Transforming Basic Robotic Platforms into Easily Deployable and Web Remotely Controllable Robots

Yvon Autret, Jean Vareille, Philippe Le Parc

Transforming basic robotic platforms into easily deployable and Web remotely controllable robots

Yvon Autret, Jean Vareille and Philippe Le Parc
Université Européenne de Bretagne, France
Université de Brest - EA3883 LISyC
Laboratoire d'Informatique des Systèmes Complexes
20 av. Victor Le Gorgeu, BP 809, F-29285 Brest
E-mail: yvon.autret@univ-brest.fr

Abstract—**This paper describes a way to transform basic robotic platforms into Web remotely controllable robots. Our goal is to achieve robot deployment anywhere at anytime at low-cost. As soon as full or even restricted Internet access is available (WIFI or 3G), the robot can be deployed and Web-controlled. The distant user can send commands to the robot and monitor the state of the robot. For example the distant user can make the robot move and get snapshots taken by the robot.**

Keywords-Ubiquitous robot; Web control; service robotics;

I. INTRODUCTION

The use of network technologies inside robots is nowadays classical [1, 2]. A WIFI network can be used and an on-board Web server may allow control of the robot from anywhere in the world. This solution can be easily implemented. It is used in the commercial Rovio WowWee robot [3]. Unfortunately, it has some disadvantages. First, a moving robot evolves in a limited WIFI area and may get out of control. That is a major problem for outdoor robots. Second, a robot including an on-board server cannot use any WIFI router without configuration. This means that deploying a remote controlled robot is not a "plug-and-play" operation.

Current 3G coverage is now so wide that it provides almost universal Internet access. Unfortunately, 3G networks are not perfect for remote control of robots. Many 3G providers block all ports except some outgoing ports (HTTP port 80, HTTPS port 443, ...). This means that an on-board web server cannot work on a remote controlled robot. This restriction can be overcome by using HTTP tunnelling [7]. A distant server is used and all communications performed are encapsulated using the HTTP protocol. The main problem of this solution is often the lack of performance due to the overhead of communications in the distant server.

In this paper we propose to use distant servers for only one purpose: ensuring efficient robot control in a restricted Internet environment. We will adapt HTTP tunneling to remote robot control in order to guarantee correct performances and simple configuration. The remote user has no direct access to the robot and sends commands to a distant server which will transmit them to the robot (Fig. 1). On the other side, the robot can send its state to the distant server, making it available to the user (internal state of the robot, snapshots of the environment, ...).

The distant server can control several robots which can be in various locations. The only thing required is the ability of the robot to send HTTP requests to the distant server. Either WIFI or restricted 3G networks can be used. Installing a robot at anytime and anywhere in the world is possible as soon as basic Internet access is available (for example, only an outgoing port 80).

In this paper we first present a basic robotic platform. The basic robot is very simple, just moving/turning forward or backward on tracks when powered. It is used with a computer which has the ability to control the robotic platform and get an Internet access. We then show how to transform it into a communicating robot which can be easily deployed anywhere. This ubiquitous robot [4, 5, 6] is designed to be integrated in an ubiquitous environment.
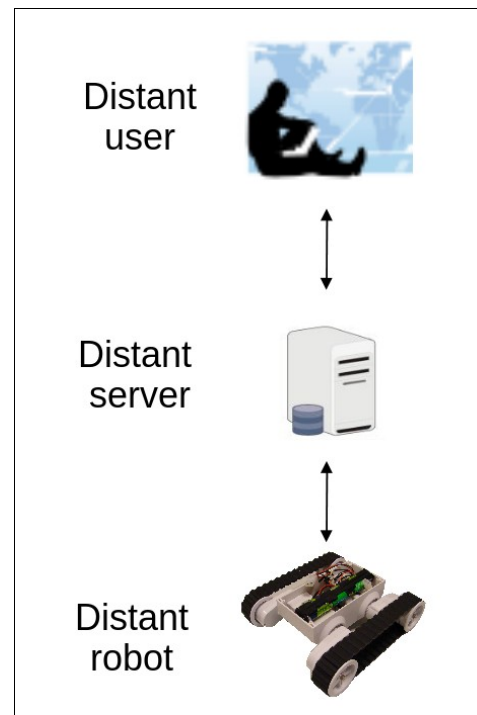

Figure 1: the proposed architecture.

II. THE BASIC ROBOTIC PLATFORM

Many commercial Web-controllable robots such as Miabot [8] or WoWee Rovio [3] are available. They can be used in an ubiquitous environment to monitor a house. The Miabot robot is rather small (about 3 inches long). The WoWee is bigger (about 14 inches long). Both are not easily expandable. That is why we start from an open robotic platform which includes only two tracks. It is a Rover 5 from RobotBase (Fig. 2). The size is that of the WoWee Rovio. When powered, it can move forward or backward and turn. It is strong enough to carry a computer and some electronic devices (up to two kilograms). The robot is controlled by the computer it carries and the computer is connected to the network to get Web-control. As the computer is a standard PC, external devices can be easily connected to upgrade the robot. The total cost (computer included) is comparable to that of a WoWee Rovio. Our robot is designed to be integrated in a low-cost ubiquitous system.

The computer we use is a PC which can be disk-less and screen-less and must run on batteries. We made tests with a Linutop computer [9] as well as with a standard mini laptop. A USB key (3G key or WIFI Key) provides Internet access. A Linux system (ISO file) is set on another USB Key and the computer boots from it in such a way that the whole system is running in Ram-Disk. The system is configured to automatically load the network configuration (WIFI WEP key, ...) from a text file on the USB key. This text file is not included into the ISO system file. This file is loaded by the Linux system at boot time (the system executes an "ifup" Linux command to load the network configuration). Thus, editing this file before starting the system is the only thing required to ensure network connection.

A Phidget 1017 electronic board (Fig. 3) including an USB port and 8 DPDT relays (Double-Pole Double-Throw) is used as an interface between the computer and the robot [10]. We use the Java language to control the relays.

The Phidget board just avoid soldering and micro-controller programming. The computer controls the DPDT relays through the USB port. Two relays (one per track) are used to make the robot presented above move or turn. Two more relays are required to reverse the current (one per track) and make the robot move or turn, forward or backward.
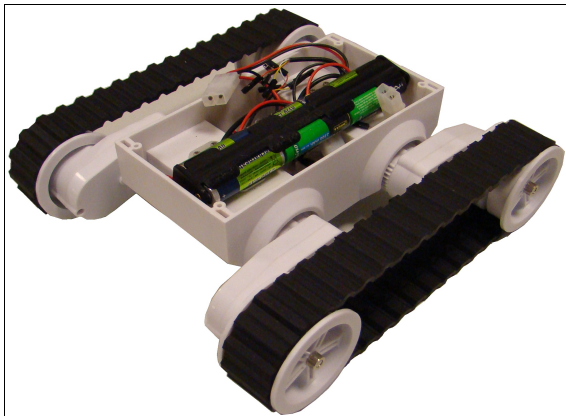

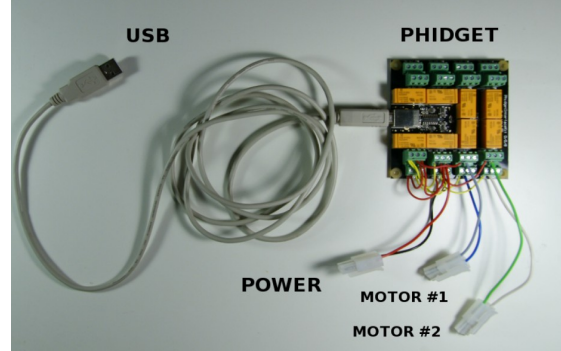Figure 2: the basic Rover 5 robot

Figure 3: Phidget wiring.

III. WEB-CONTROL OF THE ROBOT

We want our system working even if Internet access is restricted, i.e. if some ports are not available. For example, the system must work if there is only one outgoing port 80 available. That means that the robot cannot wait neither for HTTP requests nor any other request. The robot will only send HTTP requests and wait for HTTP responses. Our robot has the ability to do that because the system is connected to the Internet and a Java program inside a Unix process can be launched to send HTTP requests.

On the user side, it is the same thing. We want the distant user send only HTTP requests and wait for HTTP responses. The only thing required on the user side will be a Web browser, for example running on a PC Phone.

A. The distant server

In that system, the distant user has no direct access to the robot and we use an additional distant server. The distant user sends a robot command to the distant Web server through a Web interface. The distant server forwards the command to the robot.

1) Encapsulating robot commands into HTTP requests

To send commands to the robot, the distant user uses a Web interface which displays buttons and various fields. A robot command is nothing but a Web form. For example, the distant user clicks on the "MOVE FORWARD" button to make the robot move forward. Parameters can be used, for example how long the robot must keep moving. The Web browser will automatically encapsulate the parameters of the robot command (i.e the Web form) inside an HTTP request. The distant server processes the request as soon as it is sent.

2) Processing HTTP requests by Servlets

To process HTTP requests on the distant server, we use an Apache Web server [11]. Requests received are first forwarded to a second Web server which is a Tomcat Web server [12]. Servlets [13] running on the Tomcat server process the requests and extract the parameters (Fig. 4). The Apache server is only used to provide a standard access on port 80.

3) Forwarding requests to the robot

As we want the system working with very limited Internet access, may be only outgoing port 80, it is not possible to install any kind of server on the robot. To make the system work, the robot first sends an HTTP request to the

distant server. The request is processed by a Servlet and let pending. As soon as the distant user sends a robot command, the HTTP request processing resumes and an HTTP response is sent to the robot. The robot command is inserted in the HTTP response as a serialized object. If the distant user does not send a robot command within a few seconds, a timeout is triggered on the robot system. The robot stops waiting for the current HTTP response. A new HTTP request is sent to the distant server to wait for a robot command.

The system is working in four steps as shown above (Fig. 4) and below:

1. The robot sends an HTTP request to wait for one command
2. The user sends an HTTP request which contains the robot command
3. After having inserted the robot command in the HTTP response, the distant server sends it to the robot.
4. The HTTP response is sent to the user. This response can carry various information about command processing by the robot: in progress, not taken into account, ...

4) Acknowledgment

An optional acknowledgment can be sent from the robot to the distant server. As soon as the robot has received or terminated a command, it can send an HTTP request to the distant server.

This optional acknowledgment can also be sent from the distant server to the distant user.

B. Synchronisation and Servlet programming

The distant server synchronizes asynchronous HTTP requests from the distant user and the robot. There is no synchronization across the Internet except for single HTTP requests. Synchronization only appears in the Servlet.

It is implemented by using Java monitors ("wait' and "notify"). When the robot sends an HTTP request, a "wait" is executed in the Servlet. When the distant user sends an HTTP request containing a robot command, a "notify" is executed to let the HTTP response including the robot command come back to the robot.

From the robot, a Java program only sends HTTP requests and processes the responses.
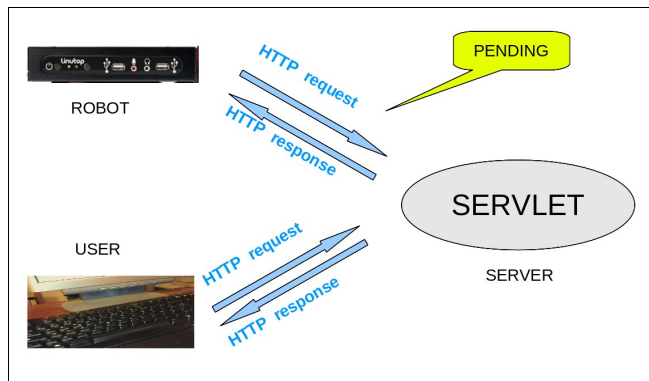

Figure 4: using Servlets for synchronization.

IV. FULL ROBOT SENDING IMAGES

The robot platform includes a PC computer and can be easily extended. One or more Webcam can be connected to the computer (Fig. 5). By using the same mechanism as that shown above, the robot can send information about itself or its environment to the distant server. In fact, as the distant user sends requests to make the robot move, the distant user send other requests to get information about the robot. In this chapter we will focus on how the distant user can get images taken by the robot.

A. The distant user asks for images

The distant user can click on a button in the user interface to ask for images. The distant user can also let the browser automatically ask for images. In this case, a thread in the browser periodically asks for images. In both cases, as seen in III-A, an HTTP request is sent to the robot to ask for an image.

B. The robot processes the image request

On the robot there is a Unix process to wait for robots commands (MOVE, TURN, ...). A second Unix process is used to produce the images. We use a third Unix process to wait for images requests and send the images. As the first process, these processes are automatically launched when the robot is powered.

The second Unix process is a local WEB server on the computer of the robot. Its role is to get images from the Webcam and make them available through a local Web server. We use MJPG-streamer [14] (also called MJPEG-streamer or M-JPEG-streamer) to do that . It is a light solution to stream JPEG files over an IP-based network. It can get images from a Webcam plugged on a USB port. We use the following options:

- "- - resolution 320x240" to send images of that size
- "- - device /dev/video0 -y" so the streamer can only use "yuv" mode to output the images (this is because our PC is USB2.0)
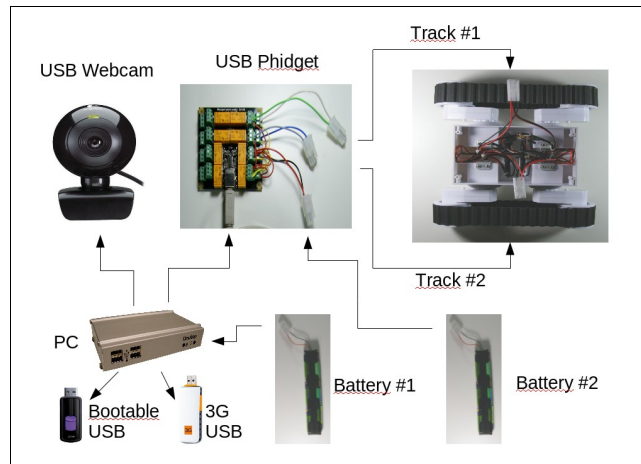- "- - port 8090" to ouput images on port 8090

Figure 5: robot components including Webcam.

MJPG-streamer is able to send streams over the network but we use it only to send snapshots. MJPG-streamer is not CPU hungry. It consumes less than 10% CPU. Several Webcams can be connected to the PCs and several MJPG-streamer can be launched (/dev/video0 port 8090, /dev/video1 port 8091, ...).

The third Unix process is used to wait for image requests from the distant user. It gets the images from the second process (MJPG-streamer) and sends them to the distant server which forwards them to the distant user. The process waits for image requests as if it was waiting for other robot commands. An HTTP request is sent from the distant user to the distant server which ensures synchronization. We just use a different Servlet to let this HTTP request pending until an image request comes from the distant user.

When the third process receives the HTTP response from the distant server, it sends a new HTTP request to the MJPG-streamer server. The aim of this request is to get an image. To get the image, we use an URL such as

http://localhost:8090/?action=snapshot

The HTTP response contains a JPEG image which is extracted and sent to the distant server as an attached file in an HTTP request. Standard Java classes "URL" and "HttpURLConnection" are used to do that. On the distant server, we use the Apache FileUpload package to extract the attached file from the request and we write it to a file which can be loaded and displayed by a Web Browser.

C. The distant server sends the image to the distant user

In fact, it is the distant user who asks for the image. To send the HTTP request from the browser, we use the Javascript language and the jQuery library [15]. As shown below, we use Ajax capabilites of jQuery to perform an asynchronous HTTP request.

```
$.ajax({
        type: "get",
        url: "../servletImage1",
        async: false,
        success: function (data) { ... }
});
```

The HTTP request is processed on the distant server and we have seen that the robot finally sends an image to the distant server. A file containing an image is created on the distant server.

Synchronization is required inside the distant server. The HTTP request from the distant user must be let pending until a new image comes from the robot. At that time, a response is sent to the distant user. On the user side, the Ajax call is a success and a Javascript function is called. This function has one parameter indicating whether an image is available or not. If there are several distant user requests pending, only one will get the ability to display the new image and the other distant user requests will have no effect on the user interface.

To display the image, we use jQuery to modifiy the DOM (to modify the HTML elements). An "img" HTML tag is present the distant user Web page. We use jQuery to modify the "src" attribute and change the displayed image.

$("webcam0").attr("src",
        "../servletImage2/?val="+Math.random());

We use a Servlet to ensure that one image cannot be sent twice to the distant user. The Servlet always sends the last image produced. It sends an HTTP response whose content type is "image/jpeg" and content is a JPEG image. The "Math.random" parameter shown above just avoids the image staying in a cache. A network failure will not affect the system. Only some images will be lost.

D. The user interface

The user interface is a Web page. A form (not shown below) is used to identify the distant user and to ensure that the robot is available. When identified, the distant user can use buttons to make the robot move. Images are displayed as they come from the robot.

Two parameters are available in the interface shown in Fig. 6. The distant user can set the duration of a robot command and the number of ms between two image requests.

The distant user can also monitor the network state and get information about the time required to get the last image.

E. Performances

Figure 7 shows the working robot powered with AA batteries. By using a WIFI connection, the robot sends an average of three images per second. By using a 3G connection, it is almost one image per second.
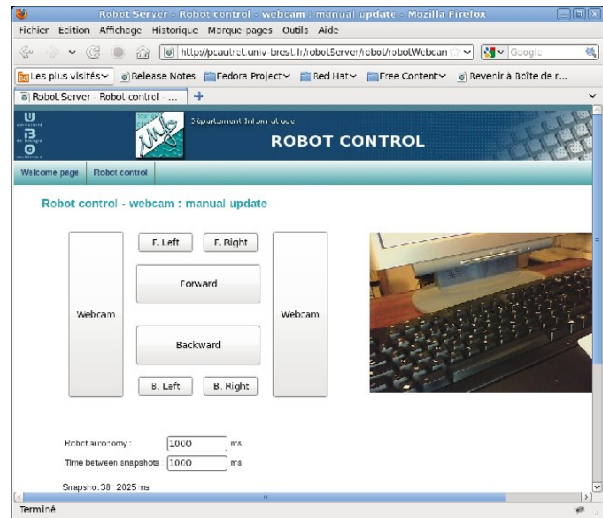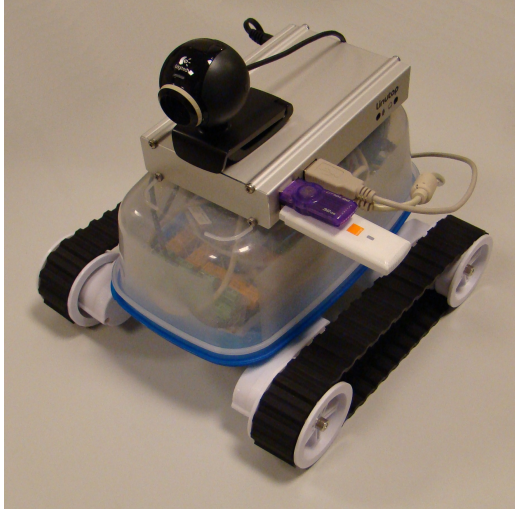

Figure 6: the user interface.

Figure 7: the working robot.

CONCLUSION AND FUTURE TRENDS

The proposed system can work indoor or outdoor. It uses free software (the Linux operating system) and thanks to HTTP, can recover from temporary network failures. In the future, it could be integrated in an ubiquitous environment for remote monitoring. It is a cheap and modular platform. The computer boots from a memory which is external. The mechanical and electromechanical parts are separated from the computer which is itself separated from the digital storage medium that carries the operating system. In case of failure of a component, the system can be repaired and restarted in a short time. MTTR (mean time to repair) is significantly reduced.

The hardware and the sofware of the proposed robot can be easily upgraded. Local or remote image processing could be performed. GPS or UWB (Ultra Wide Band) components could be added for localization. If one component is added, the operating system and the software can be upgraded plug and play. There is only one USB key to replace.

When using the Linutop computer, autonomy is limited to 10mn. When using a laptop, it goes up to 1 hour or more but a laptop is not easy to carry. Using a small tablet PC will be probably a better solution as soon as a whole Linux system will be available for them.

In the future, a better network management could also be implemented. For example, as soon as the WIFI signal becomes weak, the system should be able to automatically switch to 3G.

REFERENCES

[1] K. Goldberg and R. Siegwart, Beyond Webcams : an introduction to online robots . The MIT Press, 2001.

[2] P. Le Parc, J. Vareille and L. Marcé, Web remote control of machine-tools the whole world within less than one half-second . In ISR 2004 : International Symposium on Robotics, Paris, France, March 2004.

[3] WoWee Rovio, a Wi-Fi enabled mobile webcam, http://www.wowwee.com/en/products/tech/telepresence/rovio/rovio, [Online; accessed June 29, 2011]

[4] Kim, J.H. and Kim, Y.D. and Lee, K.H., The third generation of robotics: Ubiquitous robot, Proc of the 2nd Int Conf on Autonomous Robots and Agents, December 13-15, 2004 Palmerston North, New Zealand

[5] Ha, Y.G. and Sohn, J.C. and Cho, Y.J. and Yoon, H., Towards a ubiquitous robotic companion: Design and implementation of ubiquitous robotic service framework, ETRI journal, volume 27, number 6, pp 666-676, Electronics and Telecommunications Research Institute, 161 Gajeong-Dong, Yuseong-Gu, Daejeon, 305-350, South Korea, 2005.

[6] Kim, J.H. and Lee, KH and Kim, YD, Ubiquitous robot: Recent progress and development, SICE-ICASE, 2006. International Joint Conference , pp.I-25-I-30, Oct. 2006

[7] HTTP tunnel, http://en.wikipedia.org/wiki/HTTP_tunnel, [Online; accessed June 29, 2011]

[8] J. BAXTER, Introduction to the Miabots & Robot Soccer, http://www.asap.cs.nott.ac.uk/~robots/talks/asap-talk-2.ps, [Online; accessed June 29, 2011]

[9] Linutop, http://www.linutop.com, [Online; accessed June 29, 2011]

[10] Phidgets, http://www.phidgets.com, [Online; accessed June 29, 2011]

[11] The Apache HTTP Server project, http://httpd.apache.org, [Online; accessed June 29, 2011]

[12] Apache Tomcat, http://tomcat.apache.org/, [Online; accessed June 29, 2011]

[13] Java Servlet Technology, http://www.oracle.com/technetwork/java/javaee/servlet, [Online; accessed June 29, 2011]

[14] Mjpg-streamer, http://sourceforge.net/projects/**mjpg-streamer**/, [Online; accessed June 29, 2011]

[15] jQuery, http://jquery.com/, [Online; accessed June 29, 2011]