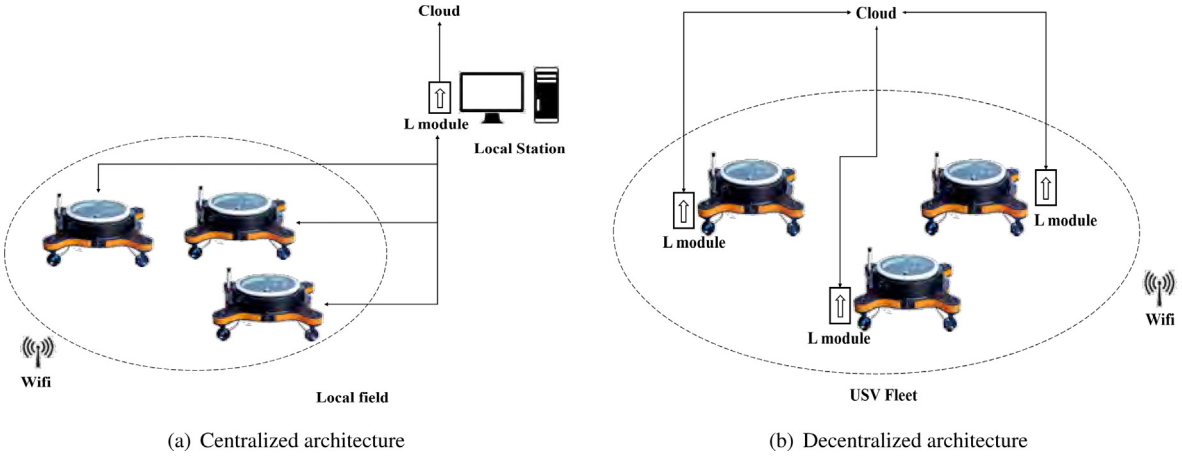


Fig. 2. Two possible architectures for cloud-based mission control system.

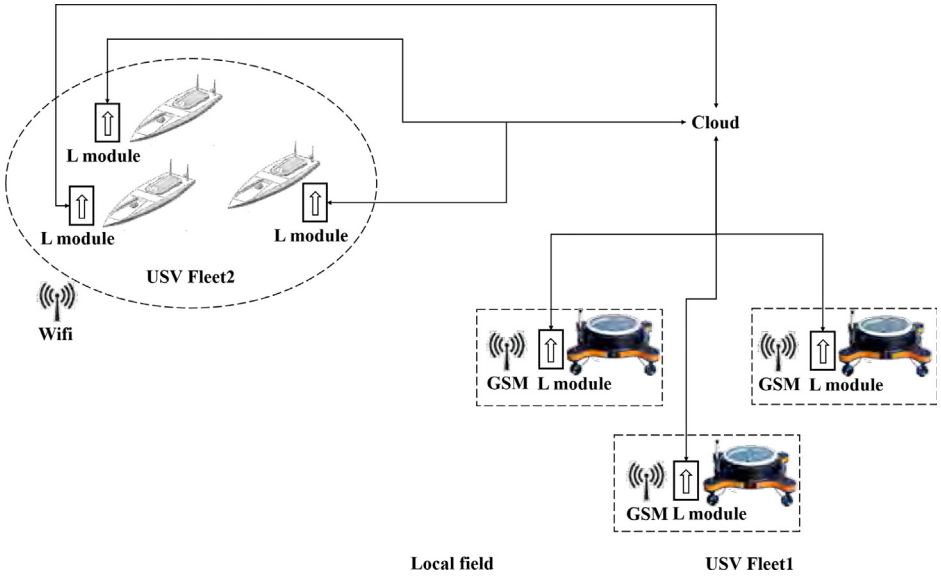


Fig. 3. Both local and mobile networks applied to the mission control of USV fleet with decentralized architecture.

now vehicles with access to mobile or Iridium communication can connect to the cloud from larger distances directly, while other, more localized vehicles, can still utilize a wireless connection with an Internet gateway towards the cloud as shown in Fig. 3. Due to apparent benefits and the readiness of the used equipment, this layout was used for the experiments presented in this paper.

3. System components

The proposed cloud-based mission control system for USV fleet consists of four main components: (a) cloud server, (b) remote user client software, (c) local interface plugin, (d) controlled vehicle.

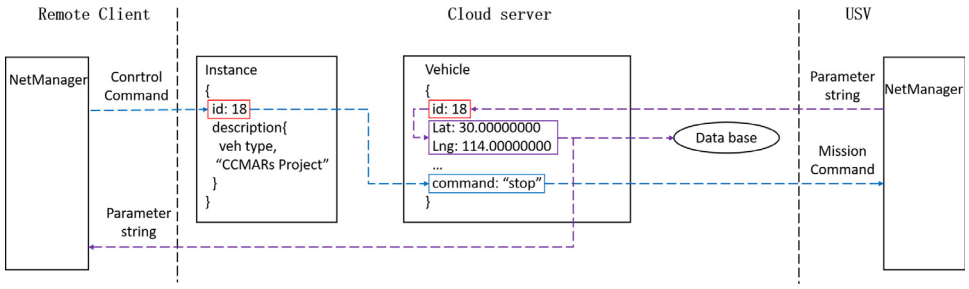


Fig. 4. Mechanism to match the command/parameter with corresponding USV through instance in back-end of Cloud server.

Table 1
Defined Server Services.

Service URI	Description
/monitor/client/get_instance	Get list of created instance in server
/monitor/client/connect_instance	Connect the selected instance
/monitor/client/get_param	Receive parameter from server
/monitor/client/submit_control	Submit mission command to server
/monitor/client/get_fbmessage	Get current control mode of system
/monitor/upper/start_instance	Create an instance in server
/monitor/upper/finish_instance	Remove created instance in server
/monitor/upper/update_param	Submit parameter to server
/monitor/upper/get_control	Get mission command from server
/monitor/upper/update_fbmessage	Switch the control mode of system

3.1. Cloud server

The mission control system is designed in client/server architecture. Therefore, the cloud-server back-end provides the corresponding HTTP backbone enabling the connection between the remote client and the local vehicle interfaces. The cloud-server is accessed through an API that provides a set of HTTP services which are applied through corresponding URIs. The vehicle subset contains methods for (de)registering new instances, data publishing and subscribing to commands. The remote client subset supports mirror functions such as connection of instance and transmission of mission command. In addition, control functions are available in the subset for publishing commands and control modes to individual vehicle instances. Definitions of Server Services are given in Table 1.

The interface client plugin creates for each USV an instance that includes the type of USV, time stamp and ID number. The instance ID is used to match a mission command with the selected USV as shown in Fig. 4. The ID is not only related to the instance, but also to the mission command and corresponding USV model object. Upon receiving the mission command, the cloud server based on the ID number searches for corresponding USV model object in the recorded list and updates the list with the new mission command. This mission command will be transmitted when the interface client with correct ID replies to the server. In opposite direction, the procedure to obtain the parameters from the server is also based on the ID number which is included in the web application message sent by remote client, as shown in Fig. 4.

Combine the correlative mechanism proposed above, the complete processing procedure of requests from local client and remote client in back-end of server can be described in two flowchart as shown in Fig. 5. For local client, first, local client requests to create its instance object in server back-end by calling the service of "*/monitor/upper/create_instance*", server will generate the only ID number for the created instance object and instantiate the vehicle data model with same ID as its property. When server receives the request to submit status parameters, the ID carried by request information will be parsed and used to find the mission command stored in corresponding data model object, then, the mission command will be serialized and sent back to remote client. If remote client requests to get mission command from cloud server in another thread, server also parses the ID from request message and captures mission command from the related data model object. For remote client, it is necessary for user to get all created instance in server back-end and the IDs of instances can be accessed by calling the service "*/monitor/client/get_instance*", then the controlled vehicles are ensured after the request which carries IDs of selected instances being sent to the server. When server receives the request to update mission command, the related model object will also be found and updated with new mission command, similarly, the status parameters will be sent back to remote client while server receives the request to get status of vehicles.

The cloud-server back-end layout is shown in Fig. 6. The back-end is divided into three parts containing four layers: communication interface, data pre-processing, global manager, data storage and sharing layer. The communication layer exposes the server service through the URI. The data pre-processing layer is responsible for gathering of received data and mission commands. These mission commands and data are matched with the vehicles through their instance ID in the global management layer, the ID of occupied instance will be collected in a pool and it will not be listed when server receives the request of getting instance by another remote user. Once the received data is analyzed, parsed and matched, the data storage layer stores the information into the data storage module. Finally, the data sharing module provides the methods for remote users and vehicles to extract required data from the storage module for visualization and other purposes. The data model integrates the motion attributes of target vehicle, the properties of vehicle are encapsulated into the file in Json format which will be transformed into a class structure in program through the Json interpreter, such as Warp and weft coordinate, north and east coordinate, linear velocity of vehicle, angular velocity of vehicle and sensor parameters. The data model of vehicle has the same definition in cloud server, remote client and local client to ensure the correctness of vehicle status parameters parsing.

Considering possible network fluctuations or outages, the instance should not be removed from the server immediately after sending. Some safety mechanism to ensure the control robustness should be proposed. The time limit for vehicle recovery after the network outage is set to one minute. Since cloud server is able to detect the termination of heart beat package from local interface, the created instance will be kept for a duration of the time limit. If the outage duration exceeds the time limit, the local observer/operator needs to login to the vehicle and create a new instance once the connection is re-established.

The control system web communication is implemented through Tornado, an open-source non-blocking web server framework with high speed and performance for concurrent processing. Tornado is lightweight framework that allows easy integration into a light programming projects written in Python. MongoDB is a distributed file storage that is selected as a database in the back-end of the cloud server. The data stored in MongoDB is in the form of a key–value pair, this structure is very similar to the class object with definition and value of its properties. Therefore, the processing of transmitted message is simple to implement, especially the storage and deserialization of parameters. Users can easily reach data after the experiment through Robo3T, a software used to visualize the data in MongoDB. IP address

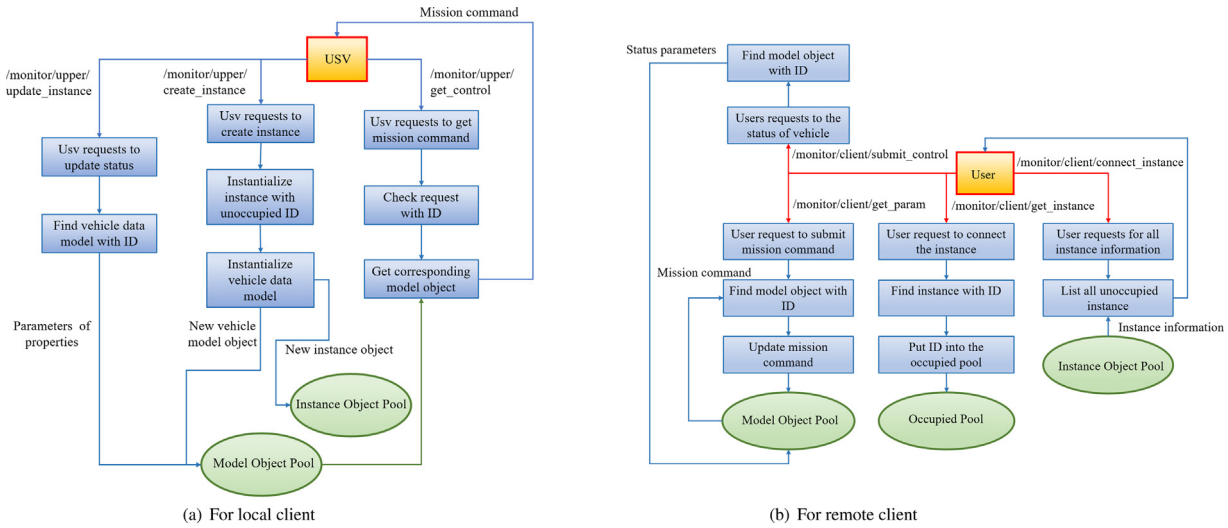


Fig. 5. Processing procedure of requests from local and remote client.

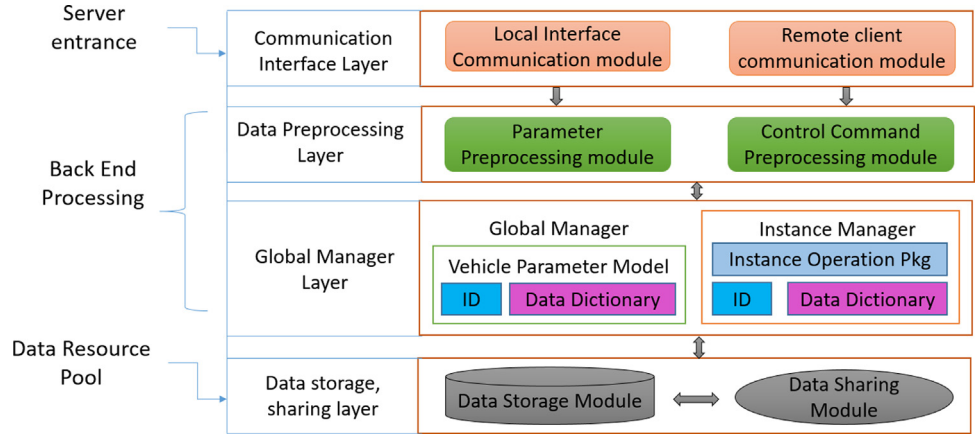


Fig. 6. The cloud-server back-end layout: three layers implement the model matching and messaging.

and correct credentials are only requirements needed to access the cloud-server database, making use and management of data stored in MongoDB very convenient.

3.2. Remote client

The remote client encompassed the graphical user interface with the corresponding back-end model. It includes five fundamental modules as shown in Fig. 7.

The parameter visualization module implements visualization extensions for supported navigation and status data objects received from the vehicles. Visualization could be implemented to directly presents deserialized vehicle attributes in textual form on the client's user interface or to show updated positions and trajectories of USVs on the map widget through vehicle markers. Based on the vehicle model provided from the cloud-server, the data storage module of the remote client splits the bundled vehicle data into separate parameters and stores them in local storage medium. This process is similar to the processing done on the cloud-server back-end and provides an additional storage redundancy. This feature is also required if local data availability for offline post-processing is required by the users. Mission commands are defined in the mission control module and include typical commands such as stop, point follow, keep station and path follow. These commands could be applied to a single vehicle or to the fleet of vehicles. The modular design of the mission control allows easy addition of new commands and mission modes. The user control module provides user interface elements for handling arbitration and allowing remote control on vehicles. The module also supports setting up a local manager, i.e. vehicle manager in the field. In this case, remote users can control the vehicle only when the local manager grants them permission. This avoids accidents or conflicting commands due to temporary loss or lack of situational awareness of the remote user.

NetManager is the most important module in remote client because it manages the invoking of web applications introduced previously. This module is divided into three layers as presented in Fig. 8. *HttpHelper* offers two web communication methods for web application managers. Application layer consists of five web application managers defined for different web missions. Any of these five web application managers could be instantiated by a corresponding function defined in *NetManager*. To establish communication with an USV, function *GetInstance* will be called first, to apply a list with information of created instance from cloud server. Then, an instance containing the name of USV is selected, the *ConnectInstance* function is invoked, a *ConnectInstanceApi* object will be created and the request message attached with ID number of corresponding instance is sent to server through *HttpGet* method provided by *HttpHelper*. *NetManager* submits the mission command through its function *submitControl* in thread and the mission should be executed. Functions *getParam* and *getMessage* are called in detached thread to get the parameters and control mode messages of USVs from cloud. *getFBMessage* is designed to receive some extra feedback messages related to vehicle control. In this study the only extra feedback message is the mode of control. Local controllers have the privilege

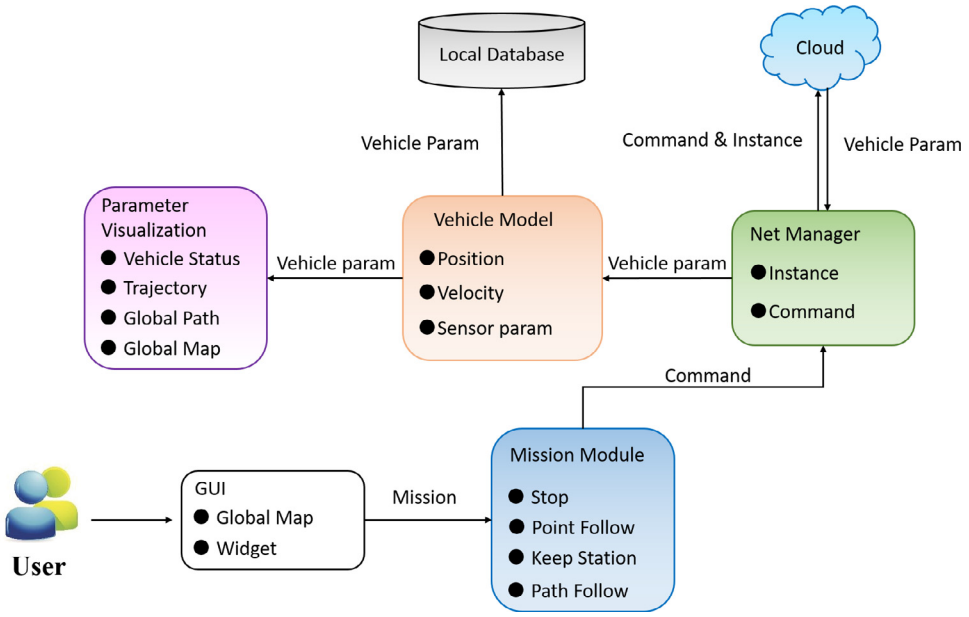


Fig. 7. The architecture of the remote client.

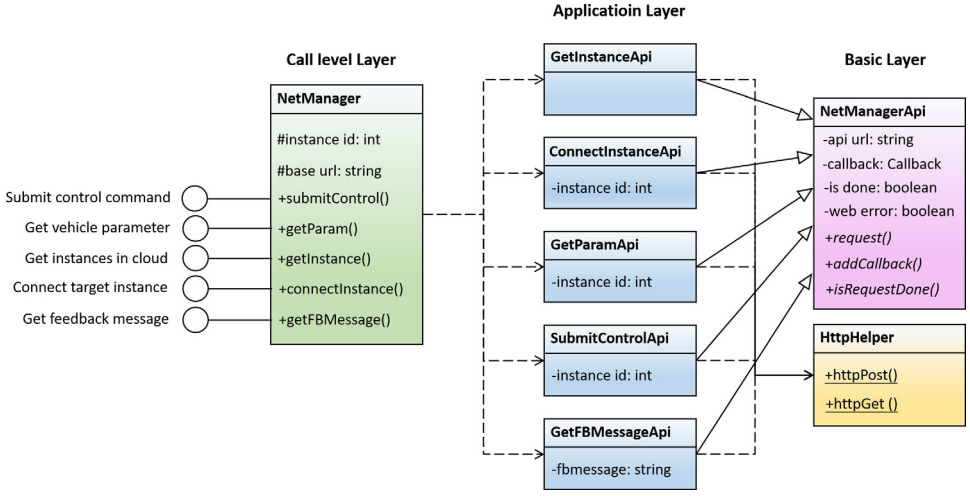


Fig. 8. UML class graph of net manager module in remote client.

to permit or forbid the remote access to ensure vehicles safety. In emergency situation, control priority should be transferred from the remote control to the local control. In this hierarchical structure, it is more convenient to manage the behavior of web communication in remote client and modify the invoking of web application.

Quantity of controlled USVs represents an input into remote client in order to determines the number of instantiated *NetManager* and vehicle models while the vehicle data model has the same definition in the back-end of cloud server. When remote client runs, control widgets are initialized first and the controlled number is recorded. Equal amount of *NetManager* objects are created to bound the instance created by USVs. These *NetManagers* send the mission commands to cloud and write received parameters into corresponding vehicle model objects. Parameters and positions of USVs are presented on the widgets and updated regularly. The procedures of components initialization and data stream routes in remote client are shown in Fig. 9. Mission manager consists of two main modules. First module handles mission types that are supported by vehicles. For H2Omni-X vehicles, four types of basic missions or services are supported and defined in the remote client: stop/abort, point following, station keeping and path following. These services represent the high-level commands used to control the vehicle remotely via cloud. The second module is the mission planing algorithm required to translate complex mission into series of basic services i.e. high-level commands. For example, if the controlled vehicle offers the service to follow a path consisting of multiply points, this path could be planned through the classic global path planning algorithms such as A*, Dijkstra, RRT, which are integrated in the algorithm module of mission manager. The available mission types and potential implementation are defined in the form of XML file and source file respectively and it is convenient to extend the mission module.

NET Framework (Microsoft) used for development of software based on Windows systems, boosts the Web communication and the control widget of the remote client. The control widget that includes maps for visualization is supported by a third-party open source library called GMap.NET.[1] It is very convenient tool for developers aiming to design interfaces that include markers and routes on the map. For the mission planning, this tool is used to transform parameters of the point or the path that consists of a series of points, into corresponding format

[1] https://github.com/judero01col/GMap.NET.

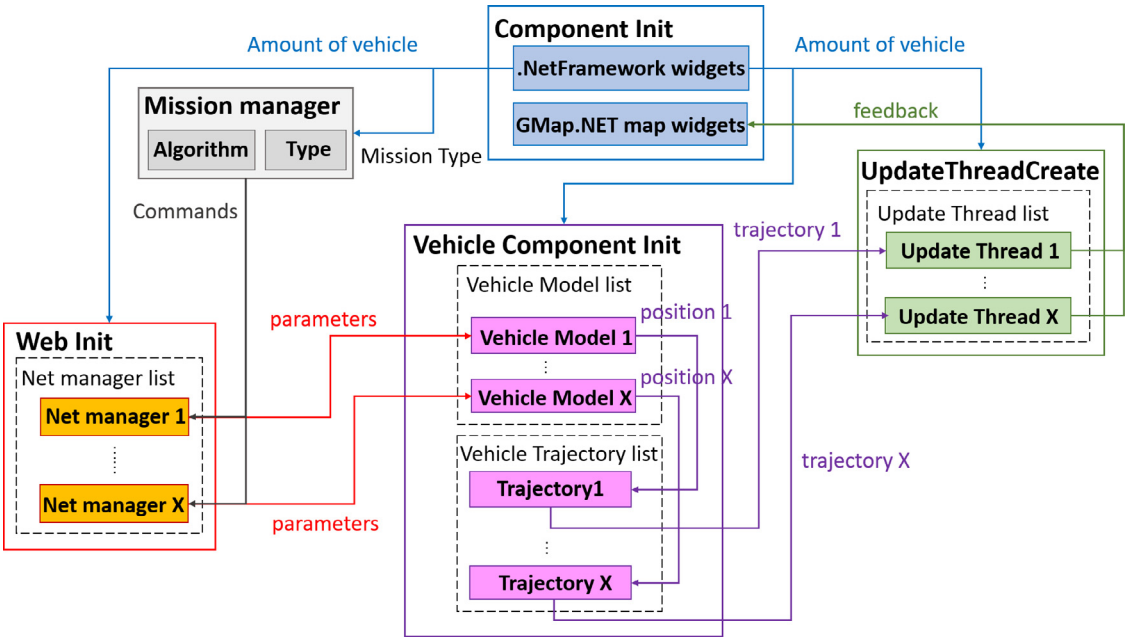


Fig. 9. Initialization procedure and route of data stream in remote client.

Table 2
Core ROS topic and service.

ROS Topic	Description
/position	Position of vehicle
/sensor_measurements	Parameters of vehicle sensors
/go2point/status	Point following status of vehicle
/instance	Messages of created instance
/ctrl_command	Received command from cloud
ROS Service	Description
commander/go2llpoint	Execute mission of point following
commander/stop_mission	Stop action
commander/dynamic_positioning_ll	Keep position

appropriate for GMap.NET to visualize the topological information. Furthermore, this library allows an integration of different kind of maps such as hybrid map, satellite map provided by e.g. Google, Bing and Yahoo, etc. Received parameters are stored as Excel files of .xls or .xlsx extensions.

3.3. Local vehicle interface

The remote client commands are received on the local vehicle interface through the cloud-server. The layout of the vehicle interface is shown in Fig. 10. The interface is divided into four modules. Mission, net manager and user control modules are conceptually similar to the ones in the remote client. The user module manages permissions, the mission module translates generic mission commands into vehicle specific calls and the net manager handles communication and data exchange with the cloud-server.

In the local interface client, the net manager is also designed in hierarchical structure similar to the remote client, as shown in Fig. 11. The biggest difference between local interface and remote client is that local interface has to be compatible with other software that run on an USV. For a ROS based USV, this is accomplished through the ROS interface. The ROS runtime "graph", running on the vehicle, is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. A ROS topic implements the concept of asynchronous data streaming, and the ROS service implements the concept of synchronous communication in style of remote procedure calls as detailed in Woodall, Liebhardt, Stonier, and Binney (2014). ROS subscriber is provided for developers to read the data published on the ROS topic, and the method to call ROS service is implemented by ROS client. Based on the communication mechanism in ROS, the vehicle status and sensor data are received from customized topics through the ROS subscriber submodule, and command execution is implemented through the ROS publisher submodule. The calling relationship between all defined ROS components and communication modules of ROS is shown in Table 2. For invoking the ROS application interface directly, the net manager is wrapped with these ROS components as *NetManagerROS* through the inheritance technology. This way the ROS environment and *NetManager* could be bridged naturally.

3.4. H2Omni-X USV platform

Experiments used the H2Omni-X USVs as shown in Fig. 12. The H2Omni-X is a versatile omni-directional surface vehicle designed for operation in shallow, crowded and/or confined environments where maneuverability is of essence. It supports a wide range of payloads for acoustic localization, underwater monitoring, bathymetry and sensors for environmental data sampling. Due to the versatility, H2Omni-X USVs have been used in number of different applications from diving support (Miskovic, Nad, & Rendulic, 2015), oil-spill detection (Vasilijević et al., 2017) to archaeology (Vasilijevic, Buxton, Sharvit, Stilinovic, Nad, Miskovic, Planer, Hale, & Vukic, 2015).

H2Omni-X was developed by the Laboratory for Underwater Systems and Technologies at University of Zagreb Faculty of Electrical Engineering and Computing in Croatia. It is currently produced by H2O-Robotics.[2] H2Omni-X is over-actuated which enables horizontal motion in any direction and orientation. It has a diagonal length of 1 m with 0.35 m in height, and it weighs in at 20–30 kg, depending on battery type and payload configuration. It is equipped with state of the art 9-axis inertial navigation system and RTK capable GNSS. The main navigation, guidance and control software is operating on the ROS framework and the vehicle is equipped with GSM and WiFi allowing the integration of the proposed cloud-based framework.

H2Omni-X has a multi-level control structure, implemented in a top-down approach. In Autonomous mode, high-level commands are directly set by the remote client through the cloud server in the form

[2] https://h2o-robotics.com/.

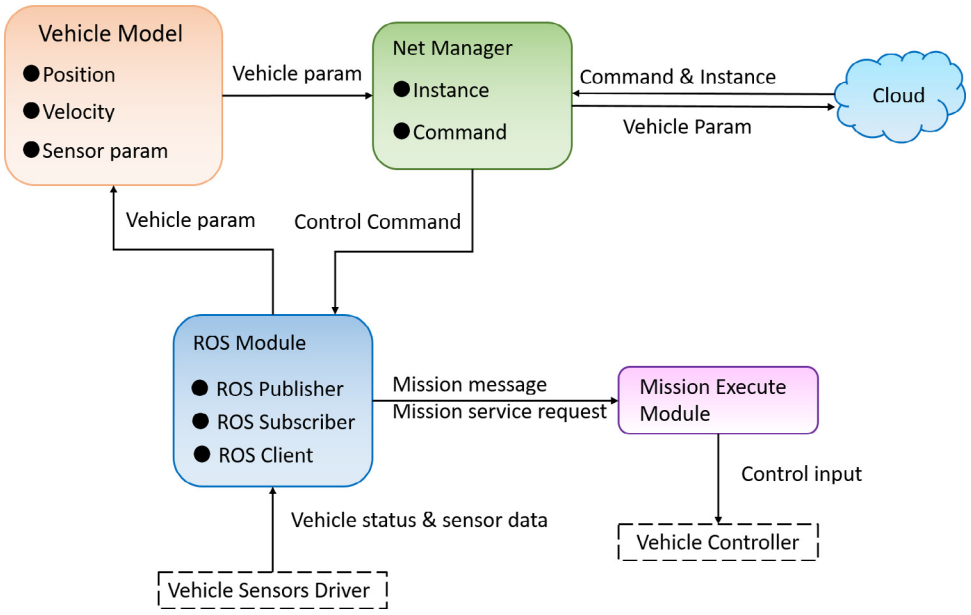


Fig. 10. The architecture of the local interface.

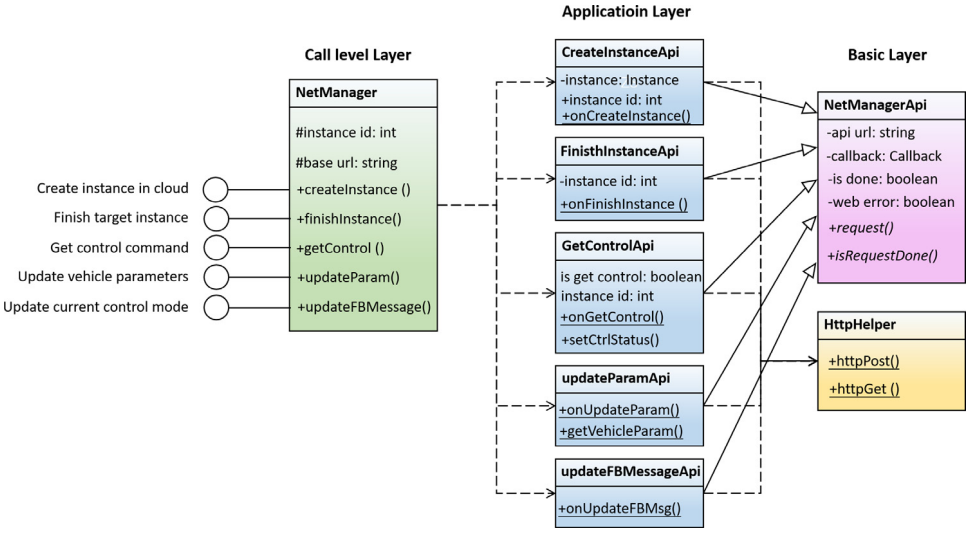


Fig. 11. UML class graph of the net manager module in local interface.

Fig. 12. The H2Omni-X USV and the USV fleet.

of: stop/abort, point following, station keeping and path following commands. High-level commands contain only the mission-necessary parameters and information such as mission type, time stamp, point(s), speed and orientation. These parameters are parsed as soon as they

Table 3
Average communication delay of a subset of available Remote system service.

Remote system service	Average delay (ms)
/monitor/client/get_instance	44
/monitor/client/connect_instance	37
/monitor/client/get_param	104
/monitor/client/submit_control	31
/monitor/client/get_fbmessage	31
/monitor/upper/start_instance	34
/monitor/upper/finish_instance	39
/monitor/upper/update_param	127
/monitor/upper/get_control	34
/monitor/upper/update_fbmessage	33

are received by an USV. High-level commands are transferred to the intermediate-level vehicle controls: Heading control, Dynamic Positing and under-actuated and fully-actuated line following controls. Point(s), speed and orientation set by high-level commands, are inputs to the Intermediate-level control system. Combining different Intermediate-level functionalities, desired behaviors could be obtained. Stop/Abort command stops all vehicle's thrusters. The point following command makes vehicle go to the commanded point with commanded speed and orientation (if applicable). On the station keeping command, vehicle keeps commanded position with commanded orientation (if applicable). Path following is complex command that is executed as a multiple point following. H2Omni-X Low-level controls are speed controls: surge, sway and yaw rate controls.

4. Experiments and results

Before real experiments, simulations were performed to validate whether the H2Omni-X can execute the proposed missions through the cloud infrastructure. The provided cloud server was located in Frankfurt, Germany. Pool and open sea experiments took place in Biograd na Moru, Croatia.

4.1. Simulation

Prior to experiments with the simulator, the expected server–client communication delay has been measured. The delay has been tested for a subset of system service. The average delays are shown in Table 3. Results showed that the average delay was low across all of the methods. Remote client service of *"/monitor/client/get_param"* and local service of *"/monitor/upper/update_param"* had a higher delay due to larger payload. However, the delay of services of command submitting and getting (e.g. *"/monitor/client/submit_control"*) were small as well as the instance related services. Due to the fact that Intermediate- and Low-level control loops are closed locally, vehicle is able to perform e.g. line following or dynamic positioning autonomously. High-level commands are only used to change the mission way-point, station keeping position or to abort the mission. Marine vehicles are systems with slow dynamics, mostly operating in open wide environment and therefore, communication delay in e.g. way-point update, on the order of seconds would have light impact on control performance. From the perspective of data collection and online processing the delays were still within margins to allow near real-time mission (re)planning.

The simulation experiments consisting of point following and path following, were performed on the simulator. The vehicle simulator was provided in the form of Docker[3] container. Vehicle kinetics, kinematics and sensors were simulated while all other software components, including navigation and control, were identical to those located on the vehicle. The guidance API was accessible directly from ROS or via `rosbridge`.[4]

[3] www.docker.com.
[4] robotwebtools.org.

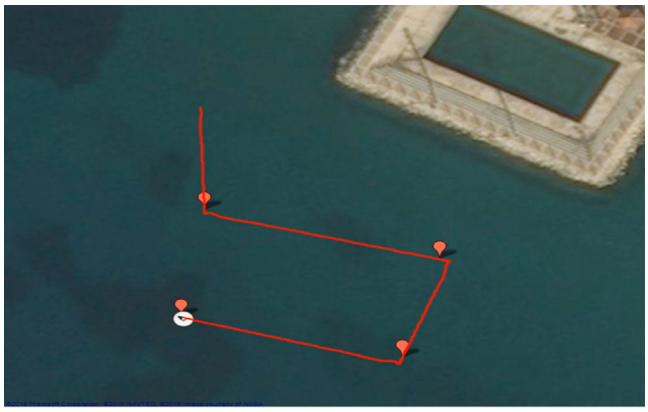


Fig. 13. Way-point simulation: marked target points were sent sequentially to the vehicle in the mission.

Fig. 14. Path following simulation: the blue line shows the planned path generated by modified A* algorithm (Wang & Xiang, 2018). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

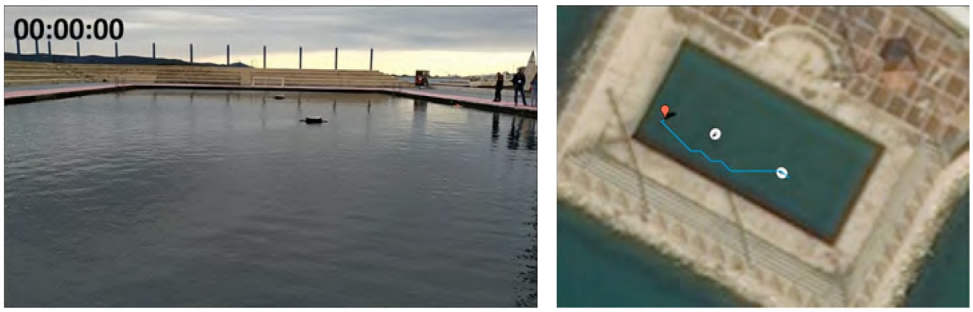
The way-points for the simulation experiment were selected by the user through the map widget in the remote client. In addition to single way-point, the users could provide a set of way-points based on which, the integrated path-planning module would generate desired path. This path was transferred to the vehicle/simulator for execution. Single way-point and path-following simulation results are shown in Figs. 13 and 14, respectively. The markers (shown as red icons) represent the selected way-points the vehicle should visit. Fig. 14 shows two planned paths with a set of way-points generated by the modified global path planning algorithm, the first one has been completely followed by the simulated vehicle and the trajectory of vehicle is shown in red and the second planned path is shown in blue in Fig. 14.

These simulations focused on validating functionality rather than tracking performance. Although the simulation scenario encompassed only a single vehicle, the simulations can be easily scaled to multiple vehicles to validate fleet operation.

4.2. Pool experiments

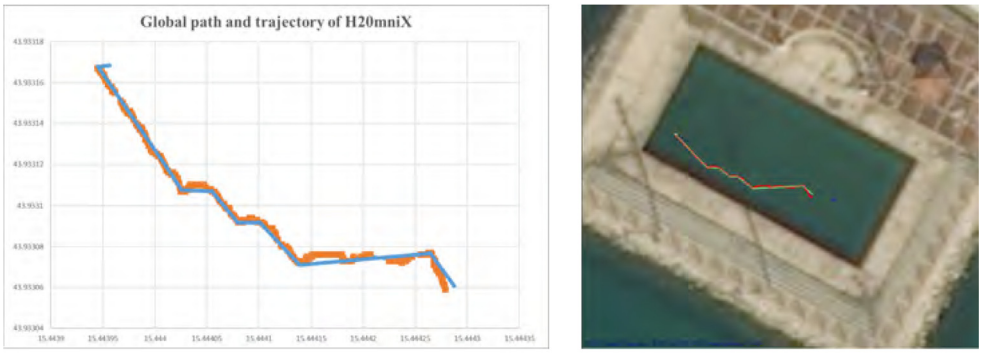
The first real-life experiments were performed at Biograd na Moru, in a controlled pool environment. Experiments included line following, station keeping and path following with two vehicles which are named as Proteus and Platripos. During the experiments, the vehicle velocity limit was set to 0.5 m/s. Similar to the simulation experiments, the cloud server was located in Frankfurt, Germany, while remote client was operated by the experimenter beside the pool.

Figures show the display of the remote client (on the right) and corresponding frames extracted from the video recorded locally (on the



(a) Initial status of H2OmniX

(b) Intermediate mission progress



(c) Trajectory of H2OmniX

Fig. 15. Path following experiment in the pool: frames extracted from the video recorded locally (left), the global map with the path, vehicles and target in remote client (right).

left). Two types of experiments were performed: simultaneous point following with two vehicles and path following with one vehicle while another vehicle represented the drifting obstacle. In case of the path-following experiment, shown in Fig. 15, the desired path around the obstacle is shown in blue in Fig. 15(b) and (c) while the practical path of vehicle is shown in red. The row axis and col axis represent latitude and longitude of vehicle respectively. It can be noted that the path passes next to the obstacle, i.e. the drifting second vehicle.

During way-points guidance, shown in Fig. 16, two diagonal points, relative to current vehicle positions, were selected as target points. All vehicle information, including position estimates, were stored in the cloud and could be easily retrieved. This feature prove to be very convenient for analysis and review of the vehicle status and trajectories generated during the experiment by both vehicles. The latitude and longitude coordinates and vehicle paths, presented in Fig. 16(c) and (d) were retrieved directly from the cloud after the experiment and replayed in the map module, the row axis and col axis represent latitude and longitude of vehicle respectively as same as Fig. 15.

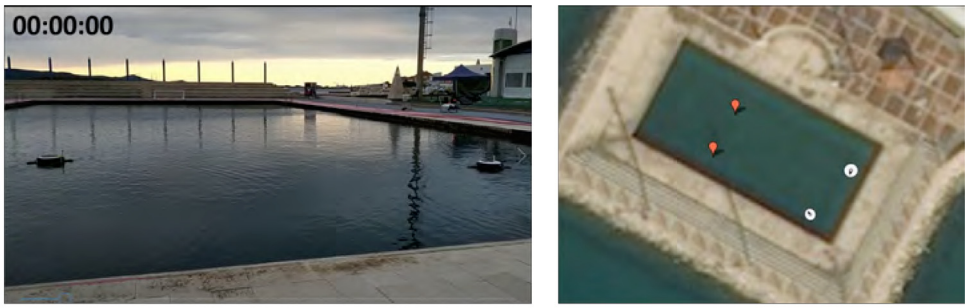
The vehicle information were stored in MongoDB[5] and were extracted through Robo3T,[6] a data visualization tool for MongoDB. In addition to position estimates, the information stored in the database includes vehicle model parameters, sensor measurements and other navigation, guidance and control signals. These data can be easily accessed by any user with corresponding credentials.

4.3. Sea trails

Mission control experiments for USV fleet at sea were planned and executed in Biograd Na Moru, Croatia. Three H2Omni-X USVs named Proteus, Platripos and Cres were used in experiments. Experiments envisioned path following to be performed by two vehicles and station/target keeping to be performed by the third vehicle. During the experiments, the maximum velocity of each vehicle was set to 0.5 m/s. The cloud server was located in Frankfurt, Germany. The "remote" user was not present at the experimental site but operated the remote client from the nearby hotel Ilirija. The user monitored and remotely controlled the fleet during the sea trails.

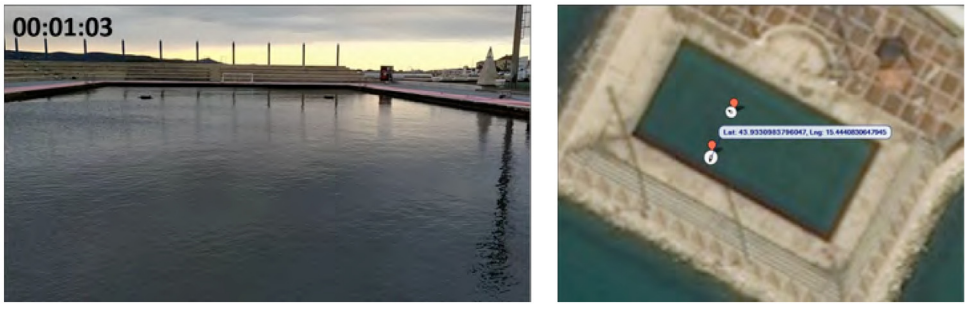
Path following or station keeping way-points for each of vehicles were selected through the remote client. These way-points were attached to the corresponding mission command and sent to vehicles via cloud server. The progress of the experiment at sea with three vehicles is given in Fig. 17. Frames extracted from the video recorded on the site

[5] mongodb.com.
[6] robomongo.org.

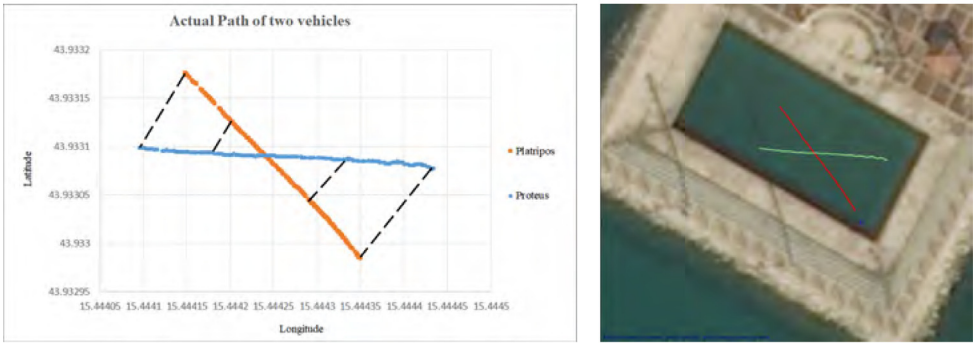


(a) Initial status of two USVs

(b) Intermediate mission progress



(c) Target points reached

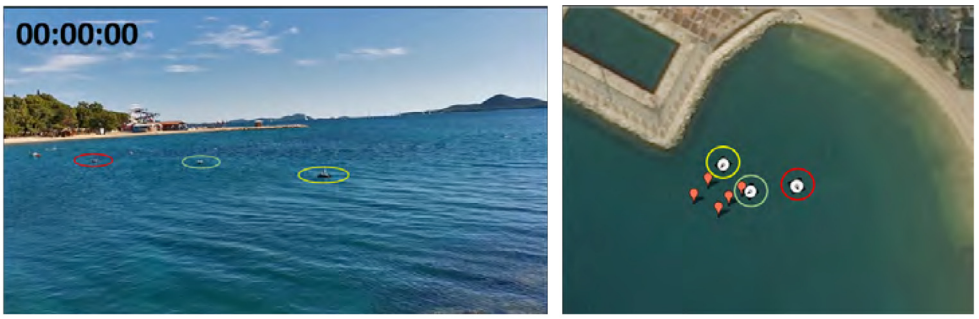


(d) Coordinated trajectories of two USVs shown on the map

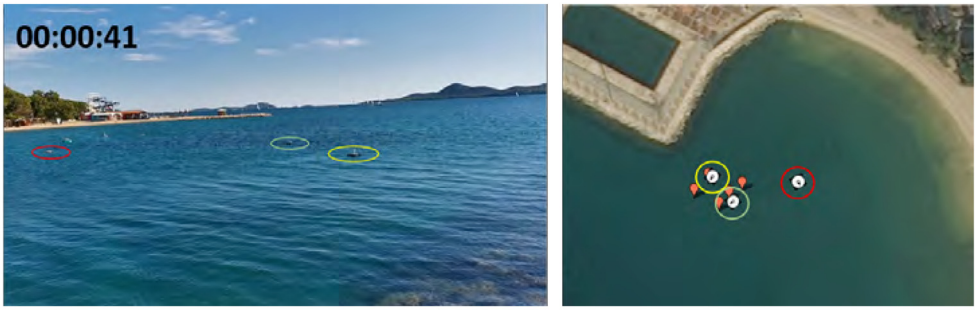
Fig. 16. Point following experiment in the pool: frames extracted from the video recorded locally (left), the global map showing paths, vehicles and targets in remote client (right).

are presented on the left and the global map with the path, vehicles and target shown in remote client is presented on the right. Selected way-points for vehicles to follow are represented by red bubble markers while vehicles in the experimental arena are represented by the circles with different colors on the interface of the remote client. Figure Fig. 17 shows that two vehicles performing the path following and third vehicle performing the station keeping, successfully accomplished the mission in real environment, exposed to wind and waves.

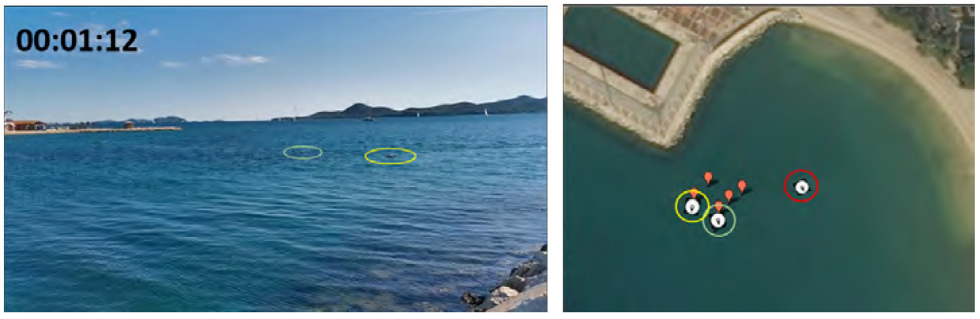
The last experiment tested the robustness of the system in case of network outage while the mission was in progress. There are different options and action that could be set for vehicle to follow in emergency (e.g. during the outage), such as to continue the mission based on the



(a) Initial status of three USVs



(b) Intermediate mission progress



(c) Target point reached

(d) Trajectories of USVs during robustness experiment

Fig. 17. Experiment at sea: Circles with different colors match the vehicles in scenery to markers in remote client.

last mission command received, to stop and station keep or to stop and drift until the network is up and running again. We have chosen to let the vehicle drift, just to get the clearly noticeable spatial visualization of the network outage on the remote client interface. Using the remote interface, two vehicles were set to station keep while the Platripos vehicle was set to path following, similar to the previous experiment. During the experiment, Platripos network was cut off manually for 1 min and vehicle drifted in south-east direction due to the wind and waves. After the network recovery, local interface bridged Platripos and cloud server, sending again the same mission command to the vehicle. As a result, Platripos continued to move towards the target point and mission was finalized successfully. The Platripos path, presented on the

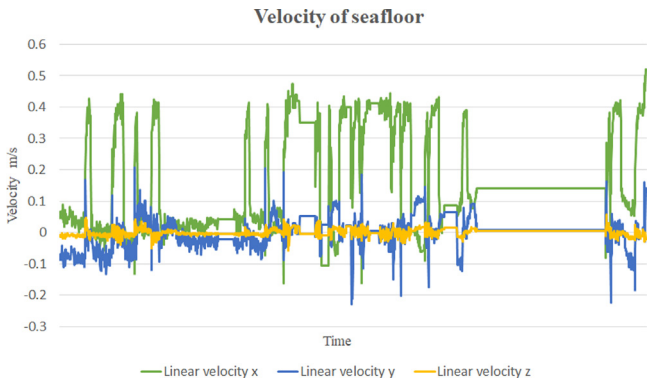


Fig. 18. Seafloor velocity collected by proteus USV.

map as green line, is shown in right part of Fig. 17(d). Compared to the previous experiment, the achieved path is an polyline rather then a straight line due to vehicle drift during the network outage.

The same as in previous experiments, data was stored using MongoDB and visualized using Robo3T. It is important to mention that data was not lost during the network outage but was stored locally and transferred after the network recovery. As an example of continuity of the collected data and use of Robo3T tool, seafloor velocity collected by Proteus during the experiments is given in Fig. 18.

5. Conclusions

The paper presented a cloud-based framework for mission control of USV fleet. The proposed system architecture was presented with individual components in detail. The experimental results show that the proposed system is feasible for coordinated control of multiple H2Omni-X vehicles, or any unmanned vehicles satisfying the current software/hardware requirements for the framework.

First step in testing the feasibility were experiments performed on the simulator. The goal was to validate whether the H2Omni-X could execute the proposed missions through the cloud infrastructure. The way-points in the mission were selected by the user through the map widget on the remote client and then the connected path was transferred to the vehicle/simulator for execution. Experiments on the simulator validated all targeted functionalities, therefore, we proceeded to experiments on real vehicles.

The field experiments were performed both, in a controlled environment i.e. in the pool, and in the operational environment i.e. at sea. Experiments consisted of missions with multiple vehicles performing path following or station keeping. The missions were planned through the remote client and were successfully accomplished in both environments.

This study also explored the robustness of the system in case of network outage during the ongoing mission. Results showed that during the network outage, vehicle stopped and drifted due to the wind and waves as expected, and after the network recovery, vehicle continued to move towards the target point and mission was finalized successfully.

One of the biggest challenges in the proposed framework, is that the mission control is sensitive to the communication delay between vehicles and the cloud-server. The approach taken in this study was to use high-level commands such as way-point guidance, to remotely control fleet of vehicles. Lower-level controls such as speed or heading control were executed locally on the vehicle. Experiments showed that communication delay was tolerable, but bearing in mind that both, the server and clients where located in the same region Europe, within 1200 km. However, for a sea-trials with remote cloud-based control, located in a different, remote region there are still technical concerns to be addressed. Therefore, we tested delays for a cloud-server located in Wuhan, China. Results showed that delays are getting close to 1 s. However, these delays could still be considered tolerable in guidance of vehicles with slow dynamics such as marine vehicles. When crossing regional boundaries with these systems, instead of sending direct guidance commands, the mission control could be limited to supervision-level commands.

Future research could be focused on simultaneous control and cloud-based management of more marine vehicles, yet mission complexity could be increased for specific applications, e.g. towards distributed environmental sensing and monitoring. Another interesting topic would be to expand the concept towards a more heterogeneous fleet by including different vehicles, and coordinated them in a team with different vehicle types, for instance, wheeled mobile robots and unmanned aerial vehicles.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Ali, S. S., Hammad, A., & Tag Eldien, A. S. (2018). Mc2ps: Cloud-based 3-d place recognition using map segmentation coordinates points. *IEEE Communications Letters*, *22*(8), 1560–1563. http://dx.doi.org/10.1109/LCOMM.2018.2833122.

Ansari, F. Q., Pal, J. K., Shukla, J., Nandi, G. C., & Chakraborty, P. (2012). A cloud based robot localization technique. In M. Parashar, D. Kaushik, O. F. Rana, R. Samtaney, Y. Yang, & A. Zomaya (Eds.), *Contemporary Computing* (pp. 347–357). Berlin, Heidelberg: Springer Berlin Heidelberg, ISBN: 978-3-642-32129-0.

Cai, M., Wang, Y., Wang, S., Wang, R., & Tan, M. (2019). Ros-based depth control for hybrid-driven underwater vehicle-manipulator system. In *2019 Chinese Control Conference (CCC)* (pp. 4576–4580). http://dx.doi.org/10.23919/ChiCC.2019.8865762.

Chu, Z., Xiang, X., Zhu, D., Luo, C., & Xie, D. (2018). Adaptive fuzzy sliding mode diving control for autonomous underwater vehicle with input constraint. *International Journal of Fuzzy Systems*, *20*(5), 1460–1469.

Doriya, R., Chakraborty, P., & Nandi, G. C. (2012). Robotic services in cloud computing paradigm. In *2012 International Symposium on Cloud and Services Computing* (pp. 80–83). http://dx.doi.org/10.1109/ISCOS.2012.24.

Gallimore, E., Stokey, R., & Terrill, E. (2018). Robot operating system (ros) on the remus auv using recon. In *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)* (pp. 1–6). http://dx.doi.org/10.1109/AUV.2018.8729755.

Guo, H., Wu, X., & Li, N. (2018). Action extraction in continuous unconstrained video for cloud-based intelligent service robot. *IEEE Access*, *6*, 33460–33471. http://dx.doi.org/10.1109/ACCESS.2018.2842088.

Hartanto, R., & Eich, M. (2014). Reliable, cloud-based communication for multi-robot systems. In *2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)* (pp. 1–8). http://dx.doi.org/10.1109/TePRA.2014.6869142.

Hu, G., Tay, W. P., & Wen, Y. (2012). Cloud robotics: architecture, challenges and applications. *IEEE Network*, *26*(3), 21–28. http://dx.doi.org/10.1109/MNET.2012.6201212.

Kaliski, B. (2008). Multi-tenant cloud computing: from cruise liners to container ships. In *2008 Third Asia-Pacific Trusted Infrastructure Technologies Conference*. http://dx.doi.org/10.1109/APTC.2008.16, pp. 4–4.

Karimi, H. R., Duffie, N. A., & Dashkovskiy, S. (2010). Local capacity H_∞ control for production networks of autonomous work systems with time-varying delays. *IEEE Transactions on Automation Science and Engineering*, *7*(4), 849–857.

Koubaa, A., & Qureshi, B. (2018). Dronetrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet. *IEEE Access*, *6*, 13810–13824. http://dx.doi.org/10.1109/ACCESS.2018.2811762.

Krupinski, S., Desouche, R., Palomeras, N., Allibert, G., & Hua, M.-D. (2015). Pool testing of AUV visual servoing for autonomous inspection. *IFAC-PapersOnLine*, *48*(2), 274–280. http://dx.doi.org/10.1016/j.ifacol.2015.06.045.

Lai, Y., & Cheng, J. (2014). A cloud-storage RFID location tracking system. *IEEE Transactions on Magnetics*, *50*(7), 1–4. http://dx.doi.org/10.1109/TMAG.2014.2303810.

Lorencik, D., & Sincak, P. (2013). Cloud robotics: current trends and possible use as a service. In *2013 IEEE 11th International Symposium on Applied Machine Intelligence and Informatics (SAMI)* (pp. 85–88). http://dx.doi.org/10.1109/SAMI.2013.6480950.

Mancini, A., Frontoni, E., & Zingaretti, P. (2015). Development of a low-cost unmanned surface vehicle for digital survey. In *2015 European Conference on Mobile Robots (ECMR)* (pp. 1–6). http://dx.doi.org/10.1109/ECMR.2015.7324189.

McGillivary, P. A., & Zykov, V. (2016). Ship-based cloud computing for advancing oceanographic research capabilities. In *OCEANS 2016 MTS/IEEE Monterey* (pp. 1–7). http://dx.doi.org/10.1109/OCEANS.2016.7761339.

Mendonça, R., Santana, P., Marques, F., Lourenço, A., Silva, J., & Barata, J. (2013). Kelpie: a ros-based multi-robot simulator for water surface and aerial vehicles. In *2013 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 3645–3650). http://dx.doi.org/10.1109/SMC.2013.621.

Miskovic, N., Nad, D., & Rendulic, I. (2015). Tracking divers: An autonomous marine surface vehicle to increase diver safety. *IEEE Robotics & Automation Magazine*, *22*(3), 72–84. http://dx.doi.org/10.1109/MRA.2015.2448851.

Mohanarajah, G., Hunziker, D., D'Andrea, R., & Waibel, M. (2015). Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*, *12*(2), 481–493. http://dx.doi.org/10.1109/TASE.2014.2329556.

Mohanarajah, G., Usenko, V., Singh, M., D'Andrea, R., & Waibel, M. (2015). Cloud-based collaborative 3D mapping in real-time with low-cost robots. *IEEE Transactions on Automation Science and Engineering*, *12*(2), 423–431. http://dx.doi.org/10.1109/TASE.2015.2408456.

Peng, Z., Wang, J., & Han, Q. (2019). Path-following control of autonomous underwater vehicles subject to velocity and input constraints via neurodynamic optimization. *IEEE Transactions on Industrial Electronics*, *66*(11), 8724–8732.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). Ros: an open-source robot operating system. In *Workshops at the IEEE International Conference on Robotics and Automation*.

Riazuelo, L., Tenorth, M., Di Marco, D., Salas, M., Galvez-Lopez, D., Mosenlechner, L., et al. (2015). Roboearth semantic mapping: A cloud enabled knowledge-based approach. *IEEE Transactions on Automation Science and Engineering*, *12*(2), 432–443. http://dx.doi.org/10.1109/TASE.2014.2377791.

Saha, O., & Dasgupta, P. (2018). A comprehensive survey of recent trends in cloud robotics architectures and applications. *Robotics*, *7*(3), http://dx.doi.org/10.3390/robotics7030047.

Salmeron-Garca, J., Inigo-Blasco, P., Daz-del-Ro, F., & Cagigas-Muniz, D. (2015). A tradeoff analysis of a cloud-based robot navigation assistant using stereo image processing. *Ocean Engineering*, *12*(2), Article 106840. http://dx.doi.org/10.1109/TASE.2015.2403593.

Shen, C., & Shi, Y. (2020). Distributed implementation of nonlinear model predictive control for AUV trajectory tracking. *Automatica*, *115*, Article 108863. http://dx.doi.org/10.1016/j.automatica.2020.108863.

Silva, M., Moita, F., Nunes, U., Garrote, L., Faria, H., & Ruivo, J. (2012). Isrobotcar: the autonomous electric vehicle project. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4233–4234). http://dx.doi.org/10.1109/IROS.2012.6386292.

Sivcev, S., Rossi, M., Coleman, J., Dooly, G., Omerdic, E., & Toal, D. (2018). Fully automatic visual servoing control for work-class marine intervention ROVs. *Control Engineering Practice*, *74*, 153–167.

Song, J., Gupta, S., Hare, J., & Zhou, S. (2013). Adaptive cleaning of oil spills by autonomous vehicles under partial information. In *2013 OCEANS - San Diego* (pp. 1–5). http://dx.doi.org/10.23919/OCEANS.2013.6741246.

Sun, B., Zhu, D., Tian, C., & Luo, C. (2019). Complete coverage autonomous underwater vehicles path planning based on glasius bio-inspired neural network algorithm for discrete and centralized programming. *IEEE Transactions on Cognitive and Developmental Systems*, *11*(1), 73–84.

Vasilijevic, A., Buxton, B., Sharvit, J., Stilinovic, N., Nad, D., Miskovic, N., Planer, D., Hale, J., & Vukic, Z. (2015). An asv for coastal underwater archaeology: the pladypos survey of caesarea maritima, israel. In *OCEANS 2015 - Genova* (pp. 1–7). http://dx.doi.org/10.1109/OCEANS-Genova.2015.7271495.

Vasilijević, A., Nad, D., Mandić, F., Mišković, N., & Vukić, Z. (2017). Coordinated navigation of surface and underwater marine robotic vehicles for ocean sampling and environmental monitoring. *IEEE/ASME Transactions on Mechatronics*, *22*(3), 1174–1184. http://dx.doi.org/10.1109/TMECH.2017.2684423.

Vick, A., Vonásek, V., Pěnička, R., & Krüger, J. (2015). Robot control as a service – towards cloud-based motion planning and control for industrial robots. In *2015 10th International Workshop on Robot Motion and Control (RoMoCo)* (pp. 33–39). http://dx.doi.org/10.1109/RoMoCo.2015.7219710.

Wang, Y., & Han, Q. (2016). Network-based fault detection filter and controller coordinated design for unmanned surface vehicles in network environments. *IEEE Transactions on Industrial Informatics*, *12*(5), 1753–1765. http://dx.doi.org/10.1109/TII.2016.2526648.

Wang, N., Karimi, H. R., Li, H., & Su, S. (2019). Accurate trajectory tracking of disturbed surface vehicles: A finite-time control approach. *IEEE/ASME Transactions on Mechatronics*, *24*(3), 1064–1074.

Wang, L., Liu, M., & Meng, M. Q. . (2015). Real-time multisensor data retrieval for cloud robotic systems. *IEEE Transactions on Automation Science and Engineering*, *12*(2), 507–518. http://dx.doi.org/10.1109/TASE.2015.2408634.

Wang, Z., & Xiang, X. (2018). Improved astar algorithm for path planning of marine robot. In *2018 37th Chinese Control Conference (CCC)* (pp. 5410–5414). http://dx.doi.org/10.23919/ChiCC.2018.8483946.

Woodall, W., Liebhardt, M., Stonier, D., & Binney, J. (2014). ROS topics: Capabilities [ROS topics]. *IEEE Robotics & Automation Magazine*, *21*(4), 14–15. http://dx.doi.org/10.1109/MRA.2014.2360622.

Wu, H., Lou, L., Chen, C., Hirche, S., & Kuhnlenz, K. (2013). Cloud-based networked visual servo control. *IEEE Transactions on Industrial Electronics*, *60*(2), 554–566. http://dx.doi.org/10.1109/TIE.2012.2186775.

Xiang, X., Yu, C., Zhang, Q., Wilson, P. A., & Xu, G. (2020). Manoeuvring-based actuation evaluation of an AUV with control surfaces and through-body thrusters. *Applied Ocean Research*, *96*, Article 102046. http://dx.doi.org/10.1016/j.apor.2019.102046.

Zhang, Q., Lapierre, L., & Xiang, X. (2013). Distributed control of coordinated path tracking for networked nonholonomic mobile vehicles. *IEEE Transactions on Industrial Informatics*, *9*(1), 472–484. http://dx.doi.org/10.1109/TII.2012.2219541.

Zhang, Q., Zhang, J., Chemori, A., & Xiang, X. (2018). Virtual submerged floating operational system for robotic manipulation. *Complexity*, (2), 1–18. http://dx.doi.org/10.1155/2018/9528313.

Zhang, L., Zhang, H. Y., Fang, Z., Xiang, X., Huchard, M., & Zapata, R. (2017). Towards an architecture-centric approach to manage variability of cloud robotics. Computing Research Repository, abs/1701.03608 http://arxiv.org/abs/1701.03608.

Zheng, Y., Deng, F., Zhu, Q., & Deng, Y. (2014). Cloud storage and search for mass spatio-temporal data through proxmox ve and elasticsearch cluster. In *2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems* (pp. 470–474). http://dx.doi.org/10.1109/CCIS.2014.7175781.

Zheng, Z., Sun, L., & Xie, L. (2018). Error-constrained LOS path following of a surface vessel with actuator saturation and faults. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *48*(10), 1794–1805.