A client-server teleoperator for the EyeBot micro-robot

Esther García Morata
egarcia@infotool.es
Infotool Computer

José M. Cañas
jmplaza@gsyc.escet.urjc.es
tel: 916 647 468
Universidad Rey Juan Carlos

Vicente Matellán
vmo@gsyc.escet.urjc.es
tel: 916 647 472
Universidad Rey Juan Carlos

Abstract

This paper describes a teleoperator system developed for the EyeBot robot under a client-server architecture. The teleoperator is made up of two different programs: a server, running on robot's microprocessor, and a client running on a PC. They talk to each other through a serial connection following an specific protocol. The server gathers sensor information required by the client, sends it and executes movement commands. The client asks for sensor data to the server on behalf of the human operator and sends motion orders. It also shows her a graphical interface, so she can see the values from all robot sensors and she can request different movements to its motors and servomotors.

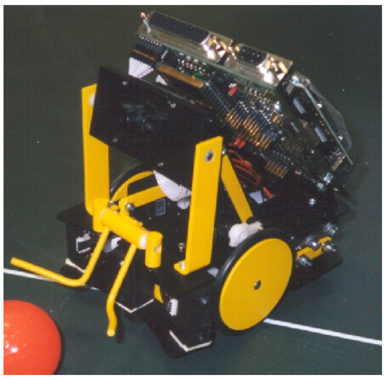
Figure 1: EyeBot robot

1 Introduction

The main target of this work was to develop a complete teleoperator for the EyeBot[1] [2] microrobot. Teleoperators are man-machine systems that augment and extend human sensory and manipulative abilities to remote or dangerous places [1]. In this way teleoperation has encountered application in space programs like Sojourner robot and industrial environments like nuclear plants operation. In our context a teleoperator is a useful tool for microrobot demos and the building of the teleoperator made the research group familiar with the new robot, its capabilities and its programming environment.

The main advantages of this EyeBot robot[2] over other small models like Lego or Khepera are the local camera, the radio link and a higher computing speed. These features make it suitable for autonomous behavior development, and even for playing in the Robocup[2]. EyeBot is endowed with three infrared sensors, a digital camera and two encoders for dead reckoning. In the actuators side, it has two DC motors for differential steering and two servomotors, one for a kicker and another for the camera movement, as shown in figure 1. Additionally it has a serial port offering 115200 bauds and a radio link providing 9600 bauds in a wireless way for communication purposes.

[1]http://www.ee.uwa.edu.au/~braunl/eyebot
[2]http://www.robocup.org

For interaction with human user the robot has an LCD and four buttons in a row. All these devices are connected to a motherboard with a Motorola 68332 microcontroller, running at 35 Mhz.

The operating system of the robot and user's programs run in this platform using its 512KB of flash ROM and 1MB of RAM. Its OS is called RoBIOS (Robot Basic I/O System) and has three main components: a console for program downloading and execution; a table with hardware devices; and a application program interface which provides user programs with access to robot resources. These programs are written in ANSI-C in a PC, compiled using a cross-compiler and their executables are downloaded into the robot through the serial port.

Several teleoperators have been developed for different indoor and outdoor robots [3][5][8]. For instance Xavier robot[3] [5] from Carnegie Mellon University and Hermes robot [8] from Instituto de Automática Industrial have the teleoperators shown in figure 2. Such teleoperators include graphical interfaces to see the robot sensors values, like sonar data, bumpers, dead reckoning, etc. In addition to telesensing the interfaces also allow the teleactuation of the robot. As shown in left part of figure 2, Xavier can be commanded to move left, right, forward, backward using graphical buttons. Human operator can also click on any point of top right joystick window in Hermes teleoperator's

[3]http://www.cs.cmu.edu/~xavier

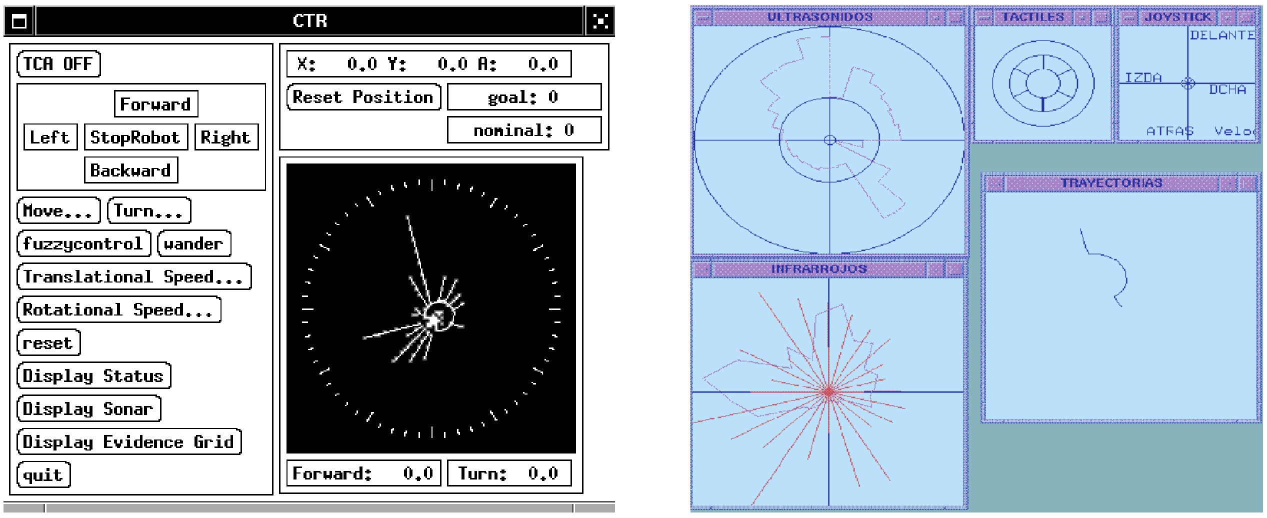

Figure 2: Teleoperator GUIs for Xavier (left) and Hermes (right) indoor robots

display (right part of figure 2) to make it move at certain translation and rotation speeds.

2 Client-server teleoperator

The teleoperator is compound of two different processes, client and server, communicating each other, as shown in the figure 3. The *server* runs on board the robot and its main function is to send required sensor information to the client and to implement motor movements as ordered. The *client* runs on the PC and shows a graphical interface to the human operator, who can observe and ask for different sensor values and command different movements to the robot.

This client-server distribution has been followed by several robots in the robotics community. For instance, Saphira architecture comes with a server program running in the Pioneer robot [6]. In TCA architecture developed on Xavier robot all control programs are rooted in a server called CTR [5] running inside one of the robot computers. This distribution allow different clients, maybe running in different machines, to connect concurrently to such module using communication facilities. The main advantage of this distribution is than client programs can run everywhere, as far as connection to the server is provided. This way, the limitation to on board computing power can be overcome distributing the clients to other off board machines.

The server developed in this work, as is, can be attached to the teleoperator client, but also to any other potential client holding the communications protocol. For instance, an autonomous controller which needs to obtain sensor readings and closes the control loop off board sending back movement commands. This is

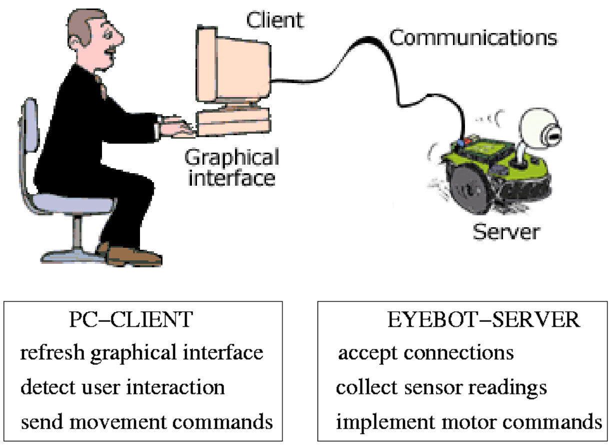

Figure 3: Client-server architecture for teleoperator

very convenient in the small league of the Robocup. In that scenario computing power of small robots usually is very short, and information coming from a birdeye camera is received and processed on an outer PC. For example [7] developed fast local control loops, which are continuosly activated and parameterized from an off board machine depending on its analysis of the current image.

2.1 Communications protocol

A *communication protocol* has been developed to rule the dialogue between server and client. It has been specially designed to fit teleoperator requirements. There are three kinds of data: short sensor readings, movement commands and images. Each type has its own size and timing, and so three different types of transmissions were developed, as shown in figure 4.

1. Infrared values and encoders data are short sen-

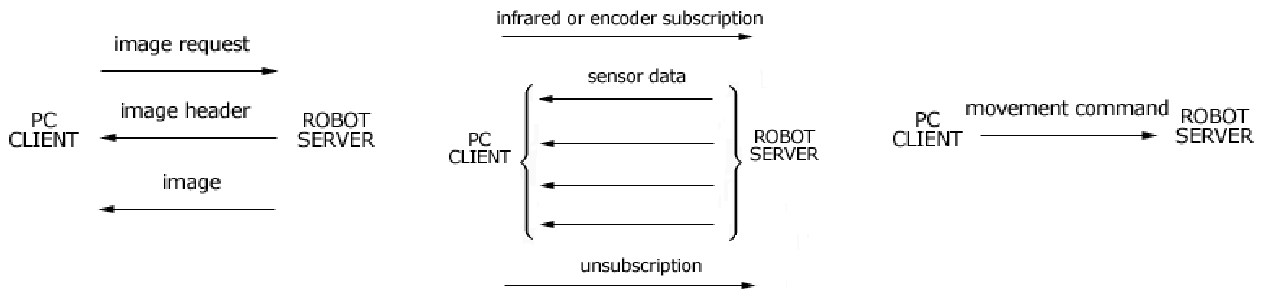

Figure 4: Three protocol transfers

Online message format	*Description*	*Sender*
INFRAREDSUBSCRIBE\n	PSD subscription	client
INFRAREDUNSUBSCRIBE\n	PSD unsubscription	client
INFRARED LEFT (l,%d) RIGHT (r,%d) FRONT (f,%d)\n	PSD readings	server
ENCODERSSUBSCRIBE\n	encoders subscription	client
ENCODERSUNSUBSCRIBE\n	encoders unsubscription	client
ENCODERS LEFT (l,%d) RIGHT (r,%d)\n	encoders readings	server
GREYIMAGEREQUEST\n	Grey image request	client
GREYIMAGE (rows,%d) (cols,%d) (Bpp,%d) (max,%d)\n	Grey image	server
COLORIMAGEREQUEST\n	Color image request	client
COLORIMAGE (rows,%d) (cols,%d) (Bpp,%d) (max,%d)\n	Color image	server
COMMANDED DRIVE_SPEED (ds,%f) STEER_SPEED (ss,%f)\n	traslation and rotation speeds command	client
STOP\n	stop EyeBot movement	client
SERVOKICK (angle,%d)\n	angle command for kicker servo	client
SERVOCAM (angle,%d)\n	angle command for camera servo	client
HELLO\n	start message	client
GOODBYE\n	closing connection	both

Table 1: Protocol messages

sor readings generated in the server and they are sent through simple messages following the pattern ``header body \n''. Client can subscribe to such readings (sending a subscription message) and the server sends them as new data are available, without continuous requests by the client.

2. Motor commands are generated in the client. They are sent using another similar short message with a different header. There is no subscription here.

3. Images are produced in the robot at a higher rate than can be sent through serial port, so we chose on demand transmission for them: for every image an explicit request must be issued. The maximum serial speed reaches 115200 bauds, such bandwidth is not enough to transmit 80x60x3 bytes per image, at 4 fps.

All allowed messages are presented in table 1. The online messages are sequences of ASCII characters which make the protocol even more machine independent, suitable to little-endian and big-endian systems. In this way integer and real numbers are translated into the ASCII codes for their hundreds, tens and units. The first characters of the short messages forms the header, which helps to develop receiver routines and to decode the body of the message accordingly. Images are transmitted as a sequence of pixel values, with no compression. One single byte per pixel in greyscale images, and three bytes per pixel for RGB images. They are always preceded with a header message specifying the type of image and its size in pixels.

3 Client in the PC

The *client* runs on the PC and shows a graphical user interface (GUI) to the human operator, as shown in figure 5. From it she can ask for any sensor value: images from the camera, encoders data or infrared readings. The client displays them in the interface, in a visual and very intuitive way. The GUI allows to teleoperate the robot movement, so it can be commanded to rotate or advance from the PC. Also the camera and kicker angles can be set at will. It has been written using the Xforms library [9] for X-Window, which is the most extended window system for Linux machines.

The GUI has several elements, the main one is a canvas showing the space around the robot from birdeye point of view. The scale of this canvas and its position in the bidimensional world can be customized, and even put in a follow-robot mode. A robot sketch is continuously displayed in the canvas and it is moved

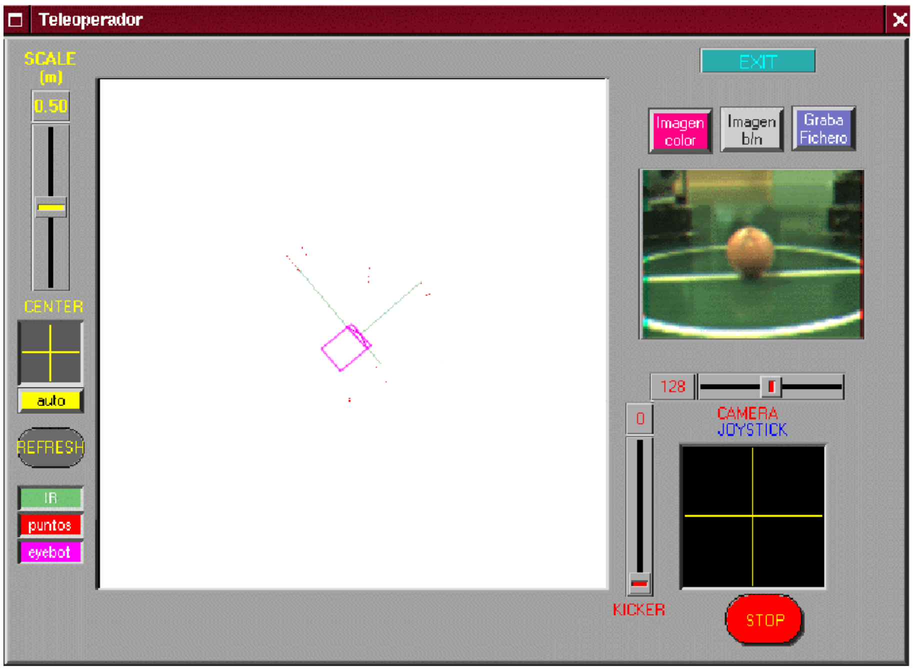

Figure 5: Teleoperator graphical interface

accordingly to encoders data, reflecting any displacement or turn of the real robot. Obstacles close to the EyeBot are detected with its three infrared sensors. Their readings are represented with three green rays showing their last value. They depart from the sensor positions in the robot sketch and their length is proportional to the measured distance. At the end of the line a red point is displayed to show that an obstacle was detected there. There is a short term memory of such points to observe wall and obstacle profiles. This objective visualization improves the subjective one chosen for Xavier and Hermes robots in figure 2, always centered on the robot. It shows robot movement inside its environment in a more intuitive way. The client can also display the images obtained with EyeBot camera, both in color or in greyscale. Their real size is 80x60, so every image pixel is extended to a 2x2 pixels in the GUI, for better display. Visualization of all elements is optional, and can be activated pushing GUI buttons.

Using the GUI the human operator can move the robot at will, make it turn, advance or both at the same time. GUI provides a bidimensional control window called joystick window (botton right window in the figure 5). Its center means no motion. Horizontal axis is used for turns, implementing a position control of the angle. Vertical axis is used for traslation, implementing a speed control of the motors. The higher ordinate, the faster the robot movement. For emergency halts a big `stop button` is provided. Position of the servos devoted to the kicker and to the camera can be selected at will using two linear dials , placed around joystick window in figure 5. To finish the teleoperator there is an `exit button`. When pressed the client closes the GUI, unsubscription and goodbye messages are sent to the server.

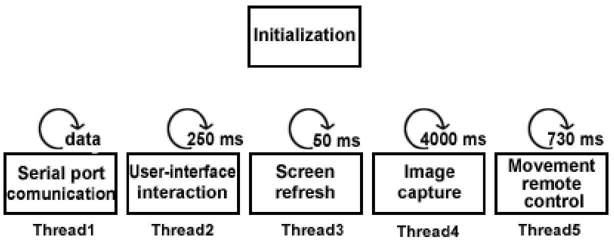

Figure 6: Client multithreaded implementation

Client side is implemented as five concurrent threads which start after an initialization phase, as shown in right side of figure 6. In the initialization phase the GUI is started, the connection to the server is established and the client automatically subscribes to encoders data. First thread manages communication with server and runs asynchronously, it is activated every time there are new data at the server port. It receives the data, compounds the messages, analyzes them and acts accordingly. Actually it has two operation modes, short message receiving mode and image receiving one. In the first one it searches for the header of the message, an integer number as shown previously, and looks for an end-of-line character to complete the message. In the second one it receives an specified amount of bytes, which can then take any value. This way data transparency is provided in the communication. Second thread checks every 250 ms whether the human operator has clicked on any graphical element. Third thread periodically displays all interface elements, updating output window. A fourth thread sends motor commands to the server every 730 ms if needed.

To avoid communications stall, fifth thread asks for images at a suitable frequency. The images and their requests have been separated from other sensor data because they consume more communications bandwidth due to their bigger size. The protocol forces on demand transmission, putting the flow control in the client side. This way the fifth thread asks for a new image every 4 seconds, in case of the human operator wants it and the last image has been completely processed. This frequency allows a slow update of images in the display without saturating the channel and letting the other sensor data to be exchanged at the speed they are obtained.

4 Server in the EyeBot

The *server* runs on board the robot and its main function is to send required sensor information to the client and to implement motor movements as ordered. It starts initializing all the robot devices that are to be used, like serial port, sensors and actuators. Then three concurrent threads are launched to serve sensors, execute motor movements and check the exit button respectively, as shown in figure 7.

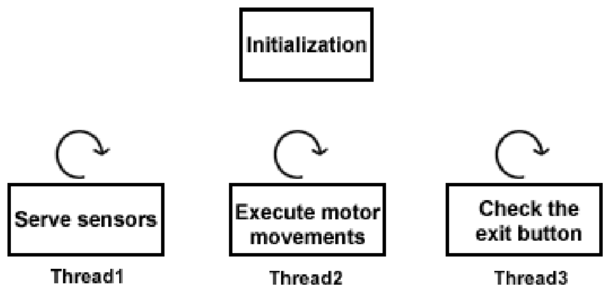

Figure 7: Server multithreaded implementation

First thread detects the opening and closing of client connections, collects sensor readings and is in charge of sending/receiving data to/from the serial port. Actually it runs an infinite loop: (a) checks for new data in the serial port, in non blocking way (b) captures the current encoders and infrared readings and (c) sends sensor values to the client if subscribed. Second thread controls motors movement. This one checks if the robot received a movement command, and if so, calls the primitive functions for motor activation. Current functionality includes simultaneous speed control for traction and position control for rotation. Once the robot has started to move, it will keep on doing so until the client issue a stop message or a new movement command. Third thread is a service one, which checks whether the human operator struck the exit button or not. In such a case it kills the other threads, frees all robot resources and ends the program.

5 Conclusions

A teleoperator system has been developed for an EyeBot micro-robot using a client-server architecture. Client and server have been implemented with multithreading programming and both communicate each other through a serial port following an specific protocol. The system allows the remote visualization of sensor data from the robot, and to move the robot from the PC using the mouse in a intuitive way. Teleoperator code, a video demo and a more detailed technical report[4] are available in the web[4].

Making the teleoperator let us to get familiar with the robot capabilities and its programming environment. That way, we have now the background and the required know-how to develop new behaviors with the same platform.

As a future extension of the teleoperator already presented here we are working in replacing the serial cable with a wireless radio connection through a radio port, changing low level communication primitives. The serial cable imposes a physical limitation on the movements of the EyeBot, so using the radio will provide more movement freedom to the teleoperated robot. Nevertheless it has to be taken into account that lower bandwidth is available through the radio link. A second future line is to expand the movements repertoire with speed control for turns and position control for traction.

References

[1] Antonio Barrientos, Carlos Balaguer, Luis Felipe Peñín, and Rafael Aracil. *Fundamentos de robótica.* McGrawHill, 1997.

[2] Thomas Braünl. EyeBot: a family of autonomous mobile robots. In *Proceedings of 6th International Conference on Neural Information Processing (ICONIP'99)*, pages 645–649a, Perth (Australia), 1999.

[3] Leandro Fernández García. Desarrollo de una interfaz gráfica para el control teleoperado del robot escalador ROMA. Proyecto fin de carrera, Universidad Carlos III, Madrid, 2000.

[4] `http://gsyc.escet.urjc.es/robotica/pfc-teleoperador.html`

[4] Esther García Morata. Construcción de un teleoperador para el robot EyeBot. Proyecto fin de carrera, Universidad Carlos III, Madrid, 2002.

[5] J. J. O'Sullivan, G.D. Armstrong, and Karen Zita Haigh. Xavier- The manual v0.4. Technical report, Computer Science Department, Carnegie Mellon University, April 1997.

[6] Kurt Konolige and Karen L. Myers. The Saphira architecture for autonomous mobile robots. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: case studies of successful robot systems*, pages 211–242. MIT Press, AAAI Press, 1998. ISBN: 0-262-61137-6.

[7] A. Oller, J.L. de la Rosa, R. García, J.A. Ramón, and A. Figueras. Micro-robots playing soccer games: a real implementation based on a multi-agent decision-making structure. *International Journal of Intelligent Automation and Soft Computing*, 6(1):65–74, 2000. Special Issue on Soccer Robotics: Micro-robot WorldCup Soccer Tournament'97.

[8] C. Vázquez Regueiro, J.M Cañas, and M.C. García-Alegre. Real time visualization of robot-environment states in local piloting. Technical report, Instituto de Automática Industrial (CSIC), March 1997.

[9] T.C. Zhao and Mark Overmars. *Forms Library: a graphical user interface toolkit for X.* http://world.std.com/ xforms/, 2000.