Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 11233–11238

Distributed Real-time Control Architecture for ROS-based Modular Robots

Eduardo Munera * **Jose-Luis Poza-Lujan** *
Juan-Luis Posadas-Yague * **Jose Simo** *
J.Francisco Blanes Noguera *
* *Institute of Control Systems and Industrial Computing (ai2)*
Universitat Politècnica de València (UPV)Camino de vera, s/n. 46022
Valencia, Spain, (e-mail: emunera, jopolu, jposadas,
pblanes@ai2.upves)

Abstract:
Nowadays ROS (Robot Operating System) based platforms have been widely used in state of art robot researches because of providing reliable mechanisms for fast robot development and algorithm reuse. Among them, modular robots are presented as a Distributed Control System (DCS), in which the data supply and the flow rates must be ensured to guarantee a proper execution. Although ROS network architecture provides a solution for distributed communication, it entails some disadvantages. Real-time constraints assurance and large networks management are some of the main problems. Therefore, this works introduces a ROS-compatible solution for Smart Resource services decoupling on modular robots. Smart Resources are defined as cyber-physical systems that provide high-level operation services. The design and analysis of a new distributed communication architecture for accessing Smart Resource services is addressed along this document. This architecture is called ROS Multi-Peer Architecture (RMPA) and features real-time communication, fully decoupled ROS device execution, and device discovery. So, the integration of RMPA in ROS based developments provides real-time access to the high-level supplied by the Smart Resources. This integration promotes the development of ROS-compatible robot modules based on the Smart Resource design. Next, a set of tests is presented to compare RMPA and ROS communication performance. Moreover, a detailed study of the RMPA performance is also addressed. Finally, as a result of these experiments main advantages and withdrawals of this solution are discussed leading to the establishment of future work and enhancements.

Keywords: Distributed computer control systems; Real-time communications; Decoupled subsystems and Mobile robots.

1. INTRODUCTION

Modular robotics offers a wide range of advantages such as: module utilization, process decentralization, and ubiquity. Despite of this, Distributed Control Systems (DCSs) must fulfill some strict communication requirements. Although ROS (Robot Operating System) based systems offer many advantages, it provide some disadvantages when dealing to modular robotics. The high number of devices connected to the ROS network and the need to export the ROS master from a well-known master device are big withdrawals for modular robot development.

In a ROS distributed network the master device is defined as the device that executes the ROS core. ROS core includes the execution of the ROS master instance, which provides topic name services and manages the peers connection. The dependence on the master device arise some problems. The first problem is the need of ROS devices to interact with the ROS master instance in order to exchange ROS topic data. Therefore, devices cannot operate into a fully isolated way, being unable to perform if the masters instance is not running or has been shutdown. Next problem is to make the master device (which executes the ROS master instance) reachable from every device. To reach the master device, its IP address and port number must be known by all the devices in the network. Therefore, these devices must be previously configured in order to set a distributed ROS system. Any change in the master device address requires the system to be reconfigured again. Furthermore, the master device must be also reachable from any sub-network in order to exchange ROS topic data. Last problem is that ROS devices are not aware of other devices connected in the network. Whenever that information is required it must be requested to the ROS master.

Thus, the aim of this paper is to achieve the next objectives:

- Design a robot module architecture for fully isolated ROS device execution: This architecture is based on the Smart Resource implementation and every device is designed as its own master with no dependencies on external systems for ROS execution.

10.1016/j.ifacol.2017.08.1600

- Develop a real-time communication architecture to distribute the ROS topic data between Smart Resource modular devices: The development of a ROS MultiPeer Architecture (RMPA) provides enhancing mechanisms such as automated peer discovery and proxy functionalities.
- Integrate the RMPA in a real modular robot application to analyze its execution performance within a DCS with real-time constraints: The benefits and withdrawals of the RMPA integration are compared to a classic ROS solution, and the RMPA communication performance is detailed.

According to the proposed goals, the article is organized as follows: First, Section 2 introduces some related work about developing ROS distributed systems. Next, Section 3 presents the concept of Smart Resources in order to provide the execution base to develop isolated ROS-compatible modules for robotics. The communication architecture to establish the data flow between modules and its characteristics are detailed in Section 4. Some experiments and results to validate the architecture proposal are analyzed in Section 5. Finally, Section 6 addresses the conclusions and introduces the future work.

2. RELATED WORK

As any computer requires of an operative system, robot platforms also need a software architecture that offer main configuration and execution routines (Liu et al. (2000)). Furthermore, in a DCS communication mechanisms are critical (Dafang (2010)). In these systems the control data flow has to be supplied in a reliable way by ensuring the message reception and a proper data rate.

According to this, the Robot Operating System (ROS) (Quigley et al. (2009)) has been proved to be a highly spread software architecture for robot both research and commercial platforms (Cousins (2011)). ROS is an open code solution which provides libraries for executing most common robot tasks. The ROS execution is based on code blocks called modules which can be localized or distributed in the network. These modules are connected trough a communication system based on publication-subscription (Cousins et al. (2010)). It can be found some works like (Rossi (2014)) and (Yakovlev et al. (2015)) that presents a DCS implementation based on the ROS platform.

Within a DCS, connected ROS devices turns into distributed control service providers Remy and Blake (2011). Devices that are designed to be configured and interacted trough the ROS mechanisms are defined as ROS-Ready devices Araújo et al. (2014). Therefore, ROS-Ready devices ofers a high-level abstraction for accessing to distributed control services. ROS-ready devices provide a set of advantages to the robot designed and robot user. These advantages include the possibility to make compatible different ROS nodes, the reuse of different developments made by different research groups and companies, and the ability to simulate different devices, robots and scenarios. In Chen et al. (2013) an example of robot on-board sensors integration as ROS-Ready devices is addressed. ASTRO (Saraydaryan et al. (2014)) provides a service architecture oriented to big and complex robot scenarios. In some works ROS is accessed by means of a method to translate ROS messages to devices functionalities. An en example PhaROS (Estefó et al. (2014)) offers an architecture that adapts a robot programming language to be used in ROS to program dynamically a robot. I ROSbridge (Crick et al. (2011)) a bridge to view ROS based systems from non-ROS users, usually web services and Internet, is provided.

Although there is a wide range of robot-oriented communication systems (Elkady and Sobh (2012)), ROS introduces its own message system in order to interact with the ROS-Ready devices distributed in the network. Usually, systems use bridges to adapt the messages to suit the ROS communication requirements. A bridge solution is introduced in (Kang et al. (2012)) which implements and "ad hoc" solution using TCP/IP sockets. Another example is RosBridge (Crick et al. (2011)) which provides a web sockets based communication. Due to the publish-subscriber topology implemented by the ROS communication system some works implements a Data Distribution Service (DDS) layer in order to provide real time capabilities. A clear example is the work ANIMO framework introduced in (Rodríguez et al. (2014)). In Pardo-Castellote (2003) is proposed a solution for abstracting the real time communication system by providing a classic ROS communication interface.

These are some ROS integration examples that can be applied in automation and robotic developments. As previously described, ROS services can be accessed through a real time communication system that wraps the ROS interface. Along this works an communication architecture that provides Smart Resource integration as ROS-Ready devices. These devices will be characterized as robot modules integrated into a DCS network in order to support robot execution.

3. SMART RESOURCES FOR MODULAR ROBOTICS

When a device offers its functionality as services, it is necessary that the system hides all aspects related with the technology beneath. For example, in order to provide a sensing service there is no need to know the type of sensors or the micro-controller used to process the high-level information. As service providers these devices cannot be managed as nodes in a distributed system, but they can be characterized as distributed resources. Since these resouces provide high-level services that can be easily accesed and configure they are characterized as Smart Resources (Munera et al. (2015)). In a DCS system the services provided by the Smart Resources will be oriented to offer control tasks as sensing, acting or controlling.

Therefore, the main features and advantages of implementing a Smart Resources based robot architecture are addressed along this section. Furthermore, the developement of a ROS-based Smart Resources is also introduced. As a result Smart Resource service can be easily integrated into ROS-based platforms.

3.1 Smart Resource for robotics

In order to introduce main features of the robot oriented Smart Resoures it is detailed how the proposed solution can suit the needs of robot applications. The overview of Smart Resource implementation is showed in Fig. 1.

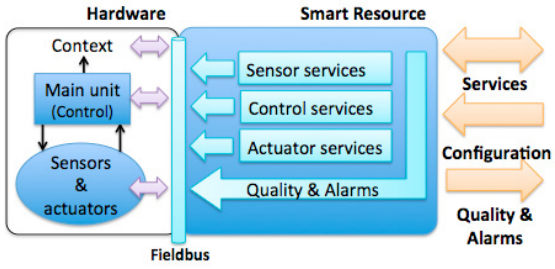

Fig. 1. Scheme of a robot-oriented Smart Resource design

The services required by the robot are the same as a in any other control system. Furthermore, modular robotics characterized as DCS, requires of the service distribution. In order to fit these requirements Smart Resource provides distributed services designed as high-level oriented control tasks. Smart Resources can be provided by an heterogeneous set of devices which can include both, physical and simulated ones.

Distributed services provided by Smart Resources are accessed through a network interface. This interface relies on a communication middleware whose main features are: network peer detection, active peer lists management, and a publish-subscriber communication (Eugster et al. (2003)). As any publish-subscriber implementation, communication is driven by topics which can be checked or updated by the network peers. Therefore, every Smart Resource requires of at least three main different kinds of topics. These three topics are distinguished according to its supplied functionality : the configuration and task execution parameters, the service input/output information, and the quality of service measures.

As stated before, Smart Resources offer the capability of adapting to the system which is configured to work within some quality bounds. When a service is requested, it must be configured in order to fulfill the need of the client. Most of these requirements are set in terms of temporal and spatial requirements, information reliability and operation performance.

A certain configuration with specific quality restrictions is defined as a System Profile. System Profiles are pre-programmed in the Smart Resource. A quality requirement is translated into the equivalent (more similar) System Profile. Because of some of these qualities cannot be provided, Smart Resource must try to adapt its execution to suit the requested configuration. Alarms notifies the client every time a quality is not satisfied, being managed as an adaptation event. If minimum requirements can not be satisfied, Smart Resource will execute the most similar performance according to the existing restrictions by offering a best-effort profile. In that case, system alarms make the client aware of a non-adaptable event that should be managed as an undesirable or unexpected situation. These non-adaptable events must not be common and are interpreted as a bad design of the Smart Resource or a non-realistic quality requirements set by the client.

Although the service quality adaptation mechanism offers an optimum management of Smart Resource capabilities, the context configuration turns out to be a critical step for increasing the system efficiency. Robots can be designed to perform in a wide range of environments and contexts. More specifically, the mobile robots design usually faces different contexts and dynamic environments and situations. According to this the performance of the required services should be adapted to the context in which the robot is developing its tasks. For this reason Smart Resources can be configured to manage different kind of information according to the needs of the robot, and also modify the quality requirements by changing the System profile.

3.2 ROS integration

Although the Smart Resources have been originally developed in order to work with their own execution middleware, they have been modified in order to integrate ROS systems, just as is detailed in Fig. 2. For this purpose, every Smart Resource must execute a ROS Core instance in order to execute all the service related tasks as ROS nodes. In the same way, the internal network has been replaced by the ROS publish/subscriber communication infrastructure. Consequently the service, configuration and quality topics are be associated to ROS topics. As a result, this proposal allows the Smart Resources to provide abstract services that relay on ROS execution nodes.

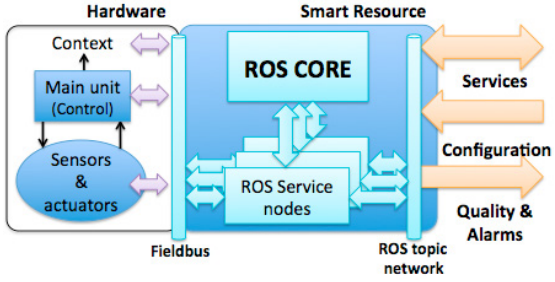

Fig. 2. ROS based Smart Resource design

4. DISTRIBUTED CONTROL ARCHITECTURE: ROS MULTI-PEER ARCHITECTURE

Once the Smart Resources topology has been introduced, the design of a distributed control architecture is introduced along this section. The aim of this architecture is to offer stand-alone robot modules based on Smart Resources that can be accessed through a distributed real-time communication system. Consequently, all the elements required for service provision are detailed next.

4.1 Smart Resource Level

As introduced before, the main function of Smart Resources for robotics is to provide control task services. Smart Resources are designed to work in collaboration with other devices (whichever are implemented as Smart Resources or not) that requires its services. Nevertheless, each Smart Resources is designed as a ROS subsystem by executing its own ROS Core. It means the topic subscription its restricted to the device local network. In order to allow other devices to access to its services the local ROS topic information must be distributed among the robot control network.

4.2 Multi-peer Topology

For distributing the information across the network it is required to establish a connection between all the devices in the control system. Every device connected to the control network will be characterized as a network peer. The connexion between peers are established by forming a connection mandala just as depicted in Fig. 3.

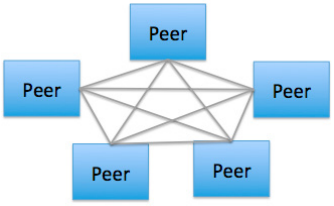

Fig. 3. Multi-peer Topology

Peers in the network are differentiated by a Global Unique Identifier (GUID) defined as a 64 bits number established by combining its IP address (unsigned 32 bits) and the connection port (16 bits). The GUID is not only used as an unique identification but also to establish an order between peers. This order is used to organize the connection sequence as detailed following.

4.3 Peer Discovery Mechanisms

In order to form the connexion mandala previously described it has been developed an automated peer discovery mechanism. As a peer is connected to the network it has to announce its presence to the system in order to be known. For that purpose a multicast diffusion channel is established by choosing an IP address and a port in order to listen to connections. Therefore, new peers will notify its GUID by means of this diffusion channel. Every time that a peer receive a new GUID it reply with their own GUID Consequently peers in the network are aware of the GUIDs of the other ones. When there is no new peer GUID, diffusion channel stays inactive. Nevertheless, each five seconds a heartbeat will check that network is still alive.

4.4 Communication Level

Once all the peer GUIDs in the network are known it must be established as TCP/IP communication between them. In order to start the communication each peer needs a server thread to accept other extern connections. Given a certain peer with an specific GUID, it will establish the connection with those ones with a higher GUI. On the reverse, that peer will be waiting for lower GUID peers to request the connection. This connection sequence is detailed in Fig. 4.

4.5 Interfacing ROS node communication

As mentioned before, ROS offers some problems related to the strong dependency to the master device which is executing the ROS Core for data distribution. Therefore module devices has been develop as stand-alone by its won ROS core, that means its own local ROS communication

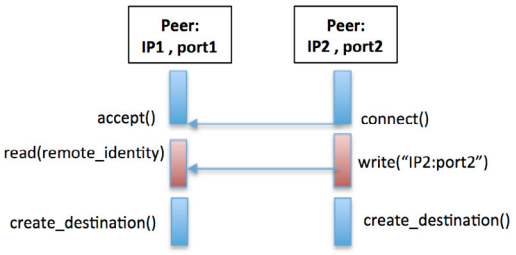

Fig. 4. Peer connection

network. So, the main objective is to migrate from a distributed system where a ROS Master manages the connection (just as depicted in Fig.5) to a set of ROS stand-alone subsystem in which each one executes its own ROS Core (as showed in Fig. 6). In the stand-alone systems establish its own local network which has to be bridged in order to distribute the information among the network peers. For that purpose the previously introduced peer detection and communication mechanisms are applied. As a result by integrating the ROS Multi-peer Architecture increases the reliability of the system as the device communications are not dependent of a master. In the classic distribution approach approach(Fig.5) any failure of the main device running the ROS core will disable the communication between the rest of the system devices. In the proposed solution (Fig. 6) failures will only affect to the device itself. In that case the criticality of the failure will be determined by the relevance of the services that this device is supplying to the system.

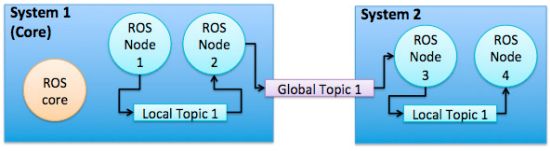

Fig. 5. Classic ROS distribution

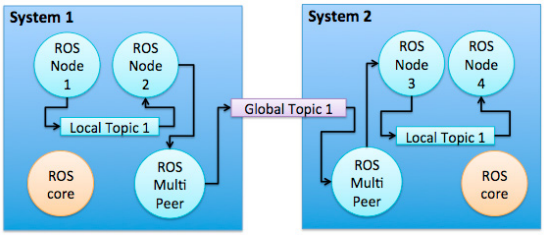

Fig. 6. ROS Multi-peer distribution

5. EXPERIMENTS AND RESULTS

Along this section a set of experiments and results are presented in order to evaluate the advantages of the proposed architecture. For that purpose it is designed am experimental setup that involves some different devices that are connected to the network for providing different services within the control chain. The whole experimental setup as is detailed in Fig. 7 includes three BeagleBone Black(BBB) and regular PC. The first BBB board is connected to an USB Joystick to read the movement orders and publish them into the 'Command Information' topic. The second BBB access this information and computes a

velocity which is also published as the 'Velocity Order'. The last board, connected to a Turtlebot robot uses the velocity commands to characterize the robot displacement. While this, the PC monitors all the topic information in the network to analyze the data flow and check if the real-time constraint are fulfilled.

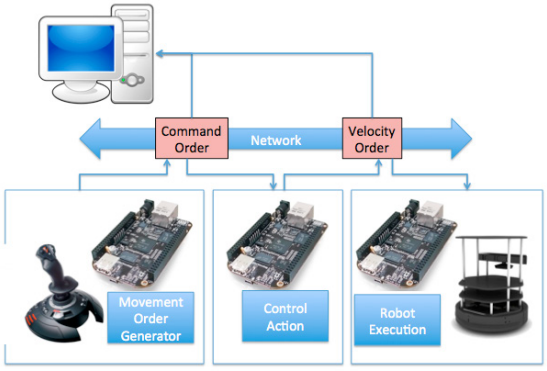

Fig. 7. Test Setup

According to the proposed setup two type of experiments has been carried out. The first set of tests makes use of the classic ROS data distribution. For that purpose, the PC is configured as the master device by executing the ROS Core. The BBB boards export this core from a well known port number from the PC IP address in order to distribute the ROS topics. In the second experiment, each device is designed as a stand-alone Smart Resource which runs its own ROS Core and exchanges ROS topics through a RMPA communication. During both experiments the control data rate is configured at 10Hz. Along the execution it will be recorded the data flow sequence for analyzinf the supply rate and the control computation time. The computation time is expressed as the difference between the reception time of the command (CMD) order and the velocity (VEL) order. Since the computational cost of the RMPA implementation does not depend on the number of peers or topics in the network it can be bounded in time and considered during the design step.

As can be observed in Fig. 9 and Fig. 8, the reception rate of both, the CMD and the VEL topics are around to 0.1s just as expected. According to the control computation delay, it is noticed how the time is significant lower in case of implementing the RMPA. As a main withdrawal, in this case the dispersion of the rates is wider than in the case of the ROS classic distribution.

A numeric comparison between both experiments can be reviewed on Table 1. As previously announced, the mean rates are around the 10Hz, but in the case of the control computation delay the mean value is reduced to almost the half than the original ROS distribution time. This effect is produced due to a more efficient execution of the ROS nodes because of running a local ROS core (supported by the Smart Resource design). Table 1 also include the dispersion values as has been graphically displayed on previous figures.

To provide a more detailed characterization of the RMPA performance a more exhaustive set of tests has been designed. These tests analyze the transmission latency according to different transmission rates and message

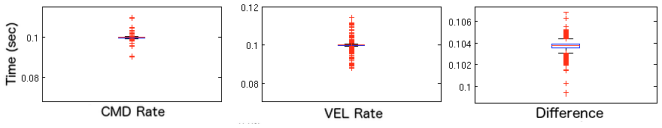

Fig. 8. ROS distribution and computation rates

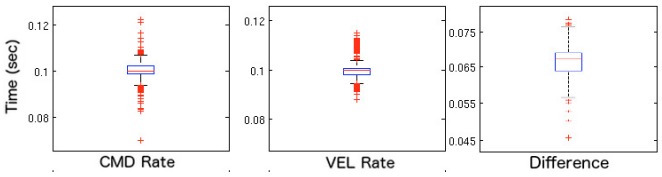

Fig. 9. RMPA distribution and computation rates

Table 1. RMPA vs ROS distribution

	Avg Rate (sec)		Std. Deviation (s)	
	RMPA	ROS	RMPA	ROS
Command Order	0.0999	0.0999	5.059e-4	2.724e-4
Velocity Order	0.0999	0.0999	5.006e-4	3.2e-4
Control Computation Delay	0.0613	0.1038	4.91e-4	1.2e-4

payload sizes. In the first set of tests, transmission rate is increased from 10Hz to 50Hz in batches of 100 messages for every evaluation. As can be observed in Fig. 10, the distribution of the measured transmission latency is grouped around 350us until 40Hz, becoming more irregular whenever the rate is increased over the 40Hz. In the second set of tests, the payload size of the sent messages is increased from 1KB to 1MB in batches of 100 messages. Figure 11 shows how the latency is affected by the payload size. As can be observed in the graph, the relation between both parameters is almost linear. According to these results, RMPA is established as an useful architecture for enhancing real time distributed communications on ROS systems.

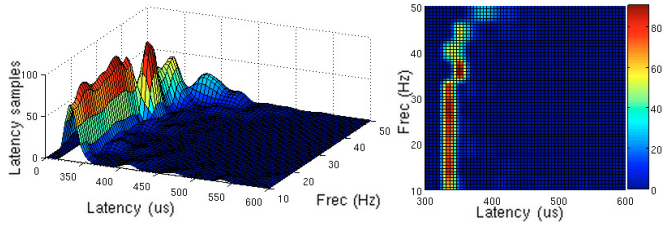

Fig. 10. Latency distrubtion according to transmission rate

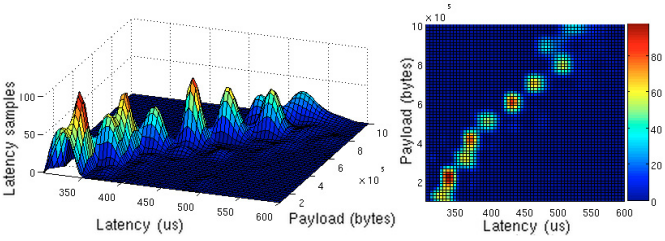

Fig. 11. Latency distrubtion according to payload size

6. CONCLUSIONS

Along this work it has been fully detailed an architecture for fast and simple development of real-time modules for robotics. This proposal offers control services that can be easily accessed by being characterized as ROS compatible stand-alone devices. As a conclusion it must be remarked that the initial goals have been successfully fulfilled.

Thanks to the development of a ROS Multi-Peer Architecture, the dependence between the master device (which executes the ROS Core) and the rest of devices in the network is avoided by setting each device as its own master. The designed modules are characterized as stand-alone devices which execution is based on a Smart Resources topology integrated with ROS. As a result the reliability of the whole system is increased since there is no execution dependency between devices. The RMPA also offers a new layer of communication which allows to manage real-time data flow which offers high communication capabilities between devices. It also provides some communication enhancements, just as the automated peer (device) discovery among the network, even when dealing between different sub-network.

Finally, this architecture has been tested by performing several tests in which some robot module exchange data within a DCS context. These results have been also compared with the performance results obtained when dealing with a classic ROS data distribution. As the result of the comparison, even that the desired rates have been fulfilled in both cases, the local execution of ROS nodes is improved when dealing with a stand-alone ROS system which executes its own ROS Core instead of exporting it from a master node. Next, a full set of tests has been developed in order to characterize the RMPA performance according to the desired transmission rate and the message payload size. These results validate the proposed architecture as a suitable solution for ROS distribution.

As a future work, it should be analyzed how the quality information and configuration capabilities of the Smart Resources (as execution base for robot modules) could be improved thanks to the RMPA capabilities. Also, it must be studied how the inclusion of metadata and persistent information into the system can lead to improvement of the global system performance.

ACKNOWLEDGEMENTS

Work supported by the Spanish Science and Innovation Ministry MICINN: CICYT project M2C2: Codiseo de sistemas de control con criticidad mixta basado en misiones TIN2014-56158-C4-4-P and PAID (Universitat Politècnica de València): UPV-PAID-FPI-2013.

REFERENCES

Araújo, A., Portugal, D., Couceiro, M.S., Sales, J., and Rocha, R.P. (2014). Development of a compact mobile robot integrated in ros middleware. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 11(3), 315–326.

Chen, H., Cheng, H., Zhang, B., Wang, J., Fuhlbrigge, T., and Liu, J. (2013). Semiautonomous industrial mobile manipulation for industrial applications. In *Cyber Technology in Automation, Control and Intelligent Systems (CYBER), 2013 IEEE 3rd Annual International Conference on*, 361–366. IEEE.

Cousins, S. (2011). Exponential growth of ros [ros topics]. *IEEE Robotics & Automation Magazine*, 1(18), 19–20.

Cousins, S., Gerkey, B., Conley, K., and Garage, W. (2010). Sharing software with ros [ros topics]. *Robotics & Automation Magazine, IEEE*, 17(2), 12–14.

Crick, C., Jay, G., Osentoski, S., Pitzer, B., and Jenkins, O.C. (2011). Rosbridge: Ros for non-ros users. In *Proceedings of the 15th International Symposium on Robotics Research*.

Dafang, H.F.L.Q.C. (2010). Research on network structure of dcs. *Process Automation Instrumentation*, 1, 004.

Elkady, A. and Sobh, T. (2012). Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012.

Estefó, P., Campusano, M., Fabresse, L., Fabry, J., Laval, J., and Bouraqad, N. (2014). Towards live programming in ros with pharos and lrp. *arXiv preprint arXiv:1412.4629*.

Eugster, P.T., Felber, P.A., Guerraoui, R., and Kermarrec, A.M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2), 114–131.

Kang, J.S., Yu, D.U., and Park, H.S. (2012). A robot software bridge for interconnecting opros with ros. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2012 9th International Conference on*, 296–297. IEEE.

Liu, F., Narayanan, A., and Bai, Q. (2000). Real-time systems.

Munera, E., Alcobendas, M.M., Poza-Lujan, J.L., Yagüe, J.L.P., Simo-Ten, J., and Noguera, J.F.B. (2015). Smart resource integration for robot navigation on a control kerned middleware based system. *International Journal of Imaging and Robotics*, 15(4), 117–129.

Pardo-Castellote, G. (2003). Omg data-distribution service: Architectural overview. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, 200–206. IEEE.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A.Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5.

Remy, S.L. and Blake, M.B. (2011). Distributed service-oriented robotics. *Internet Computing, IEEE*, 15(2), 70–74.

Rodríguez, Y., Alejo, C., Alejo, I., and Jiménez, A.V. (2014). Animo, framework to simplify the real-time distributed communication. In *UBICITEC*, 16–26. Citeseer.

Rossi, M. (2014). *ROS package for distributed control of networks of dynamical systems*. Ph.D. thesis, Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje.

Saraydaryan, J., Jumel, F., and Guenard, A. (2014). Astro: Architecture of services toward robotic objects. *International Journal of Computer Science Issues (IJCSI)*, 11(4), 1.

Yakovlev, K., Khithov, V., Loginov, M., and Petrov, A. (2015). Distributed control and navigation system for quadrotor uavs in gps-denied environments. In *Intelligent Systems' 2014*, 49–56. Springer.