Platform-independent AIBO Navigation through the Internet

Zsolt Szabó, Zoltán Vámossy

Budapest Tech, John von Neumann Faculty of Informatics
Institute of Software Technology
Nagyszombat u. 19, H-1034 Budapest, Hungary
Phone: +36 1 3689840/178, Fax: +36 1 3689632
szabozs@bmfnik.hu, vamossy.zoltan@nik.bmf.hu

Abstract: The article is about planning and creating a software-pair (a client and a server) that allows safe and platform-independent navigation of the robot-dog named AIBO. The occurring practical problems and their solutions are described as well.
The two softwares were developed side by side. The task of the server is accepting commands from the clients and transmitting them to the AIBO if they are considered to be safe. In addition to that, the server has to accept the AIBO's camera picture, which is to be forwarded to the clients after compression. The clients have to control the AIBO using the connection with the server, decode and display the compressed camera picture stored in YUV format.
The communication between the softwares is TCP/IP based: the server-AIBO communication uses the R-CODE protocol developed by SONY, while the client-server communication uses a self-developed textual protocol.
The development of the system was made with Borland Delphi and Borland Kylix developing environments, and with the PHP scripting language. The system runs both under Microsoft Windows and under Linux/UNIX operating systems.

1 Introduction

1.1 Robots

With the decreasing size and increasing performance of computers, people could use computers for more and more different kinds of tasks: what once only existed in science-fiction books that could become possible in some years, and could even become reality in a decade. The same happened with robots: nowadays developing humanoid robots is one of the most quickly expanding sciences. Companies like Sony or Honda devote really much money on developing different walking robots [for example ASIMO]

However, non-humanoid robots were developed as well. The most successful of those is the AIBO developed by Sony, which is not only the most advanced piece of the entertaining robots industry, but it is also the most popular one.

This article is about developing a system that allows a platform-independent navigation of the AIBO through the Internet.

1.2 AIBO

Around the 1990s, Sony Entertainment started developing entertaining robots that were capable of communicating with the world. The aim of the project - besides entertaining people - was to create a robot that can operate, make decisions, move, react to stimuli independently from other devices, so a robot that can be identified as „intelligent". This two aims are also represented by the fact that the word „AIBO" can mean two things. Firstly, it can mean „Artificial Intelligent roBOt", and on the other hand, the Japanese „aibou" means „partner" or „mate".

The first version of these robots, the AIBO ERS-110 was made in 1999. The product was popular, so the engineers of the Sony kept on developing it. The second-generation robots were identified as ERS-210 and ERS-220, while ERS-311 (312) was the only member of the third generation. The most advanced robot dog today is the ERS-7m2.

These models were mainly different in two things: firstly, their AI programs became more and more complex, and secondly, more and more sensors were put into the robots. The ERS-311 is an exception from that rule, since there are fewer and less sensitive sensors in it, and its AI is far weaker, it recognizes fewer words than its predecessors do.

There are two devices from the many that should be emphasize: firstly, there is a slot for a WiFi PCMCIA card in the AIBO, and with that it is possible to establish standard wireless TCP/IP connection with the robot. The other device is the camera situated in the AIBO's nose. It can provide us with two kinds of images: one normal image that shows what is in front of the dog, and one special with an integrated color-detection. Using that, it is easy to find the pink ball that is given with the AIBO.

Budapest Tech has an AIBO ERS-210 since 2003, so the system was developed for that model.

2 Main Operational Parts of the System

After presenting AIBO, this paragraph deals with the basic structure of the project. The system has two main parts: there is a server program, that controls the AIBO

directly through local network, and there is a client program, that does indirectly the same by connecting to the server over the Internet.

It is important that while the server can only communicate with the AIBO using a protocol developed by Sony, the client-server communication uses a self-made protocol that is from now on referenced as RIC (Remote Internet Controller). The transformation between the two protocols is the task of the server program. The advantage of the solution is that this way the client practically becomes independent on the controlled device, since the server handles the device-dependent commands.

The most important part of the system is the controlling server that does the actual controlling tasks. The commands received from the clients are stored in a Receive Queue (RQ). After that, the selection stage is the next, during which the RQ of the client actually controlled is processed. The server checks the commands read from the RQ whether they comply with the rules of the protocol or not. If they do, then the commands are transformed into a form that can be sent to the AIBO.

In addition to that, the AIBO constantly sends the camera picture to the picture server, that first compresses it, then forwards it to the clients.

3 Communication Subsystem

3.1 Choosing Protocols

3.1.1 Packet-level Protocol [1]

Using the Internet, there are many possibilities to transmit packets. That is why the usage of ICMP, UDP and TCP were considered.

Using ICMP, the messages would be really short and fast, but the ICMP was simply not created for client-server communication. Other possibility would be to use UDP that would be good because the UDP packets are short, and because the lack of ACK packets cause that the same bits/second rate means faster data transfer speed. Meanwhile the lack of feedback means that to sign and to correct packet losses, an application-level filter should be implemented, because the loss of a packet that contains for example a controlling command cannot be allowed.

That is why TCP was chosen. This protocol has strong rules: the packets have to be analyzed in the order they were sent, no matter what order they were received in. Using TCP packet losses are not to be taken into consideration, since there is a protocol-level filter for detecting such (the receiver sends back an acknowledgement of each received packet).

3.1.2 Application-level Protocol

This chapter is about possible ways of encapsulating the data into TCP/IP packets.

One possibility is XML web service. It has advantages: it is very easy to implement it, and there is no need to mess with the actual network transfer [sending and receiving data, putting data into variables], since this is done by a WSDL object in the background with XML communication. The disadvantage of the method is that variables those types are not part of the WSDL specification [for example pictures, structures, records, complex variables] can only be sent with encapsulating them into streams, and analyzing these streams takes more processor time. Another disadvantage is that because of the XML communication, this method is a little slower than byte-level TCP communication.

Byte-level TCP communication can be done with the Indy components *[2]*; from those the TIdTCPServer and the TIdTCPClient were used in the project. These components allow two computers to communicate on any ports using unencrypted, telnet-like raw communication. The advantage of using the Indy components is that not only byte-level communication can be used, but also it is possible to read and write lines terminated with enter character, to read and write different streams, and to read and write different typed variables. This way the organization of the network communication is simpler than it is with XML web service.

It is an important remark that in the developed system all TCP/IP communications are managed by a separate thread, so the main programs only have to deal with pre-processed data, the receiving of the data and the variable handling is done by the separate threads.

3.2 Communication with the AIBO

3.2.1 R-CODE *[6] [7] [9] [10]*

R-CODE was developed by SONY. This scripting language makes possible the direct controlling of the AIBO from a computer. After connecting to the AIBO on the 21002 TCP port, script commands can be sent, that will be interpreted and transformed to movements by the R-CODE interpreter stored on the AIBO's memory stick. The R-CODE commands are simple; the mini-commands are separated with a colon. In the „PLAY:ACTION:WALK:0:5" command PLAY means that the command is a motion, ACTION means it is a simple motion (MWCID would mean that it is a complex one), and WALK means what the motion is. The next two numbers are the parameters of the motion, the first number is the degree of the walk (0=forward), the second is the length of the motion in seconds.

The extension of the R-CODE is the R-CODE+. R-CODE+ allows binary communication through the 21001 TCP port and adds some text commands that for example allow low-level control of the AIBO's servos.

3.2.2 YART *[9]*

R-CODE actions can be easily rewritten using the YART. YART is a high-level developing environment that is first purpose is to permit changing complex actions and actions done in response to sensor values. This device cannot be used for controlling purposes.

3.2.3 OPEN-R SDK *[8]*

OPEN-R is a low-level programming environment developed by SONY. It is based on the GNU C Compiler, its syntax is similar to C++. Unlike R-CODE, OPEN-R programs are stored on the memory stick, and the AIBO's CPU directly executes the programs. That is why everything the AIBO is capable to can be done with OPEN-R. Its disadvantage is the complexity of the programming language, the aim of the Tekkotsu project is to make the developing in OPEN-R simpler.

Both the OPEN-R and the Tekkotsu are mainly used for programming AI, AIBO personality or automatized actions [games like robot-soccer].

As a conclusion, it can be stated that while R-CODE was mainly designed for controlling the AIBO through the WiFi connection, all the other programming environments were overcomplicated for this task. That is why only R-CODE+ was used in the project.

3.3 Compression

To minimalize the bandwidth usage in the client-server connection, it is essential to compress the transmitted picture. There are more possibilities for that, this chapter deals with those.

3.3.1 Using an External Compression Program

This solution would be the simplest, but because of the quick compression speed (5 times/second) this method cannot be used due to the time-consuming program calls.

3.3.2 Self-made Compression Routine

Despite the fact that Borland Delphi has integrated support for JPEG pictures, this format is not supported under Borland Kylix, so a self-made JPEG compression routine should have been implemented.

Even thought at first sight this may sound to be the fastest solution, this is not true: a compression routine that is completely written in Object Pascal cannot be faster than a compression library that has parts of the code written in assembly. This is why a good, fast and platform-independent external compression library was needed.

3.3.3 External ZLib library *[3]*

The binary formed ZLib library was the first that was implemented into the image sender routine. The library was 100% written in assembly, and then it was compiled and distributed as a Windows compatible library. It was very easy to use it, and from all the possibilities (including the Abbrevia component and the ZLib unit) this compressor routine was the best (compression rate: 29-35%) and the fastest as well.

This library would have been the best choice, but unfortunately the libraries did not work under Linux.

3.3.4 Delphi/Kylix ZLib Unit

The low-level ZLib unit was part of the development environment since Delphi6 and Kylix3. It uses two components, TCompressionStream and TDeCompressionStream to implement platform-independent compression and decompression. After implementation it turned out that the compression rate was an acceptable 42-48%, but the time needed for the compression was a little high (70ms/compression).

3.3.5 Abbrevia [4]

The freeware Abbrevia component library that can be used in both environments provides full-scale handling of various compression methods using visual and non-visual components. Basically, it knows the PkZIP, ZLib and GZip compression routines and it has many extra features too (like compression of directories and password-protected compression). In addition to those, it has surprisingly fast, unit-based stream compression methods called InflateStream and DeflateStream.

After implementation and tests, it turned out that the compression rate [mainly for dark pictures] is a little worse than with the ZLib unit (45-51%), but the compression speed is a lot better than with the ZLib unit (50ms/compression[1])

Altogether, the usage of Abbrevia turned out to be the best solution.

4 Image Processing Subsystem

4.1 YUV Processing

The AIBO's camera picture can be downloaded through the binary connection as an image with YUV10 format. The size of one frame is 31700 bytes. The first 20 bytes contain the header information, and after the header there is the Y component, followed by the U and V components. It is important that the Y component storing the grayscale values occupy alone as much bytes as the color-representing U and V components do together; so the color values are less accurate than the grayscale values.

The 10 bits/pixel YUV10 format is built up in the following way: in the first ten out of the twenty bits that represent two pixels are the two Y components representing one pixel each. The remaining ten bits are equally shared by the U and V component representing both pixels.

When showing the camera picture, the server and the client use the same YUV10➔RGB24 conversion. First of all, the YUV10 is transformed into YUV24 using bit shiftings. In this format the Y, U and V components are unchanged, but all components are represented in eight bits per pixel. After this, the following equations are used for the YUV➔RGB conversion:

$$\begin{aligned} R &= Y + V \\ G &= Y - 0{,}19 * U - 0{,}51 * V \\ B &= Y + U \end{aligned} \tag{2}$$

[1] Measured using Zprofiler *[5]*

[2] YUV➔RGB conversion suggested by SONY [11]

4.2 Color Detection

4.2.1 Integrated Color Detection

The AIBO has an integrated, fast color detection routine, so it may look natural that to recognize colored shapes this color recognition image should be used. As long as only the pink ball should be recognized, this color recognition image is perfect.

But after printing out same colored objects and trying to make the AIBO recognize them it turned out that for these object the integrated color detection does not work.

The AIBO's integrated color recognition is done according to a color detection matrix stored on the memory stick. The construction of the matrix is simple: it stores color IDs and YUV intervals paired to those IDs.

4.2.2 Self-made Color Detection

The problem is that this matrix cannot be modified with R-CODE, only OPEN-R has commands that can modify the intervals. Since the use of the wireless connection binds the system to the usage of R-CODE+, so an own color detection routine had to be implemented.

This self-implemented color detection works the same way the integrated one does. The difference is that to speed up the object recognition the picture size is decreased using the image-pyramid method. This way the result of the YUV thresholding is a four times smaller black and white image. In that image white pixel represents a pixel with a color that is close to the color we are looking for.

4.3 Object Recognition

4.3.1 Pixel Errors

This black and white image is however not noise-free. The Y component is more emphasized in the AIBO's camera picture, thus colors are not as sharp as it would be expected. This is why there are many pixel errors in the binarized picture, those are corrected using closing error correcting image transformations.

After correcting the pixel errors, the object is cut off from the image, and the object recognition is done on this small cut off image.

4.3.2 Shape Recognition

The algorithm used for shape recognition is similar to the one used at character recognition. One feature is the ratio between the amount of white pixels and the area of the image.

In addition to that, the program takes five rows of the cut-off image. In each five rows, the ratio between the position of the first white pixel and the cut-off the size of the image is counted, these ratios are features as well. Those ratios are firstly shown, and on the other hand, they are compared with the ratios of previously stored ideal shapes, currently the circle and the X shapes are stored, but it is very easy to add shapes to the system (these ideal shape-ratios are the average of the measured ratios during 1000 runs of the program).

For every stored shape two numbers are generated by the system: firstly the difference between the ideal area and the actually measured area, and secondly the cumulative absolute difference between the ideal shape ratios and the measured shape ratios. From these numbers it can be clearly decided that which object is currently seen by the AIBO. After tests, it can be stated that mainly the shape-difference is correct for classification, a number below 0.8 or 0.6 means that the shape of the object the AIBO is currently seeing is similar to the stored shape of the object. At the moment objects are only recognized if the whole object is in the camera picture.

5 Client-Server Connection

5.1 RIC

The most important feature of the RIC (Remote Internet Controller), the protocol used in the client-server communication is that it is device-independent. The module that keeps the connection with the clients has to transform the commands following the rules of he RIC protocol into R-CODE commands, and it also has to compress the camera picture received from the AIBO, and then transfer it to the clients.

In the RIC protocol, there is no separated channel for the binary data. The connection is basically textual, meanwhile a label is transmitted before the picture, so the clients can read the binary camera picture and then they switch back to textual mode. Since the server identifies the clients with their usernames, usernames has to be identical.

The RIC protocol is simple, only four commands can be used. With the SIMPLE and the COMPLEX commands the AIBO's actions can be controlled by the

clients, with the USERS command the server sends the client-list to the clients, and with the OBJECT message the server tells the clients that someone found the searched object, so another user can control the AIBO.

As previously stated the commands sent by the clients are stored in a Receive Queue until further processing by the selection stage.

5.2 Integration to Browser

5.2.1 Basic Principles

A certain level of platform-independency can be reached with making the programs for Linux and Windows, the aim was to make a client that is absolutely not dependent from the environment it is running in. That is why the client integrated to browser was created.

The aim was to create a client that can run in any web browsers, and that does not depend on any previously installed extra components. This is why the usage of JAVA and ACTIVEX embedded objects were not considered, rather a pure HTML based client generated with PHP script was implemented.

The communication with the server is file based, the PHP script and the server constantly checks the contents of different given files with known names (the same communication method is used by a module of the music player XMMS).

The server identifies the files of the different clients with the PHP's unique client ID called „session ID". When a PHP client connects, the server saves its session ID, so the server will know that which files should be checked for the commands sent by the client. After reading the commands from the corresponding file, they are copied to the RQ of the client, after that the command processing is the same as the one described for the TCP/IP clients.

5.2.2 Operation

After the user sends his username with an HTML form, the PHP script creates a file *logging.in* with the chosen username and the PHP session ID in it. The server then executes the same username check as in the case of TCP/IP clients, and it notifies the PHP script if the login was successful or not with the creation of files with known names (*accepted.in* and *refused.in*). If the login was successful, then an HTML form offering the same controlling possibilities that is offered by a TCP/IP client appears to the user.

During the connection, the client creates two files, in the first one the sent commands are stored, and the other one is a „ping", a file that is refreshed in every second. If that second file is not refreshed within thirty seconds, then the server

removes the client from the client list as it means that the user probably closed the browser.

The server creates two files as well, in the first one the AIBO's camera picture, in the second one the server messages are stored. It can happen that the communicational files are being accessed by the server and by the client at the same time, and to avoid that, a lock file is used by both sides to sign that the file is being written or read.

5.3 Results

5.3.1 Testing during Development

The development of the system and the necessary testing was completely done under Debian GNU/Linux operational system. Because of the multiplatform development, a Windows was necessary as well, so a Windows XP was installed to a virtual machine. On both operating systems the browser-client was made using the Apache web server and PHP scripting language. On Windows the 1.3.31 Apache and the 4.3.9 PHP, on Debian the current „unstable" 1.3.x and 4.3.x test versions were used.

There are many benefits of using virtual machines. First of all, it allows the simulation of a small network containing computers with different operating systems without having more computers. On the other hand, this way it was possible to develop and test all five parts of the system (windows based client and server, Linux based client and server, PHP client) on one computer.

5.3.2 Testing after Development

After the development, the testing of the mature system was done in two steps: firstly on a computer on a 100MBit local network, and secondly on computer with an average 512/128KBit ADSL line.

After the tests it turned out that the client can work on both network speeds (using uncompressed picture transfer a transfer speed larger than 1024KBit/second should be needed, with compression the 512KBit/seconds transfer speed is suitable for the image transfer that needs a speed of approximately 60KBytes/second).

With four clients the transfer speed requirement of the server is approximately 250KByte (=2.2 MBit)/seconds. That is why the server can only run on a computer that has a fast internet connection (10MBit/sec).

5.3.3 Results

During the tests it turned out that even thought the byte-level transfer and the image compression together lowered the transfer speed requirements, unfortunately the computer running the server still cannot be an average home computer.

A possible solution is the usage of video streaming instead of separated picture transfers, since movie compressing codecs like XVID or DIVX can produce far better results than a simple Zlib compression.

This way, the current limit of four clients/server could be increased as well.

In addition to that, if the system is running on computers with fast enough Internet connection, the tests showed that the system fulfills the objectives formed at the start of the project.

Conclusion

During the development, the following objectives were fulfilled:

- A server side system was developed for both Windows and Linux operating systems. The Linux version is capable of running on any UNIX systems that have the X11 and QT libraries installed. The server can efficiently control the AIBO and can handle the AIBO's camera picture.
- A client side system was made for both Windows and Linux operating systems. With the client, it is possible to connect to the server and control the AIBO indirectly. The client was also integrated to a browser.
- Using the developed shape recognition subsystem the object recognition task was implemented.

References

[1] Dénes Székely: Computer Networks, chapter 2
http://www.webgobe.ro/konyv/2tcpip.html

[2] The Indy Project
http://www.indyproject.org

[3] Delphi FAST Zlib © Roberto Della Pasqua
http://www.dellapasqua.com/delphizlib/

[4] Abbrevia component library *(ex-TurboPower)*
http://tpabbrevia.sourceforge.net/

[5] Zprofiler © Igor P. Zenkov
http://www.geocities.com/izenkov/download.htm

[6] Csaba Kertész: Introduction to the usage of AIBO, 2004, Budapest Tech

[7] Sony Corporation: Open-R-Code
http://www.us.aibo.com/openr_code.htm

[8] Sony Corporation: Open-R SDE
http://openr.aibo.com

[9] AiboHack RCODE page
http://aibohack.com/rcode/index.html

[10] AiboRemote
http://aibohack.com/210/AiboRemode252.zip