

Palm Wireless Interface For Distributed Robots

Sarun Teeravechyan
GLCA/ACM
Knox College

Computer Science and Mathematics Division
Dr. Lynne E. Parker
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831 – Box 6355

Prepared in partial fulfillment of the requirements of the Office of Science, Department of Energy
Program under the direction of Dr. Lynne E. Parker in the Computer Science and Mathematics
Division at Oak Ridge National Laboratory.

Participant: _____
Signature

Research: _____
Advisor Signature

Table of Contents

| | |
|--------------------------------|----|
| Abstract | 3 |
| Introduction..... | 4 |
| Materials and Methods..... | 6 |
| Results | 7 |
| Discussion and Conclusion..... | 10 |
| Acknowledgements | 11 |
| Figures..... | 12 |
| References | 16 |

Abstract

Palm Wireless Interface For Distributed Robots.
Sarun Teeravechyan, Knox College, Galesburg, IL 61401
Dr. Lynne E. Parker
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831

There are situations in which the best course solution would be to deploy a team of distributed mobile robots. However in these cases, it is most likely that one would be confronted with geographically adverse conditions for setting up the communications system. This particular problem is addressed by introducing the PalmVx PDA into the relay system, wherein it would act as the interface through which humans could communicate radio commands to the robots. The handheld device acts as the first link in a chain through which remote commands are issued. The GUI on the PalmVx would take in parameters that would allow a connection-oriented Ethernet "conversation" with the specified server. This server would act as the buffer stage, taking in simple function commands from the remote Palm device, parsing them and piping them out to the appropriate set(s) of robots. These instructions would, in turn, serve to initialize or shut down distributed robotics functions that have been preprogrammed onto the team. All in all, the lower layers of the structure would be transparent to the user. The only interaction between the operator and the system would be at the application layer, which constitutes of the Palm GUI. From there, all simple functions can be issued to the robotics team.

Research Category (Please Circle)

ERULF: Physics Chemistry Biology Engineering Computer Science Other _____
CCI: Biotechnology Environmental Science Computing

TYPE ALL INFORMATION CORRECTLY AND COMPLETELY

| | |
|-----------------------------------|-------------------------------|
| School Author Attends: | Knox College |
| DOE National Laboratory Attended: | Oak Ridge National Laboratory |
| Mentor's Name: | Dr. Lynne E. Parker |
| Phone: | (865)241-4959 |
| e-mail Address: | parkerle@ornl.gov |
| Author's Name: | Sarun Teeravechyan |
| Mailing Address: | K-1575 Knox College |
| City/State/ZIP: | Galesburg, IL 61401 |
| Phone: | (865)482-6433 |
| e-mail Address: | steerave@knox.edu |

Is this being submitted for publication?: Yes No
DOE Program: ERULF CCI PST (circle one only)

Introduction

The invention of the computer has vastly increased human capacity to make fast and accurate calculations. As greater advances are made in this particular field we begin to rely more and more on these machines to do our grunt work, which would previously have taken us hours if not days, months or even years. Such amazing feats of calculations, once considered impossible, are now mundane tasks we take for granted. Computers are now in charge of controlling air traffic, calculating stock indexes, predicting the weather as well as regulating a huge portion of our information exchange through an intricate web of connected networks we call the Internet. Sometimes one begins to wonder how much potential these machines have, how much easier they can make our lives.

To date, one of the ultimate computer services is in the concept of a robot. Modern robots are built for a specific task, which they unerringly perform. Some of the earlier forms of robots were used in automotive assembly plants, where they execute with pinpoint accuracy the manipulations needed on an assembly line to put the vehicles together. As robotics technology progress, they have been specialized to perform more complex tasks which are too dangerous, require too much precision or just plain impossible for humans to perform. Now they are responsible for the nanometer silicon linings on a microchip as well as collecting dirt samples from the planet Mars. The uses that we have found for these automated systems are far and wide, and their potential limited only by our imagination. At the rate our technology is currently advancing we can be sure to expect bigger and better things. However, what lurks in the horizon for us in terms of robotics? The answer is artificial intelligence.

At the present, humans do most of the decision-making for the robots. We decide what specific tasks they are to perform, and in the unfortunate case of it encountering an unanticipated

situation, we have to step in and extract it from its predicament. Any novel tasks we want robots to execute have to be programmed into them. However, with developed AI, it is possible to expect a robot to dynamically solve a problem on the go or even learn new tasks from another robot. This concept of an “intelligent” machine is where we are headed towards now. In order to make them intelligent we have to give them the power of perception. They should be able to gather information from their surroundings and incorporate them into a decision-making algorithm and then respond appropriately. The next step would be to include a number of these robots into a team with a specific task in mind. Thus the birth of the autonomous distributed robotics systems.

In the creation of such a system there are many components that we have to develop. There are two main segments that the project could be broken into: the Artificial Intelligence and the controls. The AI part would involve research on decision-making algorithms. That subject, however, does not pertain to this paper. The purpose of this particular research is to create a user-friendly wireless controls system for a specific team of autonomous robot team.

As with any interface, the top layer should be intuitive and unambiguous while simultaneously presenting the user with the necessary options to initiate communications with the robots and set them to their tasks. While for some purposes a complex interface is necessary to achieve the desired levels of control, we need only the bare minimum. This is done to moderate the learning curve as well as maximize retention rate. From an economics point of view this would lower costs of training new labor as well as reducing the chances of a costly error. Detailed error checks and warnings are also integral parts of a user-friendly interface.

What lies hidden beneath the visible interface is all the code and hardware that is required to make a functional relay system. Sheltered inside that hull are the command translation, connection establishment, data transfer as well as various parsing tasks. The workings of the lower

layers, however, should always be transparent to the user. All these functions should operate flawlessly and smoothly enough so that it would not be necessary for the human operator to have any knowledge of the system's inner mechanisms. The program should be robust enough so that any errors that do occur would be at the fault of the user.

As the field advances, the direction it follows is towards increased applicability of the technology. Most research has been conducted under laboratory environments, where the controls are set behind lumbering PC's. Though tests may prove that certain robot teams are able to perform the required tasks, it is still necessary to bring the team out to the task site to do the real job. A problem arises when the mobility of the controller is brought into consideration. It would be impractical, and in many cases, impossible, to tow a computer along to the work area. This is where the Palm Pilot comes into the picture. It is a compact, yet versatile piece of machinery. Palm Pilots can be programmed to perform complex functions and they are capable of wireless networking. It is, in essence, the ideal top layer interface device. One can envision a man at an edge of a battlefield deploying a team of robots to sniff out mines. This application demonstrates how versatile this technology can truly be.

Materials and Methods

The creation of the communication relay system can be divided into three sections: the server side, the client side and the wireless interface. These three parts are interconnected by a series of network programs that allow for data transfer through exclusive sockets. The lines of communication are set up specifically for our particular application and are promptly torn down after the program is closed. This allows for a steady, uninterrupted dialogue between the three network devices. Figure 1.1 shows the overall structure of the system, putting each of the aforementioned components in their place.

The interface device is the unit that contains the GUI. It represents the sole medium of interaction the user has with the system. From here, commands are issued to the robots. The robot team constitutes the client side, which receives all directives that are sent out from the controller of the interface. The server side is a workstation that acts as a mediator between the two aforementioned components. Its job is to first establish a reliable connection that would allow free flow of information. The other responsibility it fulfills is in the translating of commands so that both sides would be able to understand each other. Ultimately, it also shuts down the connection when no longer needed.

The server side of the system is run on a Solaris platform on a Sun Sparc20 Workstation, locally identified as Neptune. The development work of the program, which was written in standard C, was actually done for another project. However, with a little modification, it suitably fit our needs. The purpose of the program, which was named `communication.c`, was to set up a listening port, allowing commands to be relayed in from the PDA interface, parsed, and then sent out to the waiting client side program.

The client side consists of four Nomad 200 series robots from Nomadic Technologies. Each of these robots has Linux as their operating system. They also contain radio technology that allows their end of the network program to be remotely uploaded into their directories. Once initiated, they stay online and wait for commands to be sent in from the communications server. The information they receive when contacted allows them to determine what type of function needs to be performed. However, it must be kept in mind that the aforementioned task must have been preprogrammed into the team as well as modified to work within the relay system. We focused on one such function - `sobounce.c`

Sobounce.c has been revised and uploaded to each one of the four robots and is used to test the interactions between the server and the client sides. Once activated, each unit begins to wander around. Readings from their sonar prevent them from hitting each other or any path obstacles. In the original version, the program would keep on running until manually interrupted by the user pressing Ctrl+C. To serve the purposes of this communications system, however, the program must be able to receive and respond to a “stop” command. This feature has been implemented in the new version, and now will terminate the program on reception of the appropriate trigger command.

The choice interface device is the PalmVx, chosen for its large memory (8 MB), its ability to connect to a wireless Ethernet device and its mobility. The Mercury-EN from Nomadic Technologies serves as the Ethernet device that plugs the PalmVx into the network. As mentioned earlier, this is the only section in the whole relay system that interacts with the user. This fact has to be kept in mind when designing the interface program.

The development of interface code was originally done on Red Hat Linux 6.0 and was later moved to Windows 2000. Though some may argue otherwise, Windows does have an advantage in Palm development with its wide options and availability of Interface Development Environments (IDEs). Codewarrior Palm came as the highest recommended tool, and was subsequently chosen as the choice IDE for the project. Its power lay in its ability to allow coders to visually place and allocate resource objects. Form objects, text objects, menus and menu bars can all be drawn out on a simulated Palm screen. Once compiled it would create the necessary “.h” resource file that the main program can call. Another benefit of the Codewarrior Palm Constructor (shown in Figure 1.2) is that it dynamically assigns (and reassigns) indexes to the objects you create. As a result a programmer can do all the resource manipulation he/she wants without ever having to worry about

the details of changing their identification numbers. The only knowledge they need would be of the objects name, which in most cases would be intuitive and reflective of their functions or position in the program.

The main program was written in standard C. Although the IDE (Figure 1.3) used did have a lot of features that aided in debugging and organizing the code, the most useful tool used was an external program downloaded from the Internet. This piece of software is called POSE (Palm OS Emulator), and is considered the best Palm debugger available. When writing code for a different system, one of the most tiresome activities is in the testing of the newly compiled program. In order to do that, it would require the uploading of the whole program onto the other device. This is where POSE saves developers a lot of time. Instead of having to install the program on the PalmVx every single time a new compilation is made, it provides an emulation of the Palm on the PC (our development platform). POSE accurately reflects how the Palm would run the software as well as how it would respond to erroneous code or memory misallocation. Another strong feature it provides is called Gremlins. This is basically a series of instructions sent out by the emulation software to try as many actions as possible in order to test the robustness of the software. This function is usually activated towards the end of the development process to double check on product reliability.

The interface itself is made to be as simple as possible bearing in mind that the user would also need some degree of freedom in their configurations. There would be two main form objects in the GUI. The first one would be responsible for the establishment of the connection between the device and the rest of the relay system. It would ask for the host name of the communications server as well as the desired port (default port is set to 10000, which is also the default on the communications.c). The other form is where the user would send commands to the desired

robot(s). Any one or any combination of the robots could be picked as well as any available choice of functions to run. All this information would be sent out to the communication server, parsed and then passed on to the robots.

Results

The relay system has a number of working components such as the established communication lines between the server side and the client side. The two are able to send and receive data through a menu-driven interface. For example, commands sent through a makeshift interface on the communications side can initialize and execute `sobounce.c` on the Nomad 200 robots. Though this is in part due to the generic nature of the system, made to allow for future development and implementation of other applications.

The interface side of the project, however, is still in the making. A compiled version of the program has been written and uploaded. The basic structure of the program has been done and it only waits for modifications to be made on particular variable passing functions. As of now, memory allocation procedures pose as the largest obstacle in completing the interface application.

Discussion and Conclusion

At this point, two of the three components are able to communicate with each other. However a bit of work is still necessary for even those parts to be fully operational. The fact of the matter is that the code is functional only at a very preliminary level. None of the checks for robustness have been done, and this is definitely the area that needs the most improvement on. Considering that the media used is wireless Ethernet, it would be wise to have some sort of fail-safe device that would be able to eliminate radio interferences. A guarantee of reliable information flow should be implemented.

Another area that could be improved upon is concerning user feedback. At the present there is inadequate information flow coming back to the server. Occasional progress reports should be sent to make the user aware of the current state of communications. This provides a firmer sense of product reliability, as well as facilitating technical solutions in the case that errors do occur.

One of the major problems in the development of the GUI component is in understanding the inner workings of programming a Palm application. Not only does it require good knowledge of its library of built-in functions, the basic process of memory allocation works differently from the way the majority of machines do. Instead of reading and writing data into files, a Palm application accesses data directly from its memory. However, a lot more rules and regulations govern this action. In order to manipulate memory without error, you have to set a *handle* to it, and lock it in place. Only then will it be stable enough to be worked with. In order to finish up the coding of the GUI, extensive knowledge and experience in Palm programming is recommended.

Acknowledgements

This research is sponsored by the Engineering Research Program of the Office of Basic Energy Sciences, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

I would like to thank the United States Department of Energy- Office of Science for giving this opportunity to participate in the GLCA program.

My thanks also go to my mentor Dr. Lynne E. Parker, Computational Intelligence Group Leader, and the entire staff at the Center for Engineering Sciences Advanced Research Laboratory at Oak Ridge National Laboratory in Oak ridge, Tennessee. Also special thanks go to Andrew McDowell, who patiently waited for me to get my Oak Ridge Science Semester applications in, as

well as Aleksander “BigAl” Stefanovski for bringing the application to me when I was incapacitated with a broken fibula.

The research described in this paper was performed at the Center of Engineering Sciences Advanced Research Laboratory; a national scientific user facility sponsored by the United States Department of Energy and is located at the Oak Ridge National Laboratory.

OVERVIEW

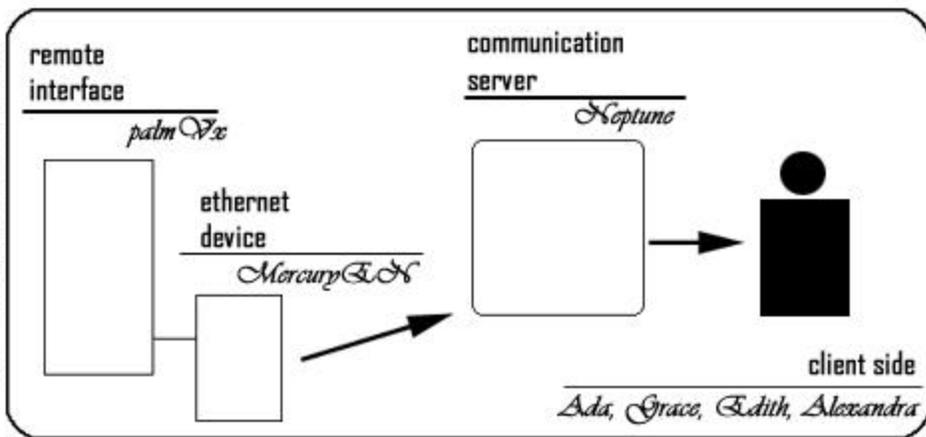


Figure 1.1 This figure represents the structure of the relay system. The three main components are identified as the Interface, Communication server and the client side.

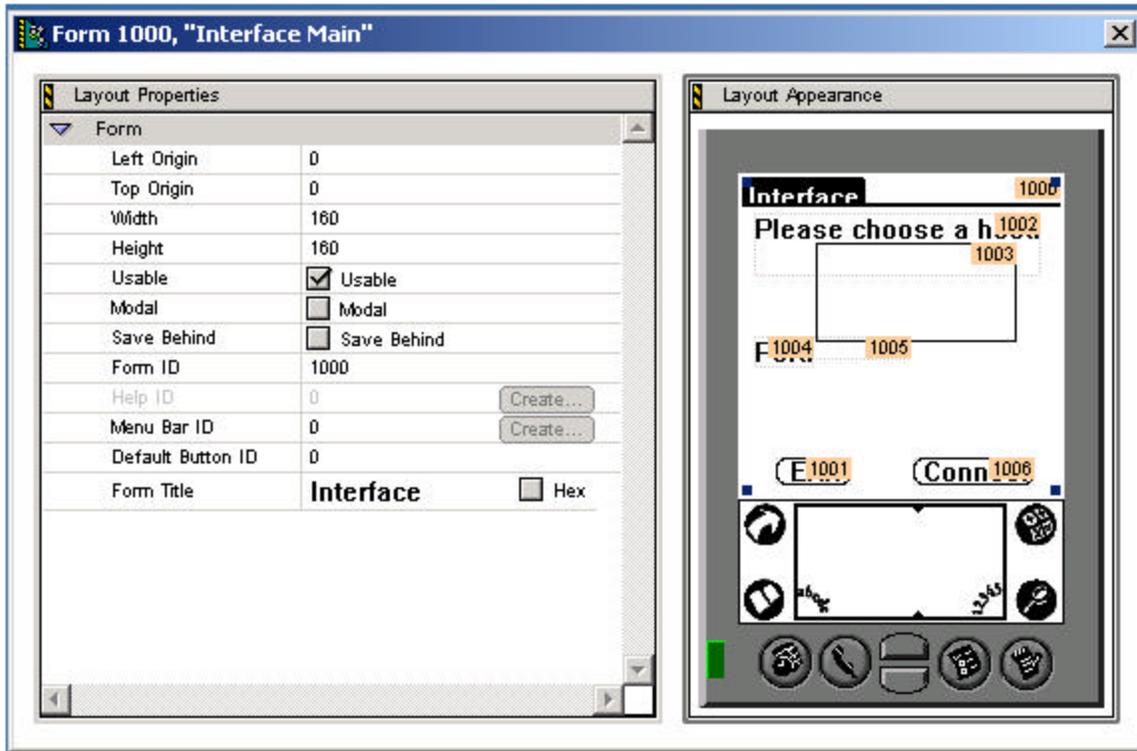


Figure 1.2 This is a screen capture of the CodeWarrior Constructor I used for developing the Palm Interface. Shown here is the *Interface Main*, the main form object.

```
MEMO PAD.c
Path: C:\Program Files\Metrowerks\TOW Lite for Palm OS\Palm OS\WinSDK\apps\Working Folder\Src\

*
* PARAMETERS: fldP - the numeric field (max - 5 digits)
* RETURNED: True once list is initialized, else false
*
*****/

static Boolean InitPort(FieldPtr fldP)
{
    FormPtr fra = FrmGetActiveForm();
    Handle oldTxtH, txtH;

    //Default Port Number Setting

    txtH = MemHandleNew(StrLen(prt) + 1);
    StrCopy(MemHandleLock (txtH), prt);
    MemHandleUnlock(txtH);

    //get field and its current text handle
    fldP = FrmGetObjectPtr(fra, FrmGetObjectIndex(fra, InterfaceMainPortField));
    ErrNonFatalDisplayIf(!fldP, "missing field");
    oldTxtH = FldGetTextHandle(fldP);

    // set the field to the new text
    FldSetTextHandle(fldP, txtH);
    FldDrawField(fldP);

    //free the handle after calling FldSetTextHandle()
    if (oldTxtH)
        MemHandleFree(oldTxtH);

    return true;
}

*****
*
```

Line: 64

Figure 1.3 This is an excerpt from the code. This particular function, *GetPortSettings*, is the source of many of the existing bugs.

Associates, Cambridge.

and

Rochkind, M.J. (1985). Advanced UNIX Programming. Prentice Hall, Englewoods Cliffs, NJ.