Web Applications for Robots using rosbridge

Jihoon Lee[1]

***Abstract*— This paper presents our efforts to adopt the adavantages of web-based solutions to develop robotic applications. We describe the usage of web applications in robotics and introduce *ROSProcessingjs*, our web-based robot application development environment, which enables the development of robot applications in a web browser in *Processing*.**

I. INTRODUCTION

Web-based applications allow for a user-friendly, platform agnostic and universally accessible, interactive environment. Typical shortcomings in robotics include poor accessibility, extensibility and usability, which can be well addressed through web-based solutions. This paper describes my efforts to integrate such advantages of web-based solutions into robotic systems.

We present the usage of web interfaces for robotics systems with three examples, and also introduce ROSProcessingjs – a web-based robot application development environment which provides abilities to develop robot applications in a browser using *Processing*.

II. PREVIOUS WORK AND BACKGROUND

Web applications for robotics have been discussed for years. Bungard and Schulzhave studied delay handling in remote operation of mobile robots [1]. Goldberg et al presented a way to interact with robots over the world wide web, which allowed remote users to do gardening with living plants over the web [2]. Taylor and Trevelyan [3] investigated the usability of 6-DOF robot controlled over the web. Osentoski et al provided a method to visualize and interact with complex robot platforms like the Willow Garage PR2 using *rosbridge* [4] and *rosjs* [5]

Crick et al [6] demonstrated that large scale user experiments can easily be performed through web accessible robots. The scale of user experiments in robotics is usually done with fewer people than other research areas and has often biased results to those who already are already familiar with robots. This is due to limited access to complex platforms. The web-interface developed for this experiment enabled 132 individuals to connect from various locations and to attempt to navigate an iRobot Create through a maze via web teleoperation.

These previous efforts and discussions have demonstrated that the character of web-based solution would help robotics industry in many ways. This paper describes more examples of web interface for use in robotics and a quick way to develop such interfaces using *Processing* [7].

[1]J. Lee is with the Computer Science Department, Brown University jihoon_lee@brown.edu

III. ROS, ROSBRIDGE, ROSJS, AND MJPEG SERVER

In our work, ROS(Robot Operating System) [8] is used as a back-end system, and key components like *rosbridge*, *rosjs*, and *mjpeg server*) are employed to compose a web-enabled robotic system. Figure 1 illustrates the overview of architectural design of web equipped ROS system. ROS, an open source robot middleware, administrates the low-level robot controllers, sensor processing, planning, navigation and manipulation. *rosbridge* manages communication channels between web applications and ROS. *rosjs* handles *rosbridge* connection on web application side. Additionally, *mjpeg server* can be used to efficiently stream video to the web.

A. ROS

ROS(Robot Operating System) is a sophisticated open-source robot middleware platform which has a large community of developers and users. It provides hardware abstraction, device drivers, and many useful libraries and tools to develop robot applications.

A ROS system consists of a number of processes called *nodes* which are connected in a peer-to-peer network topology. *nodes* perform the system's computation, and communicates with other *nodes* in ROS by exchanging 'messages', which may contain a single primitive or complex data structure like a PointCloud.

ROS provides two types of messaging mechanisms called *topic* and *service*. *topic* is a pub/sub based asynchronous message type. A publisher broadcasts *topic* messages to the system, while the subscribers listen to the published information on this *topic*. There may be more than one publisher or subscriber for the same *topic*. On the other hand, *service* is request and response based synchronous message type. Unlike *topic*, there is only one provider to handle the requests for each *services*.

In our work, ROS is employed as a back-end system to handle the low-level control interfaces, and sophisticated sensing and control algorithms. ROS allows the use effective existing navigation and manipulation robotic implementations.

B. rosbridge

rosbridge exposes the functionality of ROS systems to non-ROS systems. It enables communication through either HTML5 websockets [9] or standard TCP/IP sockets. This provides the non-ROS systems ROS-style communication mechanisms including topic publishing/subscribing, and service request/response as serialized JSON objects.

This architectural design expands the environment for running robot applications to any programming language that supports IP sockets.

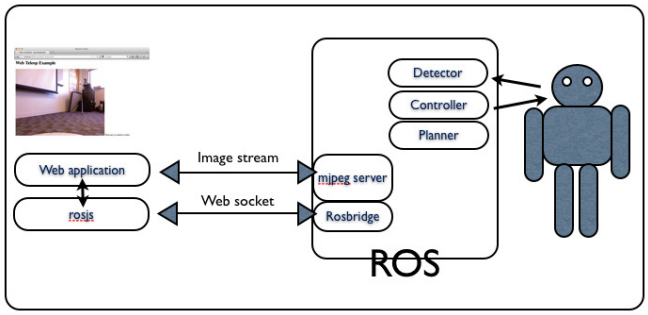

Fig. 1: Overview of system : rosbridge

C. rosjs

rosjs is a javascript [10] library that manages connections to rosbridge over standard HTML5 websockets. It provides a simple way to handle topic publishing/subscribing and services using serialized JSON objects.

The conjunction of *rosbridge* and *rosjs* can hide the system-level complexity of ROS from web developers and extend its usage. This enables the development of robot application to be much simpler, and perhaps expand the robotics field to web application developers.

D. mjpeg server

mjpeg server [11] is a ROS package that streams image topics in ROS via HTTP to a web environment. Though rosbridge also has a capability to stream images, *mjpeg server* is heavily optimized for transferring messages of this type. *mjpeg server* also allows each client to specify the quality of image and size to accomodate different client's environment like connection speed and web interface design.

IV. Examples

We have developed web applications to interact with back-end robotic system like 'PR2 Castle builder'[1] 'PR2 Commander'[2], and 'PR2 Turntable Manipulator'.

A. PR2 Castle Builder

'PR2 Castle Builder' implements castle building with toy blocks using a PR2. The user designs a castle structure with the web interface, which is then built by the PR2

Figure 2 presents the web interface used in the project. It consists of control panel to manage connections to a robot, status indicator, start/stop buttons , castle structure design tool, and video stream viewer. The web interface is developed using *ROSDojo*. *ROSDojo* is our web development framework for robotics that allows to adopt the capability of dojo [12], a powerful javascript toolkit that enables object oriented programming.

The web application works in the following order.

[1]CastleBuilder : `http://brown-robotics.org/wp/projects/hackathons/castlebuilder`

[2]Commander : `http://brown-robotics.org/wp/projects/hackathons/move-that-block-february-27-march-1`

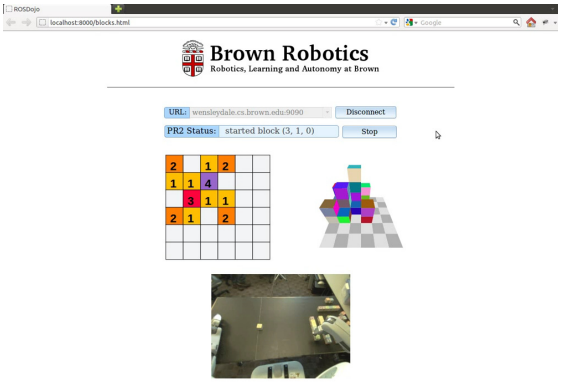

Fig. 2: Web interface for Castle builder

1) Connect to a robot: A PR2 can be connected through the control panel. The web interface connects to the robot accessed by URL. It then subscribes to the status topic, which provides the current status of the robot, and streams the video from the robot.

2) Design a castle: Designing a castle is done by increasing and decreasing digits on each cell of a 6x6 2D grid. The digits indicate the number of blocks on each cell. The designed castle model can be checked on 3D visualizer which is developed using WebGL.

3) Command: The command consists of the list of blocks' poses. When the command is sent, the PR2 starts to build the user-designed castle with toy blocks. The given block pose arrays get sorted in bottom to top, left to right, and far to near, the PR2 picks up "free" toy blocks from fixed positions, and builds the castle. It sends the "completed" message when it completes building.

We have demonstrated building a tower of 9 blocks, and also building a castle with up to 38 toy blocks.

B. PR2 Commander

The goal of "PR2 Commander" was to implement a natural language command interface for the PR2 robot. We wanted the robot to receive natural lanugage commands (such as "pick up the red block") and perform the appropriate action. To accomplish the goal, we integrated MIT's Spatial Language Understanding(SLU) [13] system with Castle Builder using *rosbridge*. SLU is a model for understanding natural language commands given to autonomous systems that performs mobile manipulation in semi-supervised environment. This integration enables the PR2 to perform castle building using hierarchical natural language commands. In the project, we accomplished picking-and-placing of toy blocks.

Figure 3 presents the web interface. Unlike "PR2 Castle Builder" which sends an array of block poses, the user either types a string command or tells it through google speech recognizer. The string commands can be anything that means pickup and drop. For example, we used "Pick up a blue/red block", "Grab a blue/red block", "Put it down", or "Drop the block". When the command is placed, SLU system interprets the command and tell the robot controller to perform the appropriate behavior.

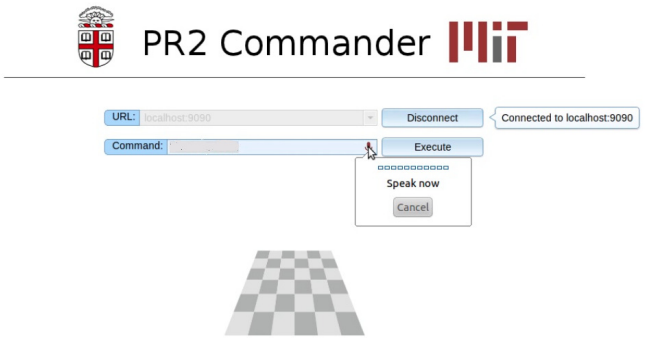

Fig. 3: Web interface for PR2 Commander

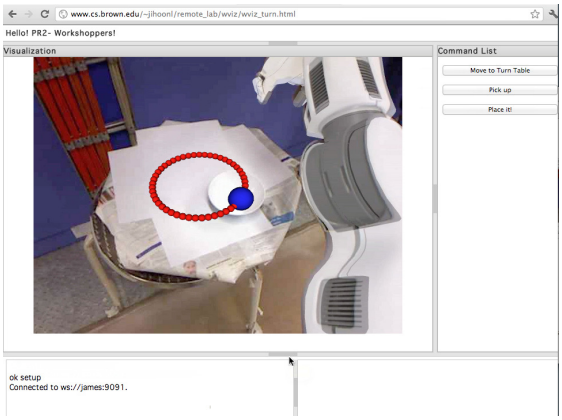

Fig. 4: Web interface for Turntable manipulation

C. PR2 Turntable Manipulator

Human-in-the-loop control is an effective model for coordinating complex tasks among robots and human operators. This control paradigm may be used to detect obscure, difficult-to-sense objects, or validating the quality of work. Figure 4 shows the human-in-the-loop web interface used for picking up an object from a rotating table. The interface consisted of a camera view overlaid with the motion model of object on the rotating table and command buttons to order moving to the turntable, picking up, and placing. The interface allowed a human operator to command a pickup at appropriate time, which performed better than letting the robot decide on a moment to pickup the object.

V. ROSProcessingjs

Processing [7] is a visual programming language for beginning programmers and designers to make data visualization, digital art, interactive animations, educational graphs, and video games. It provides many libraries for interactions and visualization that allow the development of interactive interfaces quickly and easily.

The conjunction of *rosbridge*, *rosjs*, and *Processing.js* [14] enables importing the capability of *Processing* into ROS and also allow and also allows developing a robot application in the browser written in *Processing*. This section describes the system design of *ROSProcessingjs* and usage with a teleoperation example.

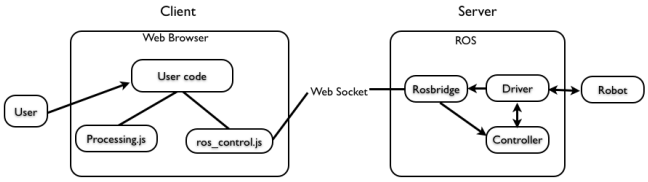

Fig. 5: ROSProcessingjs system design

A. System Overview

Figure 5 shows how a user communicates with a robot through *ROSProcessingjs* system. On client side, user's code is translated into javascript using processing.js and get connected with *rosbridge* server. The user's code may control the robot, visualize the robot's status or both. On server side, *rosbridge* listens to the client's requests and process them in ROS system.

This system, designed using *Processing.js* gives a couple of advantages. *Processing.js* provides helpful APIs for visualization and interaction, and allows mixing *Processing* with javascript. Because *Processing.js* translates the code into javascript, the functions in processing can be called from external javascript function and vice versa.

B. Example

1) Processing Code: Figure 6 presents the teleoperation code written in Processing. The code describes how to create a connection to a *rosbridge* server, stream a camera sensor image using mjpeg server, and publish a topic message to ROS.

When the program is instantiated, it connects to *rosbridge* server in line 27. connect(), which is provided by *ros_control.js*, establishes a web socket connection. *ros_control.js* also provides `publish()`, and `subscribe()`, which are wrapper functions to publish to and subscribe from a ROS topic. Then camera image streaming is being started using mjpeg server in line 30-31. The image stream can be configured by several parameters including topic name, quality, and size of image.

`move()` function in line 81 describes how */cmd_vel* topic is published to ROS. `publish()` takes three arguments topic name, topic type, and data as in a JSON object.

When the user presses or releases the key, it will invoke `keyPressed()`/`keyReleased()`, and publish the */cmd_vel* topic via move().

2) HTML Setup: Figure 7 shows the minimum requirements to program using processingjs. Importing *ros.js* and *ros_control.js* allows communication with ROS and use `connect()`, `publish()`, and `subscribe()` functions to *Processingjs* code. *processing.js* enables processing code to run in the web browser. It translates processing code into javascript. The log function is needed in order to log the status of ros connection(Connected, Closed, or Error).

As the page is loaded, the given code(*robot_control.pde*) gets instantiated and assigned to *ins* variable. Line 18 and 22 describe how to invoke the functions in given processing

```
String ip = "ws://localhost:9090";
String image_topic = "image_topic";
String image_uri = "http://localhost:8080/?topic=" +
     image_topic;
PImage img;
int imageWidth = 480;
int imageHeight = 360;

        isRunning = false;
int mainLoopVar;

PFont fontA = loadFont("courier");

float x_vel = 0.4;
float z_vel = 1;

void setup()
{
  size(500, 300);
  background(200);
  fill(10);
  textFont(fontA, 20);
  text("Move up   : W",20,20);
  text("Move down : S",20,35);
  text("Turn left : A",20,50);
  text("Turn right: D",20,65);

  connect(ip);
  isRunning = false;

  img = createImage(imageWidth,imageHeight, RGB);
  img = loadImage(image_uri);
  size(imageWidth,imageHeight);
  frameRate(50);
  loop();
}

void draw()
{
  image(img, 0, 0, imageWidth, imageHeight);
  if(isRunning) {
  // do something
  text("Move up   : W",20,20);
  text("Move down : S",20,35);
  text("Turn left : A",20,50);
  text("Turn right: D",20,65);
  }
}

void run()
{
  println("Start");
  isRunning = true;
}

void stop()
{
  println("Stop");
  isRunning = false;
}


void keyPressed()
{
  if(isRunning) {
    if(key=='w' || key=='W') {  move(x_vel, 0);}
    else if(key=='s' || key=='S') { move(-x_vel, 0);}
    else if(key=='a' || key=='A') { move(0, z_vel); }
    else if(key=='d' || key=='D') { move(0, -z_vel);}
  }
}

void keyReleased()
{
  move(0, 0);
}

void move(x,z) {
    publish('/cmd_vel','geometry_msgs/Twist','{"linear
         ":{"x":' + x + ',"y":0,"z":0}, "angular":{"x
         ":0,"y":0,"z":' + z + '}}');
}
```

Fig. 6: Web Teleoperation code in processing

```
<html>
  <head>
  <script type="text/javascript" src="ros.js"> </script
       >
  <script type="text/javascript" src="ros_control.js">
       </script>
  <script type="text/javascript" src="processing-1.0.0.
       min.js"> </script>
  <script type="text/javascript">
    var ins;
    // log function is required for ros_control.js
    function log(str) {
      console.log(str);           // error log can be
           viewed by browser error log window.
    }

    function init() {
      ins = Processing.getInstanceById("
           processing_canvas");          // get a
           processing code instance
    }

    function run() {
      ins.run();                  // call run function in
            processing code
    }

    function stop() {
      ins.stop();                 // call stop function
           in processing code
    }
  </script>
</head>
<body onload="init()">
  <canvas id="processing_canvas" data-processing-
       sources="robot_control.pde"></canvas>
    <input type="button" Value="Run" onClick="run()"/>
    <input type="button" Value="Stop" onClick="stop()"
         />
  </body>
</html>
```

Fig. 7: Basic HTML code for ROSProcessingjs.

code in javascript. This example expects *robot_control.pde* have run() and stop().

C. Brown Experiment House

This section presents our remote experiment house which allows the development a robot application in web browser and test the code directly through *rosbridge*. Figure 8[3] shows the experiment house page. It is connected to a virtual PR2 simulated on a remote server and teleoperates the virtual PR2 using the given code in the code box on the right side.

The experiment page consists of three elements: A canvas to stream a sensor image, a text area to input a processing code, and three buttons: one for initiating the code, one for running, and one for stopping. Clicking the 'init' button will crawl the code in the right-side text area, creates an instance of processing code.

This experiment environment enables testing a robot application quick and easy way, and also allows re-using the code on different robots. Because the code stays server side and communicates through *rosbridge*, the robot does not need any extra programs installed but *rosbridge*.

With this foundation framework, we built the 'Brown Remote House' prototype that allows 24/7 web-based programming of an iRobot Create at Brown University. Figure 9

[3]URL : http://www.cs.brown.edu/people/jihoonl/Brown_Remote/ros2.html

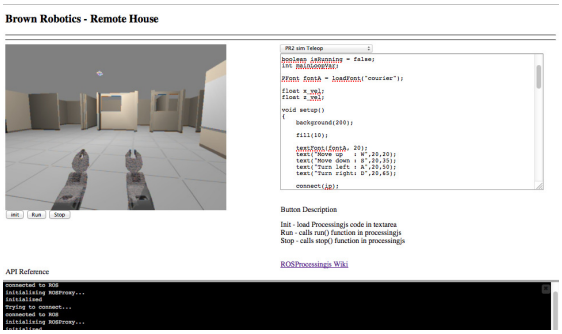

Fig. 8: Robot Simulation in ROSProcessingjs

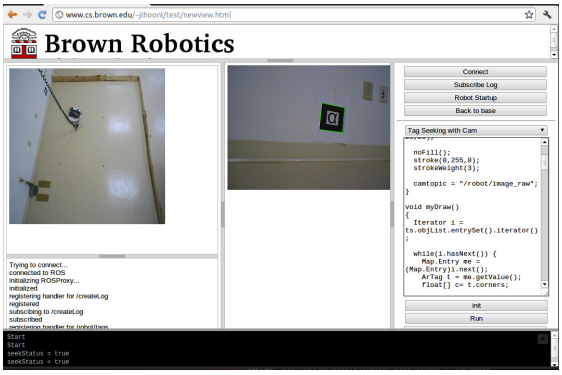

Fig. 9: Brown Remote House

presents the front-end interface of 'Brown Remote House'. The Remote House back-end is a Create running ROS that can be controlled and programmed using the *rosbridge* network interface for ROS. The user front-end is a webpage that consists of video streams from the robot and an overhead camera, and a text area for programming the robot using *Processing*.

VI. CONCLUSIONS

In the paper, we have illustrated the usecases of web application for robotics, and presented our web application development framework. These tools provide an easy method to develop robot web applications compatible with a broad array of web enabled devices.

The development of effective web applications will be increasingly important as the robotics community grows beyond the specialized researchers. Effective robot task management will need to be accomplished without a deep understanding of the system architecture from the users perspective. Web-based tools provide one solution to this problem.

REFERENCES

[1] W. Burgard and D. Schulz, "Robust visualization for web-based control of mobile robots," in *Beyond Webcams: an introduction to online robots*, K. Goldberg and R. Siegwart, Eds. MIT-Press, 2002.

[2] K. Goldberg, S. Gentner, C. Sutter, and J. Wiegley, "The mercury project: a feasibility study for internet robots," *IEEE Robotics and Automation Magazine*, vol. 7, pp. 35–39, 1999.

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

[4] C. Crick, G. T. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: Ros for non-ros users," in *International Symposium on Robotics Research (ISRR 2011)*, Flagstaff, AZ, USA, August 2011.

[5] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, and O. Jenkins, "Robots as web services: Reproducible experimentation and application development using rosjs," in *International Conference on Robotics and Automation (ICRA 2011)*, 2011.

[6] C. Crick, S. Osentoski, G. Jay, and O. C. Jenkins, "Human and robot perception in large-scale learning from demonstration," in *HRI*, 2011, pp. 339–346.

[7] B. F. Casey Reas, *Processing*, MIT Std., 2001. [Online]. Available: www.processing.org

[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] M. Pilgrim, *HTML5: Up and Running*. O'Reilly Media, 2010.

[10] *ECMA-262: ECMAScript Language Specification, 5th ed*, E.C.M.A. International Std., December 2009. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[11] B. Pitzer, *mjpeg server*, Bosch Research Std., 2011. [Online]. Available: www.ros.org/wiki/mjpeg_server

[12] *Dojo, a javascript toolkit*, The Dojo foundation Std. [Online]. Available: www.ros.org/wiki/mjpeg_server

[13] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *AAAI*, 2011.

[14] A. Salga, D. Hodgin, A. Sobiepanek, S. Downe, M. Medel, and C. Leung, "Processing.js: sketching with ¡canvas¿." in *SIGGRAPH Talks*, 2011, p. 15.