Rosbridge: ROS for Non-ROS Users

Christopher Crick, Graylin Jay, Sarah Osentoski, Benjamin Pitzer and Odest Chadwicke Jenkins

Abstract We present *rosbridge*, a middleware abstraction layer which provides robotics technology with a standard, minimalist applications development framework accessible to applications programmers who are not themselves roboticists. Rosbridge provides a simple, socket-based programmatic access to robot interfaces and algorithms provided (for now) by ROS, the open-source "Robot Operating System", the current state-of-the-art in robot middleware. In particular, it facilitates the use of web technologies such as Javascript for the purpose of broadening the use and usefulness of robotic technology. We demonstrate potential applications in the interface design, education, human-robot interaction and remote laboratory environments.

1 Introduction

At present, we are at the cusp of a revolution in robotics. For most of the field's history, scientific progress has been hindered by the fact that to have a robot meant investing a great deal in its mechanical engineering and low-level control systems. The result being that every researcher had a different system with different

C. Crick (✉)
Oklahoma State University, Stillwater, OK, USA
e-mail: chriscrick@cs.okstate.edu

G. Jay
Red Hat, Inc., Brisbane, Australia
e-mail: tjay@redhat.com

S. Osentoski
Mayfield Robotics, Redwood City, CA, USA
e-mail: sarah@mayfieldrobotics.com

B. Pitzer
Google, Mountain View, CA, USA
e-mail: pitzer@google.com

O.C. Jenkins
University of Michigan, Ann Arbor, MI, USA
e-mail: ocj@umich.edu

H.I. Christensen and O. Khatib (eds.), *Robotics Research*,
Springer Tracts in Advanced Robotics 100, DOI 10.1007/978-3-319-29363-9_28

capabilities. Furthermore, robots were extremely expensive, both in terms of money and researchers' time. Only very well-funded laboratories could have a robot, and the scope of the robot's activity was constrained by the resources, research focus and imagination of the scientists and engineers that created it.

The emergence of widely-available common robot architectures promises to mitigate the "silo effect" that has heretofore lessened the impact and wider application of research contributions within robotics. Furthermore, developments in robot middleware have begun to create the software engineering infrastructure vital to fostering interoperability and code reuse, a necessary prerequisite to the use of robots on a large scale.

However, the current state of robot middleware is such that users and developers must make a heavy ontological commitment to a particular environment and philosophy in order to use it to its full effect. Furthermore, middleware designers have (perhaps by necessity) assumed that users of their systems would be roboticists themselves, well-versed in the low-level systems programming and complex control and decision algorithms which have always been a part of robotics research. We developed rosbridge to expose these systems to the much wider world of general applications developers, with the hope of unleashing for the first time a "web-scale" revolution in robot availability and accessibility.

2 Background

Several robot middleware system have been proposed to enable code sharing among roboticists. These middleware systems include Player/Stage [8], the Carnegie Mellon Navigation Toolkit (CARMEN) [24], Microsoft Robotics Studio [13], YARP [17], Lightweight Communications and Marshalling (LCM) [12], and ROS [20], as well as other systems [14]. These middleware systems provide common interfaces that allow code sharing and reuse. While middleware systems differ in their design and features, they typically provide a communication mechanism, an API for preferred languages, and a mechanism for sharing code through libraries or drivers. Middleware systems typically require developers to code within the middleware framework, and often within a specified build environment.

At their heart, many of these middleware packages provide a messaging and marshalling protocol between processes running on multiple machines connected in some fashion to robotic hardware. The framework permits, say, a stereo camera to deliver images to a stereo image processor, which in turn can send a depth map to an object recognition routine, which then routes coordinates to an inverse-kinematics driver, which sends motor commands to processes delivering voltages to individual servos. In a complex robot architecture, the number of independent processes and the information that interconnects them quickly becomes massive. Even so, deep down, the system is merely serializing and routing messages, and rosbridge takes advantage of this fact. By way of analogy, web applications have developed huge and complex backends that span continents and perform

breathtaking feats of traffic analysis, shaping, routing, data acquisition and conglomeration, but still communicate with browsers and each other over the HTTP protocol. Likewise, robots and their controlling middleware can grow arbitrarily complex on the back end, but with rosbridge they can communicate with an application layer over a single socket and a plain-text protocol.

3 ROS

Rosbridge is designed to work initially within the paradigm established by the ROS middleware system currently maintained by theOpen Source Robotics Foundation. ROS uses a peer-to-peer networking topology; systems running ROS often consist of a number of processes called *nodes*, possibly on different machines, that perform the system's computation. Nodes communicate with each other by passing messages. Under ROS, messages are data structures made up of typed fields. Messages may be made up of standard primitive data types, as well as arrays of primitives. Messages can include arbitrarily nested structures and arrays.

Nodes can use two types of communication to send messages within the ROS framework. The first is synchronous and is called a *service*. Services are much like function calls in traditional programming languages. Services are defined by a string name and a pair of messages: a request and a response. The response returns an object which may be arbitrarily complex, ranging from a simple boolean indicating success or failure to a large point cloud data structure. Only one node can provide a service of a specific name.

The second type of communication is asynchronous and is called a *topic*. Topics are streams of objects that are published by a node. Other nodes, "listeners", may subscribe by registering a handler function that is called whenever a new topic object becomes available. Unlike services, listener nodes are unable to use their subscription to the topic to communicate to the publisher. Multiple nodes may concurrently publish and/or subscribe to the same topic and a single node may publish and/or subscribe to multiple topics.

Unlike many other robot middleware systems, ROS is more than a set of libraries that provide only a communication mechanism and protocol. Instead, nodes are developed within a build system provided by ROS. The intent is that a system running ROS should be comprised of many independent modules. The build system is built on top of CMake [16], which performs modular builds of both nodes and the messages passed between them.

Furthermore, ROS has assimilated a number of tools, algorithms and systems which can serve as a basis for complex robot control. Thus a full suite of ROS packages provides vision processing algorithms [3], 3D point cloud interpretation [21] and simultaneous localization and mapping (SLAM) [10], among many others. This represents the largest effort to date to foster a robotics community that supports code-sharing and building on the prior work of others. This alone serves as reason for applying the rosbridge architecture to ROS initially.

4 Rosbridge

Rosbridge provides an additional level of abstraction on top of ROS, as depicted in Fig. 1. Rosbridge treats all of ROS as a "back end". This shields application developers from needing intimate knowledge of low-level control interfaces, middleware build systems and sophisticated robotic sensing and control algorithms. At a bare minimum they must understand the build and transportation mechanisms of the middleware package. Rosbridge layers a simple socket serialization protocol over all of this complexity, on top of which application developers of all levels of experience can create applications.

ROS abstracts individual robot capabilities, allowing robots to be controlled through messages. It also provides facilities for starting and stopping the individual ROS nodes providing these capabilities. Rosbridge encapsulates these two aspects of ROS, presenting to the user a unified view of a robot and its environment. The Rosbridge protocol allows access to underlying ROS messages and services as serialized JSON objects, and in addition provides control over ROS node execution and environment parameters (Fig. 2).

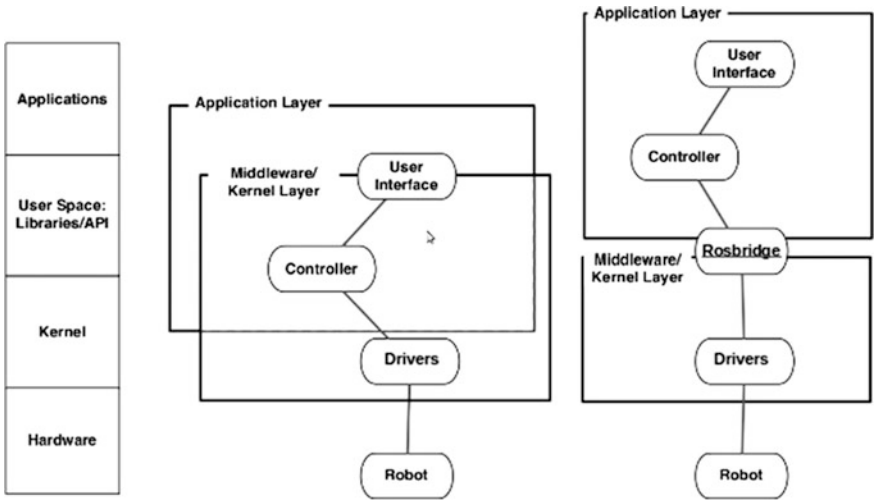

Fig. 1 Recreating traditional abstraction layers in robotics with rosbridge. As depicted at *left*, software development depends on well-established layers of abstraction. Developers and engineers working at each layer possess very different skill sets, but the enterprise succeeds due to well-defined abstractions and interfaces. At present, robotics must deal with all of these layers at once, limited by both their own skills and by the unwieldiness inherent in poorly-abstracted systems (*center*). At *right*, rosbridge attempts to establish a more clear abstraction boundary to address this problem

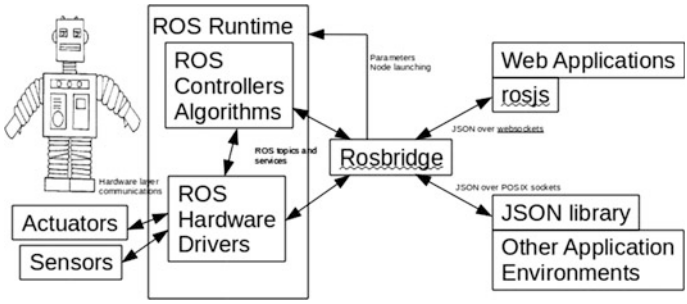

Fig. 2 Rosbridge serializes all applicable ROS topics and services over a single socket interface

Rosbridge allows simple message handling over both HTML5 websockets and standard POSIX IP sockets. For example, a simple Python client which handles data being published on a ROS topic called "/sensorPacket" can be written, simply, as

```
host_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_sock.connect((host_address, host_port))
host_sock.send('raw\r\n\r\n')
host_sock.send('\x00{"receiver":"/rosbridge/subscribe","msg":["/sensorPacket",0,]}\xff')
while True:
    incoming = source_socket.recv(1024)
    #handle sensorPacket data
```

This paradigm can be exploited in any language that supports IP sockets, which is to say, all of them. Thus rosbridge enables robot application development in a user's language of choice.

5 ROSJS

Computing paradigms have developed over the years, from batch systems to timeshared mainframes to standalone desktops to client-server architectures to ubiquitous web-based applications. Current technology allows transparent administration, redundant storage, and instantaneous deployment of software running on wildly heterogenous platforms, from smartphones to multicore desktops. This relatively new and extremely ecosystem has spawned a population of users who understand basic web technologies such as HTML and Javascript [7]. Familiarity with basic web technologies extends beyond expert application developers to users who would not necessarily call themselves programmers, but who nevertheless use the web for all manner of creation and communication and are familiar with the basic technologies. One of the goals of rosbridge is to broaden robotics to this vast

untapped population of writers, artists, students, and designers. Javascript has become the default language of the web and as such is one of the most popular languages in the world. We hope to leverage a small part of that popularity to open robotics to an entirely new audience and to make working with robotics easier for those who are already familiar.

Because this is one of rosbridge's primary goals, we have provided a large and full-featured rosbridge library in Javascript, known as rosjs. rosjs is designed to integrate ROS with the web as unobtrusively and universally as possible. Its only advanced dependency is on the HTML5 [19] technology of websockets. Currently browsers such as Safari, Opera, and Chrome fully support them, as does the nightly build of Firefox. Universality has been one of the key factors in the success of the web, and accordingly rosjs is implemented as a simple Javascript library, completely agnostic with respect to preferred development frameworks. Rosbridge is built using serialized JSON objects, which are themselves basic Javascript object syntax.

rosjs is now a large library supporting many complex features for visualization and interaction with sophisticated ROS-based manipulation and navigation algorithms. However, it can be used for extremely simple code. The following demonstrates how little Javascript code is required to send navigation commands to a robot.

```
<html><head>
<script>type="text/javascript" src="ros.js"</script>
...
var ros = new connection("ws://10.100.0.100:9090")
...
ros.publish('/cmd_vel','geometry_msgs/Twist',
            '{"linear:{"x":'+x+',"y":0,"z":0},"angular:{"x":0,"y":0,"z":'+z+'}}');
...
```

JSON is simple enough that the serialization can be done by hand, as in the above example. However, many JSON libraries exist to make the construction easier and less error-prone.

rosjs was designed to meet the needs of developers with web programming experience. There are multiple advantages to the ability to develop robot applications in the browser. Web browsers are familiar and widely-used interfaces, even by non-techical users. Allowing users to access robots through the internet may provide insights into new applications for robotics, as well be used as a tool to recruit potential scientists to the field. Javascript allows for rapid and flexible user interface and visualization development. Applications developed within a web browser are also portable across platforms, and updates and new functionality can be easily provided.

6 Rosbridge in Remote Laboratories

While middleware systems allow for code sharing and reuse, many researchers are limited by the overhead (and sometimes pure impossibility) of reproducing results on similar platforms. Large platforms like mobile manipulators are expensive and difficult to obtain for researchers at smaller institutions or companies. It is rare for researchers to have access to common platforms, let alone shared data, especially in fields focused on active learning or those requiring user studies. Additionally, the great difficulty in reproducing experimental results has hindered the robotics field for many years. It is often difficult to assess which proposed approaches per- form best. In fields where online learning and user demonstrations are required, researchers do not perform research on common platforms, let alone on shared data. A remote lab where users can compare results and share experimental data will help provide a more scientific basis for comparison.

A remote robotic laboratory would allow researchers to run experiments and compare against results produced on a common platform. We developed rosbridge and its supporting rosjs libraries in part to support the development of experimental infrastructure for the creation of remote robotic laboratories.

Figure 3 depicts a remote lab interface developed with rosjs to support research into learning from demonstration. Users can access a PR2 robot to demonstrate pick-and-place tasks, specifically setting a table. In addition, they can observe the robot's actions through a variety of sensors and camera streams, all provided through the rosbridge framework. During each session data is logged and stored in a publicly available repository. Custom controllers and learning algorithms, provided in public code repositories, can use the data and provide policies for desired tasks on the robot.

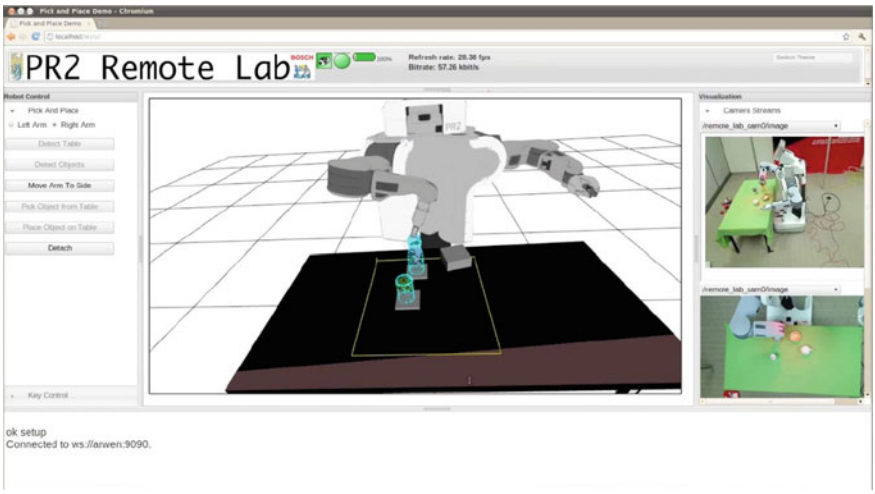

Fig. 3 A complex remote lab interface using rosjs and WebGL

There are many technical challenges to address when creating such a remote lab. The functionality provided by rosjs is instrumental to overcoming them. A web interface is required so that users can work with the robot remotely. The user must have some way of controlling the robot, either with code or through teleoperation. Users must also be able to visualize the result of the control. Security measures are required for the safety of the robot.

7 Rosbridge in Human-Robot Interaction

One of the strengths of rosbridge (and its Javascript application-layer library rosjs) is its support for quickly and easily creating remote user interfaces. Much of the teleoperation work in robotics has traditionally been aimed at tasks where robots operate in environments that are hazardous to human users, such as robotic surgery [18], search and rescue [5], and outer space [1]. In these applications, users are typically experts who have devoted a significant amount of training time to the difficult task of controlling the robot and interacting with its interfaces. Our goal with rosbridge is to allow application developers to create interfaces that are intuitive even for novice users.

Furthermore, even expert user interface designers are not necessarily experts in ROS or robotics generally. The expertise needed for developing rewarding and intuitive interactions over a simple Javascript web interface, however, is widespread and generally available.

Rosbridge has the potential to increase the number of people using, interacting with and programming robots. A recent trend in machine learning has examined the use of truly large data sets for learning rather than attempting to generalize from a small amount of data. Researchers in data mining and machine translation have able to take advantage of Google's index of billions of crowdsourced documents and trillions of words to show that simple learning algorithms that focus upon recognizing specific features outperform more conceptually sophisticated ones [11]. We conjecture that similar successes would be observed if large amounts of data could be collected for learning with robots. Human-robot interaction studies, to date, more often number in the dozens of subjects [2]. Opening up robots to the vast number of users on the world wide web provides the opportunity to gain a large number demonstrations from many different users.

The robotics community has made a few forays into human robot interaction over the internet. Goldberg et al. placed a robot in a garden and allowed users to view and interact with the robot over the web. Users were able to plant seeds, water, and monitor the garden [9]. Taylor and Trevelyan created a remote lab in which users perform tasks involving brightly colored blocks [23]. Schulz et al. examined the use of web interfaces to remotely operate mobile robots in public places [22]. This worked focused on letting remote users interact with humans within the robots' environment and did not examine the effect of the visualizations in a learning task. Burgard and Schulz have explored handling delay in remote

operation/teleoperation of mobile robots using predictive simulation for visualization [4].

In previous work, we have used rosbridge to leverage precisely this large network effect [6]. HRI research into the character of interfaces and visualizations which lead to successful human teaching of robot behavior was able to draw on a large pool of participants and develop 276 use cases and eighty thousand points of data.

8 Rosbridge in Education

The simplicity and system independence of rosbridge make it a very powerful tool for programming and robotics education. The ease of hooking into a robot system using simple sockets and text-based JSON messages means that students have a very gentle learning curve. In addition, programming languages and environments that have been expressly designed for educational purposes can easily be extended to communicate with rosbridge.

Figure 4 shows robotics development in the Scratch environment [15], a visual programming system designed for children to learn and understand programming concepts. A very simple extension to Scratch allows students interact with robots programmatically. The system has been used by middle-school students, who were able to program robots to perform basic closed loop behaviors such as line following and bump exploration, without ever being aware of the underlying complexities of ROS itself.

We are currently developing higher-level courses to take advantage of rosbridge, as well. At the college level, robotics classes have traditionally spent a great deal of time just "hacking on the machine", dealing with and learning about the massive infrastructure necessary to get robots to do useful things. Rosbridge short-circuits this process, allowing students to spend more time learning about higher-level

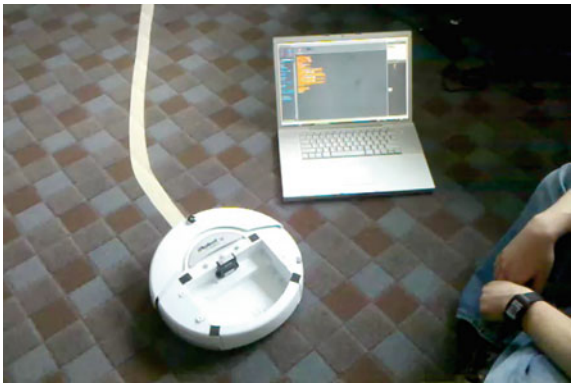

Fig. 4 Robotic control using the Scratch educational programming environment

control and perception and less time wondering how to extract images from a camera stream or compile behaviors in an abstruse and poorly-documented programming environment.

9 Rosbridge Without ROS

In addition to extending ROS, rosbridge can be extended to provide similar functionality for other middleware systems. The messaging protocol at the core of most robot middleware can be translated into JSON objects just as ROS messages can, and passed through the same sockets using the same interface. Our goal is to not only extend ROS to but to also advocate that this additional level of abstraction may be beneficial to other middleware systems.

We are currently developing rosbridge support for the LCM system [12]. This will create a common interface for robots running ROS and LCM to send messages to each other, and for application developers to write software that can support robots running either system.

10 Conclusions and Future Work

In this paper, we described rosbridge, a high-level middleware abstraction layer that exposes robot functionality to developers as a simple interaction over a socket. In addition, we have developed rosjs, a Javascript library on top of rosbridge, that supports extensive interaction and visualization of higher-level ROS constructs. We believe that web-based interaction with robots provides the largest potential pool of new users and developers, and so expanding and enhancing rosjs has consumed the largest share of our development resources. However, the rosjs framework also serves as a model for the development of other libraries for other languages. Interaction with rosbridge can be as simple as desired—no more than sending text strings over sockets—but of course advanced functionality should be developed to support whatever tasks a user wishes. Rosbridge enables that development in purely agnostic fashion.

We plan to develop rosbridge as much as possible into a simple nexus for robotics technology to meet general application development. Already we have begun work on an LCM component for rosbridge, with the hope of supporting general application development for robots using either form of middleware. In addition, we hope to support device manufacturers who need a simple, low-overhead means of interfacing with other computer systems. An embedded robot system might not be able to accommodate the computational demands of a ROS system onboard, but if it can send and receive plain-text messages over a POSIX socket, it can easily interface with ROS over rosbridge. This would greatly expand the technological resources available to the robot.

Rosbridge enables ROS to communicate with the web, applications developers to communicate with robots, and robotics researchers to communicate with each other. All of these are necessary for robots to succeed in the world.

References

1. H. Aldridge, W. Bluethmann, R. Ambrose, M. Diftler, Control architecture for the robonaut space humanoid, in *Proceedings of the First IEEE/RAS Conference on Humanoid Robotics* (2000)
2. C.L. Bethel, R.R. Murphy, Use of large sample sizes and multiple evaluation methods in human-robot interaction experimentation, in *AAAI Spring 2009 Symposium: Experiment Design for Real-World Systems* (2009)
3. G.R. Bradski, V. Pisarevsky, Intel's computer vision library: applications in calibration, stereo segmentation, tracking, gesture, face and object recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2000)
4. W. Burgard, D. Schulz, *Beyond Webcams: An Introduction to Online Robots, Chap, Robust Visualization for Online Control of Mobile Robots* (MIT Press, 2002), pp. 241–258
5. J.L. Casper,R. Robin, A.J. Murphy, Workflow study on human-robot interaction in user, in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation* (2002)
6. C. Crick, S. Osentoski, G. Jay, O.C. Jenkins, Human and robot perception in large-scale learning from demonstration, in *Proceedings of the 6th ACM/IEEE Conference on Human—Robot Interaction* (2011)
7. ECMA-262: ECMAScript language specification, 5th edn. (2009), URL http://www.ecmainternational.org/publications/standards/Ecma-262.htm
8. B. Gerkey, R.T. Vaughan, A. Howard, The player/stage project: tools for multi-robot and distributed sensor systems, in *Proceedings of the 11th International Conference on Advanced Robotics* (2003), pp. 317–323
9. K. Goldberg, H. Dreyfus, A. Goldman, O. Grau, M. Gržinić, B. Hannaford, M. Idinopulos, M. Jay, E. Kac, M. Kusahara, (eds.), *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet* (MIT Press, Cambridge, 2000)
10. G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with raoblackwellized particle filters. IEEE Trans. Robot. **23**(1), 34–46 (2007)
11. A. Halevy, P. Norvig, F. Pereira, The unreasonable effectiveness of data. IEEE Intell. Syst. 8–12 (2009)
12. A. Huang, E. Olson, D. Moore, LCM: lightweight communications and marshalling, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2010)
13. J. Jackson, Microsoft Robotics Studio: A Technical Introduction. IEEE Robot. Autom. Mag. 82–87 (2007)
14. J. Kramer, M. Scheutz, Development environments for autonomous mobile robots: a survey. Auton. Robots 101–132 (2007)
15. J. Maloney, M. Resnick, N. Rusk, B. Silverman, E. Eastmond, The scratch programming language and environment. Trans. Comput. Educ. **10**(4), 1–15 (2010)
16. K. Martin, B. Hoffman, *Mastering CMake: A Cross-platform Build System* (Kitware Inc, 2008)
17. G. Metta, P. Fitzpatrick, L. Natale, YARP: yet another robot platform. Int. J. Adv. Robot. Syst. 43–48 (2006)
18. A.M. Okamura, Methods for haptic feedback in teleoperated robot-assisted surgery. Ind. Robot. **31**(6), 499–508 (2004)
19. M. Pilgrim, *HTML5: Up and Running* (O'Reilly Media, 2010)

20. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng, Ros: an open-source robot operating system, in *Proceedings of the Open-Source Software Workshop of the International Conference on Robotics and Automation* (2009)
21. R.B. Rusu, S. Cousins, 3D is here: point cloud library (PCL), in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)* (2011)
22. D. Schulz, W. Burgard, D. Fox, S. Thrun, A.B. Cremers, Web interfaces for mobile robots in public places. IEEE Robot. Autom. Mag. **7**, 48–56 (2000)
23. K. Taylor, J. Trevelyan, A telerobot on the world wide web, in *National Conference of the Australian Robot Association* (1995)
24. K. Wyobek, E. Berger, H.V. der Loos, K. Salisbury, Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) toolkit, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2003), pp. 2436–2441