2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

PR2 Remote Lab: An environment for remote development and experimentation.

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

Abstract— **In this paper, we describe a remote lab system that allows remote groups to access a shared PR2. This lab will enable a larger and more diverse group of researchers to participate directly in state-of-the-art robotics research and will improve the reproducibility and comparability of robotics experiments. We identify a set of requirements that apply to all web-based remote laboratories and focus on solutions to these requirements. Specifically, we present solutions to interface, control and design difficulties in the client and server-side software when implementing a remote laboratory architecture. The combination of shared physical hardware and shared middleware software allows for experiments that build upon and compare against results on the same platform and in the same environment for common tasks. We describe how researchers can interact with the PR2 and its environment remotely through a web interface, as well as develop similar interfaces to visualize and run experiments remotely.**

I. Introduction

As illustrated by the high response rate to the PR2 Beta program, the demand for high quality research platforms far outweighs the supply. The scarcity of such platforms is a limiting factor in the productivity of the robotics research community and the development of the robotics market. Additionally, these platforms often come at a high cost making them difficult to obtainable for smaller universities and research groups. To address these problems, we propose a remote research lab in which users can develop, test, and compare robot controllers.

Previous attempts to create remote lab systems and online robots [1], [2], [3], [4], [5], [6] focused on simple experiments and online learning. Additionally, these systems did not build upon shared robot middleware systems. This is partly due to the fact that many of these labs were not intended for shared research experimentation. One consequence of this is that researchers could not necessarily easily extend previous efforts to build their own labs.

Shared, opensource robot middleware infrastructure makes it easier for researchers to compare to and build upon each other's work. These systems generally allow separate components to interact and share data – a feature necessary for a flexible, extensible robot software system. Trevelyan et al. created middleware for their remote lab in order to have a flexible enough system to support an effective remote access laboratory [7]. Additionally, the higher abstraction level of middleware lessens many of the problems of robot control

B. Pitzer and S. Osentoski are with Robert Bosch LLC at the Research and Technology Center North America, Palo Alto, CA 94304, USA. G. Jay, C. Crick, and O. C. Jenkins are with the Dept. of Computer Science, Brown University, Providence, RI 02912

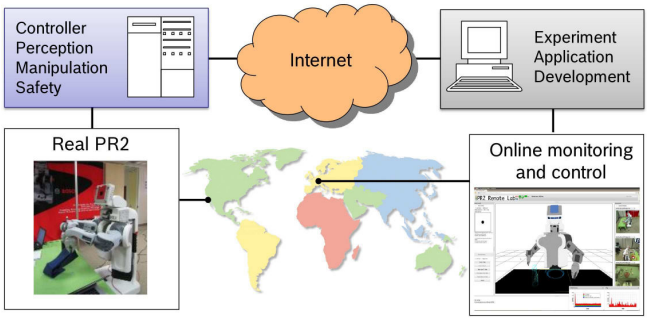


Fig. 1. In the PR2 remote laboratory clients connect over the internet, run experiments, and develop robotic applications on a shared PR2 platform.

in a remote environment. For example, by providing IK-based end-effector control of the robot's hands, middleware eliminates the need for a remote client to be capable of sending several joint velocity commands with extremely low latency. Instead, the client need only send simple messages regarding a single point in space. Similarly, robust object detection allows only object position and type be sent to a remote client for visualization rather than a bandwidth intensive point cloud or image.

We use ROS, Willow Garage's robotic middleware system [8], in part due to its popularity: currently there are at least 116 ROS repositories and 3089 ROS packages[1]. Not only can we can leverage this body of work to create a remote lab system, but the researchers whose work we build upon can also take the infrastructure we have used to build this remote lab and use it in their own research.

Previous attempts to create remote lab systems had problems with connectivity, software requirements, and giving users the ability to do closed loop control. New web technologies such as HTML5 and Javascript solve many of the difficulties of user interfaces without requiring users to download additional software or plugins other than a supported web browser.

In this paper, we present a system that allows remote groups access to a shared PR2. A *PR2 Remote Laboratory* requires robust remote control, external sensing, remote safety systems, remote error recovery, static and dynamic environment modeling, visualization, and environment configuration. By solving these problems we allow a larger and more diverse group of researchers to directly participate in state of the art robotics research. It will also give us the opportunity to carry out joint experiments and therefore foster

[1]http://www.ros.org/wiki/rosdoc_rosorg

collaboration between research labs. Additionally this lab can be used for research challenges such as the annual Learning from Demonstration Challenge. Researchers will, for the first time, truly be able to compare their results and evaluate the strengths and weaknesses of proposed algorithms.

II. Related Work

Researchers have examined using the internet to allow humans to remotely interact with and monitor robots. Remote laboratories offer the possibility to improve distance education, timetabling issues, accessibility and the cost of running laboratories. Li et al. [9] also cite the scarcity of equipment and supervisors as a reason for pursuing remote laboratories. A notable examples of a successful remote laboratory installation is the Telerobot of the University of Western Australia (UWA) [10], [7]. This robot has been online for over ten years and it has been used by many students as remote laboratory to test kinematic models and control algorithms allowing them to stack blocks. Other examples of similar remote laboratory systems can be found in [5], [11], [6], [4]. Previous remote laboratories for online learning focused on simple experiments such as manipulating simple objects [10], [7], visual servoing [12], basic physics [6], and signal processing [11].

Other online robots have mainly been built for entertainment purposes. 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 [1]. Also, Taylor and Trevelyan provided public internet access to their UWA robot, which users could use to stack brightly colored blocks [10].

Recently, work in the machine learning community examined using crowdsourcing approaches to train robots. For example, Chernova et al. [13] examined using a multi-player video game where users collaborate to provide user demonstrations. Crick et al. [14] allowed a large number of users to demonstrate policies on an actual robot within a maze environment using one of two interfaces. The remote lab described in this paper could be used for such experiments, as well as experiments that do not require learning or human user studies.

Connectivity is probably the most significant issue in all remote lab systems. If a client has a slow connection, then they may not be able to receive or send all data. The amount of data sent to a client must be adapted to the available bandwith by either discarding some of the data or selecting low bandwidth data sources. The main data source for the listed remote laboratory systems is one or multiple video streams. The datarate for a single VGA (640x480), 30 fps, MJPEG compressed video stream is 2.4 Mbps. Thus, on slow connections streaming video will be at low framerates and delayed by several seconds due to buffering [7].

Another important issue for web operated remote laboratory systems arises from the choice of the client software. For example, Dalton [15] implemented a Java applet client that provided asynchronous communication between the UWA robot [7] and the remote user. This requires the user to download and install a Java runtime engine and several plugins. Dalton also reported that different combinations of browser, operating system, graphics hardware and processor would result in different appearance of the Java applet.

III. PR2 Remote Laboratory

A. Hardware Setup

The robot platform used in the remote laboratory described in this paper is the PR2 personal robot [16], a two-armed robot with an omnidirectional base. Equipped with two 7-degrees-of-freedom compliant arms, the PR2 is designed for compliant interaction with the environment. The compliance is a important safety feature that includes both the safety of humans and objects that may be present in the robot's environment as well as protecting the robot itself when being used by a remote user. The PR2 has an extensive sensor suite allowing remote users to perform a variety of mobile manipulation tasks.

The remote lab environment is equipped with additional sensors. Four cameras observe the robot's work-space and allow the remote user to see the PR2 and objects in its environment. In addition to the 2D cameras, a depth camera is used to acquire a 3D view of the remote lab. A PhaseSpace optical motion capture system is used to obtain ground truth pose information for the robot and objects. The PhaseSpace system tracks active LED markers and provides highly accurate, real time motion data which is necessary to update the remote lab's state.

B. Software Infrastructure and Design

The software is structured on both the client and server side. An overview is presented in Fig. 2. The client side consists of a web browser based user interface implemented in HTML and Javascript. This interface is described in more detail in Sec. V.

On the server side software is roughly divided into three main layers: hardware interfaces, ROS middleware, and web services. The hardware interface layer comprises a number of software modules concerned with receiving and time stamping all sensor data. This layer also contains interfaces to the robot's actuators.

ROS, the robotic middleware controlling the PR2, provides a structured communications layer above the host operating system. It is organized into a so-called computation graph which is the peer-to-peer network of *nodes* that are processing data together. The nodes communicate asynchronously with each other by passing messages called *topics* based on publish / subscribe semantics. A synchronous communication model is realized by *services*. In this case, a client makes a request and waits for a reply before continuing. Due to its popularity and wide adoption, ROS has successfully developed from yet another robotic middleware package into an extensive community that enables exchange of software and knowledge. As a result a large collection of ROS software packages is publicly available in open source repositories. We can leverage this community by creating remote laboratory experiments on a different level than has been previously

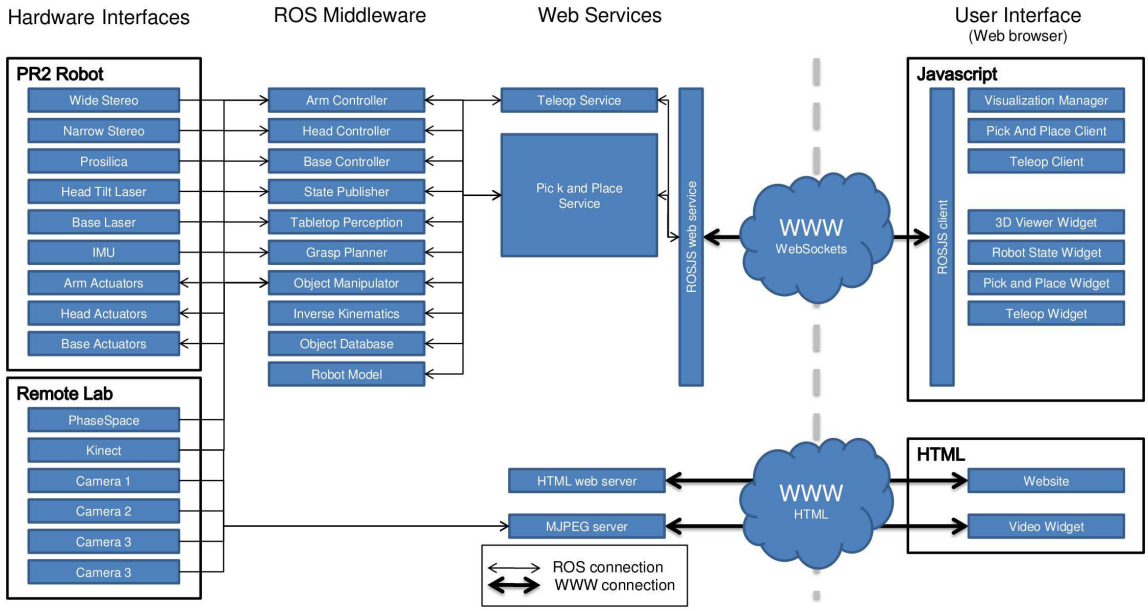


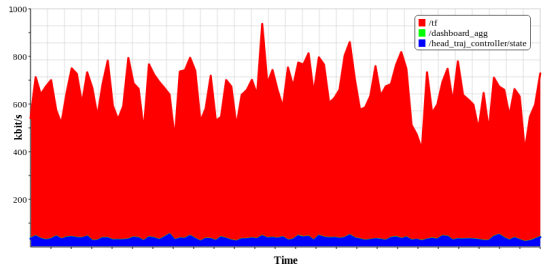
Fig. 2. Flowchart of the remote lab's software system. The software is roughly divided into four main layers: Hardware interfaces, ROS middleware, web services, and user interface.

attempted. For example, packages for manipulation[2] and navigation[3] are already available and can be used as reference implementations for experiments seeking to improve single components.

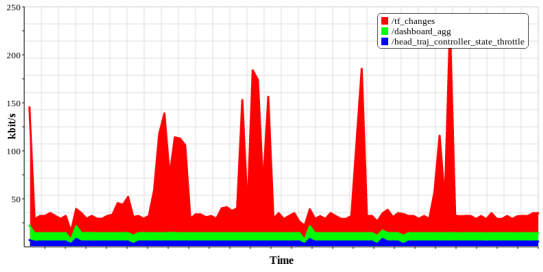
Additonally, many of the packages availble in ROS abstract robot control or perception in a high-level way. For example, navigation packages process high-fidelity range-finder data into positional information. This provides opportunities to lessen latency and bandwidth requirements while providing control of the robot and perception of its environment.

IV. Rosbridge and rosjs

ROS addresses the connectivity and synchronization requirements by providing a flexible communication layer. However, it was designed to function in a homogenous, tightly-integrated environment with extremely fast hardware connections, not for slow and unreliable connections over the web. We use Rosbridge [17], which exposes the capabilities of ROS, such as a robot's data streams and controllers, through POSIX and web sockets accessible anywhere over the internet, as well as providing security mechanisms and run-time tools for interacting with and maintaining a robot system. We also use rosjs, [18] a Javascript binding for ROS which provides a powerful array of visualization and control tools that can be used directly from a web browser. The combination of Rosbridge and rosjs hides much of the low level complexity of ROS from application developers through abstractions and is designed to meet the needs of developers with web programming experience. There are multiple advantages to the ability to develop robot applications in the web browser. Web browsers are familiar interfaces that are widely used, especially by nontechnical users. 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 easily be provided.



(a) Datarate for the three largest topics sent at production rate.



(b) Datarate for the three largest topics after optimizing. Here, the /tf topic is transmitted only as partial state update of changing state variables and throttled to 10 Hz. Spikes in the data correspond to joint motion events where multiples states are updated in a short period of time.

Fig. 3. Comparison of datarates before and after optimization

Rosbridge includes several helpful features to assist robot application developers and researchers in creating robot

[2]http://www.ros.org/wiki/pr2_object_manipulation
[3]http://www.ros.org/wiki/navigation

controllers and interfaces that are executed over the network. The first feature is that Rosbridge exposes ROS services and topics. Rosbridge provides several additional services, such as dynamic access to the types and objects associated with topics and services, as well as runtime control of the ROS environment, such as launching nodes and setting parameters.

The second feature that Rosbridge provides is access control. When allowing remote users access to a robot security is a fundamental concern. rosjs provides two types of security: protected services/topics and key authorization. Protected topics and services are necessary when there are critical services that the client should not interfere with – for example, those that enforce human or robot safety. Key authorization allows the developer to limit access to a robotic web application to specific users.

The third feature is data logging. A remote lab is most useful when there is a mechanism to record data from experiments. Rosbridge provides a mechanism for developers and users to save data. ROS developers may log data using ROS on the server side, but the logging mechanism also supports the logging of client-side information about the user's experience. Additionally, rosjs provides a data logging widgets[4] to record data for experiments so that remote users can decide on the fly what data they wish to record. This functionality is likely to be useful for fields where user studies are necessary such as Robot Learning from Demonstration (LfD) and Human Robot Interaction (HRI).

As the Rosbridge client can be any internet host, the exact network characteristics of the connection are not possible to predict in advance. However, tests of a representative selection of hosts can establish likely network characteristics. Limited bandwidth and variable end-to-end delays are a major concern for web connections. We first measure the datarate (number of bits per second) being transported through Rosbridge. Note that this does not include video streams since a separate channel is used for this data. Fig. 3(a) shows the datarates of the three largest topics sent to the client at the rate of production. The average datrate is about 580 kbit/s dominated by the /tf topic (530 kbit/s). For the PR2, the /tf topic updates the robot's state at 100 Hz. Since most of the state variables change infrequently, a simple optimization of this data channel consists of transmitting only partial state updates of changing state variables. Fig. 3(b) presents the datarates after applying this optimization. The result is a significantly reduced datarate (70 kbit/s) with spikes at times when the robot's arms move. A second optimization performed in this experiment is an active throttling of topics.

The end-to-end delay is estimated by measuring the round trip time for a small topic in ROS send from the client to a node running on the robot. This is a more accurate measure than network round trip times (ping) since it includes the overhead involved in rosjs Javascript, Rosbridge, and ROS for processing the messages. For a LAN connection this round trip is about 30 ms while a US East-West connection increases this tenfold.

[4]http://www.ros.org/wiki/topic_logger

V. User Interface

Since users will not be local, it is necessary to have a user interface that can be conveniently accessed remotely. A web-based interface is a natural fit given this constraint. Tools such as HTML5 and Javascript allow developers to create sophisticated interfaces, and rosjs was designed to take full advantage of this technology, enabling applications developed in the web browser to communicate with a robot running ROS and Rosbridge. Thus, the web interface has access to the entire ROS middleware ecosystem. rosjs is a large collection of visualization tools and widgets implemented in Javascript and communicating with ROS topics and services over Rosbridge. In this section, we describe two approaches to for visualizing the robot's state and sensor data in a browser, developed using rosjs. We also illustrate various user interaction techniques to operate the robot. A picture of the developed interface is depicted in Fig. 4.

A. Video Visualization

A natural means of interacting with and checking on the progress of the robot is through video. MJPEGs, or motion JPEGs, are a file format in which each frame of a video stream is separately compressed as a JPEG image. We created an mjpeg streaming server that subscribes to requested image topics in ROS and publishes those topics as MJPEG streams via HTTP to a web browser. While rosjs is capable of streaming video, the web browser is optimized to efficiently download images in binary format. This additional communication channel is used for increased performance benefits. Since many of the previous labs reported problems with sending a large number of high resolution video streams over the internet due to latency, we deemed it important to handle video streams as efficiently as possible. Users can also provide the desired quality and size to accommodate different connection speeds and interface designs.

B. 3D Web Visualization

One of the most useful tools in robot development is a 3D visualization environment. One benefit of many 3D interfaces is that the messages required to create them are relatively low bandwidth, especially when compared to live video streams. The remote lab interface described in this paper provides this functionality. Whether robot models, poses, maps, or custom visualization markers, the visualization interface can display views of various types of robot data. The underlying technology for visualizing 3D data on the web is WebGL, a 3D graphics API implemented in a web browser without the use of plug-ins. Similar to OpenGL, WebGL provides a programmatic interface for 3D graphics using the OpenGL ES 2.0 standard. Application developers may use the WebGL library of their choice for visualization. Our 3D visualizer provides an interface to WebGL based on world objects and other high level classes. It also provides a scene graph, which is an object-oriented representation of the 3D world. Such a representation is especially suitable for robotic development,

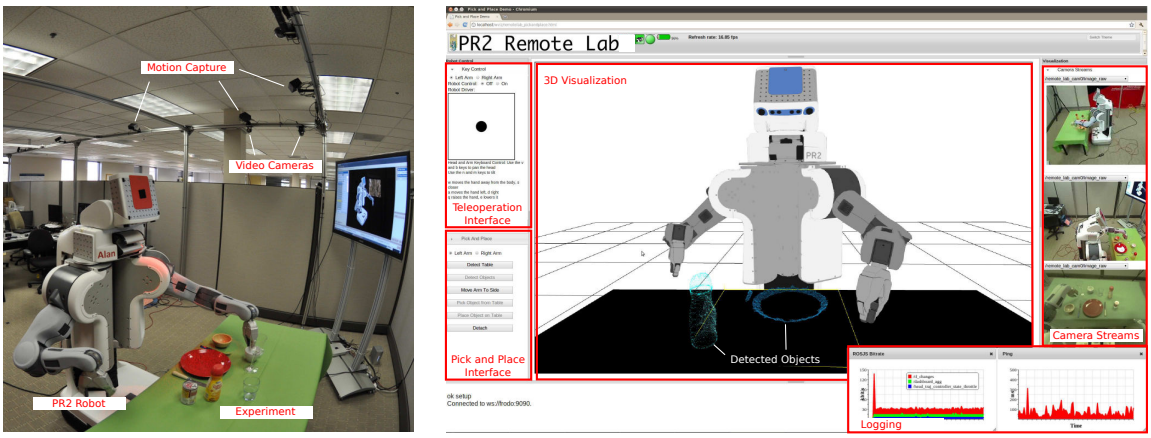


Fig. 4. In the PR2 remote laboratory the workspace is observed by multiple cameras and a motion capturing system (left). The remote users see and control the robot through a web-based interface (right). 3D visualization based on WebGL is used to show the robot's state and sensor data. A user can interact with the PR2 by using one of the predefined interfaces or by writing small Javascript programs in order to send control commands.

since data is represented in various interdependent frames. It also simplifies the extension to new data types since scene nodes can be implemented for any data type and inserted into the graph, as long as they adhere to a common interface.

C. User Interaction

In order for the remote lab to be useful, researchers must be able to interact with the robot on a variety of different levels. Different research projects require different interfaces. Additionally, if frequent tasks can be automated this reduces the burden upon the remote operator, allowing them to just run preprocessed commands. In this section, we describe three types of user interaction we have currently implemented for the remote lab. The first interface allows the user to manually tele-operate the robot through key commands and a joystick widget; the second interfaces allows the user to pick and place objects through a point and click interface; and the third interface lets users run javascript code directly within the browser. Users are able to select the type of interaction they want for a particular situation and can switch interaction modes without restarting the interaction.

1) Keyboard Teleoperation Interface: The keyboard teleoperation interface allows the user to use the keyboard to control the pan, tilt of the head, either arm, and using a virtual joystick controller. In order to use keyboard control for manipulation, the user selects the desired arm through the interface and turns on control. Key commands issued by the user, are translated into a new end position of the hand and sent to the JTTeleop controller[5] which calculates an inverse kinematic solution.

2) Pick and Place Interface: As a part of creating the remote lab we created a pick and place interface that allows users to select objects from the interface. We extended the pick and place demo [6] provided in ROS. After requiring the robot to detect the table and objects on the table the user can directly click on an object, select the appropriate arm, and select the option "Pick up object". The robot then performs calculations and selects the best stable grasping points for the selected object. While the robot is performing computations the interface is grayed out to provide feedback for the user.

3) Scripting Interface: For advanced users a simple scripting interface consisting of a text box and submit button can be provided. Users enter and execute Javascript code, including new components and classes, without the need of a server-side script. This provides advanced users with the ability to write complex robot controllers rather than issuing a set of "commands" as in [19]. This enables users to test new code without changing the entire interface as well include Javascript code from other webservers on the fly.

VI. Use Cases

We have designed the remote lab to be a flexible and extensible enough for many different types of users. In this section we describe three potential use cases, however the use of the remote lab is by no means limited to these situations.

A. Remote Development

The first use case one in which a remote developer wishes to test, debug or experimentally evaluate code or an algorithm on the provided system. In this use case, the remote user provides a link to their code within an svn repository when scheduling time on the remote lab. The user can then interactively monitor the results of the code or algorithms remotely debugging and performing experiments. This use case would most likely be appealing to users who do not have regular access to a PR2 and wish to develop, extend, or demonstrate their applications or algorithms on this platform. Users can provide benchmarking results that compare against other algorithms tested on the same platform on similar tasks.

As part of this project, we provided use of our remote lab for teams wishing to participate in the 2011 Learning from Demonstration Challenge, held in conjunction with the AAAI Conference and Robotics Exhibition. Teams were invited to use the remote lab to develop and test their algorithms on a PR2. Three teams of the four teams that

[5]http://www.ros.org/wiki/teleop_controllers/JTTeleopController
[6]http://www.ros.org/wiki/pr2_pick_and_place_demos

used a PR2 to participate in the challenge used the remote lab to develop and test their algorithms. Teams demonstrated skills both for low level control as well as at high level task planning. To demonstrate these skills, teams used the web interface for tasks including watching the results of experiments, helping techniques collect demonstration data and performing demonstrations for the robot.

B. Learning from Demonstration Experiments

Learning from demonstration is paradigm in which users, who are not roboticists or even programmers, demonstrate desired tasks to the robot. Machine learning algorithms are then used to create the controllers from the provided data. All that is required of the user is the ability to complete the task in a way that the robot can interpret. One of the challenges facing LfD techniques is the ability to gather enough data to construct truly robust and generalizable controllers. Researchers are often limited by the need to bring in a large number of subjects into the lab to gather data in order to test their algorithms.

Remote laboratories like the one described in this paper could be helpful for many types of LfD. Research would have the ability to recruit a larger number of diverse users resulting in large data sets that could be a shared resource for the field. Since the technology used to create the lab is open source and available to community, researchers can use similar interfaces and recruit users from the internet for their experiments.

C. Shared Autonomy

In many systems, robots and humans work as partners; shared autonomy aims to bridge the gap between full human control and full autonomy. Shared autonomy is useful when environments are hazardous to humans such as search and rescue domains and outer space or when humans are hazardous to the environment such as robotic surgery. A key for shared autonomy systems are intuitive and easily accessible human-robot interfaces. The PR2 remote lab system is a good example of such an interface. In much the same way that LfD is a natural fit, the remote laboratory provides the ability for researchers to test and develop shared autonomy techniques. We have begun using this system to test a variety of interfaces for shared autonomy for manipulation tasks.

VII. Conclusion

We presented a web-centered remote laboratory comprised of a PR2 and the necessary hardware/software infrastructure necessary to make available for public Internet use. We identified a set of requirements that apply to all web-based remote laboratories and focus on solutions to these requirements. Specifically, we presented solutions to interface, control and design difficulties in the client and server-side software when implementing a remote laboratory architecture. The combination of shared physical hardware and shared middleware software allows for experiments that build upon and compare against results on the same platform and in the same environment for common tasks. We describe how researchers can interact with the PR2 and its environment remotely through a web interface, as well as develop similar interfaces to visualize and run experiments remotely. The PR2 remote laboratory was designed for remote development as a primary use case. However, another significant use could be for educational purposes. A PR2 remote laboratory available for students could improve distance education, the accessibility, and the cost of running laboratories.

References

[1] K. Goldberg, H. Dreyfus, A. Goldman, O. Grau, M. Gržinić, B. Hannaford, M. Idinopulos, M. Jay, E. Kac, and M. Kusahara, *The robot in the garden: telerobotics and telepistemology in the age of the Internet*. MIT Press, 2000.

[2] D. Schulz, W. Burgard, D. Fox, S. Thrun, and A. B. Cremers, "Web interfaces for mobile robots in public places," *IEEE Robotics and Automation Magazine*, vol. 7, pp. 48–56, 2000.

[3] M. Fu, C. Yeo, Y. Lin, and F. Wang, "Waves: towards real time laboratory experiments in cyberspace," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, 2001, pp. 3470 –3474.

[4] J. Strandman, R. Berntzen, T. Fjeldly, T. Ytterdal, and M. Shur, "Lab-on-web: performing device characterization via internet using modern web technology," in *Proceedings of the Fourth IEEE International Caracas Conference on Devices, Circuits and Systems, 2002.*, 2002, pp. I022–1 – I022–6.

[5] M. Callaghan, J. Harkin, G. Prasad, T. McGinnity, and L. Maguire, "Integrated architecture for remote experimentation," in *IEEE International Conference on Systems, Man and Cybernetics, 2003.*, vol. 5, Oct. 2003, pp. 4822 – 4827.

[6] M. Casini, D. Prattichizzo, and A. Vicino, "The automatic control telelab: a web-based technology for distance learning," *IEEE Control System Magazine*, vol. 24, no. 3, pp. 36–44, June 2004.

[7] J. Trevelyan, "Lessons learned from 10 years experience with remote laboratories," in *International Conference on Engineering Education and Research Progress "Through Partnership"*, 2004.

[8] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[9] L. Li, F.-Y. Wang, G. Lai, and F. Wu, "Online autonomous guidance system for remote experiments in control engineering," in *IEEE International Conference on Systems, Man and Cybernetics, 2003*, vol. 3, Oct. 2003, pp. 2444 – 2449.

[10] K. Taylor and J. Trevelyan, "A telerobot on the world wide web," in *National Conference of the Australian Robot Association*, July 1995.

[11] A. Ferrero, S. Salicone, C. Bonora, and M. Parmigiani, "Remlab: a java-based remote, didactic measurement laboratory," *Instrumentation and Measurement, IEEE Transactions on*, vol. 52, no. 3, pp. 710 – 715, June 2003.

[12] R. Marín, R. Wirz, P. Sanz, and J. Fernández, "Internet-based telelaboratory: Remote experiments using the snrp distributed network architecture," in *Advances in Telerobotics*, ser. Springer Tracts in Advanced Robotics. Springer, 2007, vol. 31, pp. 429–444.

[13] S. Chernova, J. Orkin, and C. Brazeal, "Crowdsourcing HRI through online multi-player games," in *Proceedings of the AAAI 2010 Fall Symposium on Dialog with Robots*, 2010.

[14] C. Crick, S. Osentoski, G. Jay, and O. C. Jenkins, "Human and robot perception in large-scale learning from demonstration," in *6th ACM/IEEE International Conference on Human-Robot Interaction*, 2011.

[15] B. Dalton, "Techniques for web telerobotics," Ph.D. dissertation, University of Western Australia, 2001.

[16] Willow Garage, "Personal Robot 2 (PR2)." [Online]. Available: http://www.willowgarage.com

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

[18] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, and O. C. Jenkins, "Robots as web services: Reproducible experimentation and application development using rosjs," in *Proceedings of the 2011 IEEE International Conference on Robotics & Automation*, 2011.

[19] R. Marín, P. J. Sanz, and A. P. Del Pobil, "The uji online robot: An education and training experience," *Autonomous Robots*, vol. 15, pp. 283–297, Nov. 2003.