A Prototyping Tool for Validating Complex Robot Systems

Cezary ZIELINSKI, Gerald SEET
School of MPE, Nanyang Technological University
Nanyang Avenue, Singapore 639798.
mczielinski@ntu.edu.sg, mgseet@ntu.edu.sg

Abstract
In the development of complex electromechanical systems, e.g. robot systems, it is often necessary to produce breadboard systems to demonstrate the viability of the concept. This, typically, requires that different computer hardware operating with different software to be interconnected. Towards this aim, UDP/IP or TCP/IP networking interfaces provide a useful medium. The paper presents a universal switch that can handle the communication between diverse software modules of any large robot system. The only assumption is made that each component has the capability of using UDP/IP and resides on a node of a real computer network supporting that protocol. The switch has internal data buffering capability, so the speed of data production and consumption may be radically different. It can handle a large number of data producers and consumers. The effectiveness of the solution has been tested on a pilot training and control system for an Underwater Robotics Vehicle (URV).

1 Introduction

Robot systems are usually composed of several layers. The bottom layer consists of electromechanical devices. On top of that is the layer composed of electronic devices responsible for driving the bottom layer and acquiring sensory information both about the state of the electromechanical part and the environment. Currently it is a common practice to compose that layer of several embedded computers and some interfacing electronics. A still higher layer is formed by the control software responsible for the execution of the prescribed task, operator interfacing and generation of control signals for the lower layers. The control software layer itself may be layered too and very often is [1]. Nevertheless the design of complex robot systems always starts with the feasibility study which has to prove that the general concept is valid. In this phase we often use stand-alone computers, instead of embedded ones, running software coded using different programming language platforms or obtained from different vendors, but suiting certain partial tasks. Although each piece of software may perform its functions satisfactorily on its own, it is often very difficult to integrate it into a larger system. The main problems are caused by various speeds of communication between subsystems and the inability of synchronising actions. This paper proposes a general solution to this problem by connecting the computers by a UDP/IP (User Datagram Protocol/Internet Protocol) [2,3,4] network and introducing an extra computer responsible for gathering, storing and supplying upon request the data produced or consumed by all the other nodes of the network. As UDP/IP communication software is available for the majority of real time operating systems and the usually used programming languages (e.g.: C, C++, Pascal), the proposed concept can be used to validate software for very diverse robot systems.

2 Communication Switch

Quite often the software layer of robot devices contains modules that are responsible only for the delivery of data (e.g. sensors). On the other hand there are modules that only consume data (e.g. software generating control signals according to the feedback information obtained from the sensors). The data producers (sources) and consumers (targets) might work with different repetition times. Moreover, it may be difficult to synchronise the sources and targets to exchange the data directly. The proposed communication switch solves all of the above mentioned problems.

The communication switch is a separate computer supervised by a real time operating system QNX [5]. This computer constantly listens at a predefined socket [2] for any incoming UDP/IP datagrams from the data sources. The headers of the received datagrams are analysed to find out who is the target of the message. Subsequently the received data is stored in the internal buffer assigned to the target. When the target needs this information it asks for the data through another socket. The switch then transmits the data to the target through yet another socket. In this way the source can produce data as fast as it can, and the target always receives the latest update. All the communication with the switch uses UDP/IP protocol (fig.1). No assumption is made as to how many sources and targets exist or as to where they reside in the network. Some of them may share the same computers, others might be placed on single network nodes. This depends only on the assumed architecture, the computational load of the software and the properties of the software components used.

The switch is independent of all those factors. Any type of data can be passed between sources and targets, although the header has to be appended by the source.

The switch runs three kinds of processes (fig.2). There is one receiver process that is responsible for listening at a single predefined socket to any incoming datagrams from the sources. Once the datagram is received this process finds out by analysing the header who is the target. This information is necessary to correctly select the buffer process. The buffer process stores the received information. There are as many buffer processes as there are targets. The buffer process constantly listens to messages sent either by the receiver or the transmitter process. There are also as many transmitters as there are targets. The transmitter listens at its private socket for any requests from its target. If such a request is issued it immediately contacts its buffer process. The buffer process passes to it the stored information. The so received data is then transmitted through another socket to the target. All of the internal inter-process communication within the communication switch uses the QNX message passing method (Send - Receive - Reply). All of the external communication is handled by the sockets of the UDP/IP protocol.

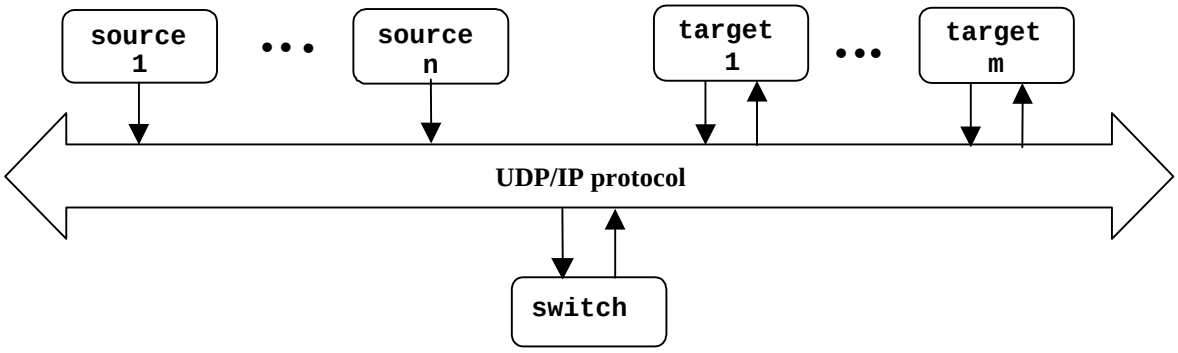

Fig.1 Network Communication of Data Sources And Targets

The receiver and transmitter processes most of their time are read-blocked, waiting for the incoming data or the request for data. The buffer processes are Receive-blocked waiting either for the receiver to pass on the data or an adequate transmitter to request the data. In this way the switch is a pure server in the terms of client-server architecture, while the sources and the targets act as its clients. In this way none of the processes occupies the processor unless it really needs it to execute its task. The processes are brought to life only if they have a task to perform, otherwise they remain dormant. This ensures fastest possible switch reaction times. The switching time is rather limited by the throughput of the network. Keeping the datagrams reasonably short (and in the case of well structured robot systems the amount of data to be passed around is rather limited) all of the data transfers can be treated as atomic actions. The transmission collisions are handled by the Ethrernet protocol, so no data is lost.

3 Utilisation

The switch has been used in a feasibility study of robotic systems created in the Robotics Research Center of Nanyang Technological University, Singapore. The system consists of a dual-purpose control system that can be used for training as well as for the control of an actual URV. The Super Safir URV (fig.3) was developed by HYTEC hydro-Technology. It is designed around a cylinder, which houses all the electronics and a camera module. At one end of the cylinder is a transparent hemispherical viewport behind which the camera is located. At the other end of the cylinder there are the connectors to the motors, lights, compass and the umbilical line. Four thrusters directly attached to the URV body provide the drive. The frame provides protection as well as support for the two floats, two lights and an adjustable float. Wires in a 250m long tether supply from the surface the necessary electrical power. In the training (simulator) mode, the real URV is replaced by a simulator module, which accepts actual commands from the control system and responds with a simulated URV reactions produced by the dynamic model of the URV. The simulator module behaves much like the actual URV.

The control system consists of several components: the real URV controller, URV dynamics model, virtual environment display and operator interface panel. Each of those components is located on a separate computer, being the node of a network. The switch, handling the communication between those nodes, resides on an additional computer (fig.4). The operator commands the real URV or its model by using a joystick (source of commands).

The same joystick supplies the force feedback to the operator (i.e. is the target). Moreover, the operator requires the knowledge of the current state of the URV. The virtual environment is the source of simulated sonar and visual data. This environment has to change its state either in response to the simulation or the real vehicle motion. In simulation mode the URV dynamics model is the source of vehicle responses and the target of operator commands. During normal operation the real URV is the source of data representing the current state of the URV. Moreover, it is the target of operator commands. In this way four sources and four targets have been identified. The switch handles the communication between them.

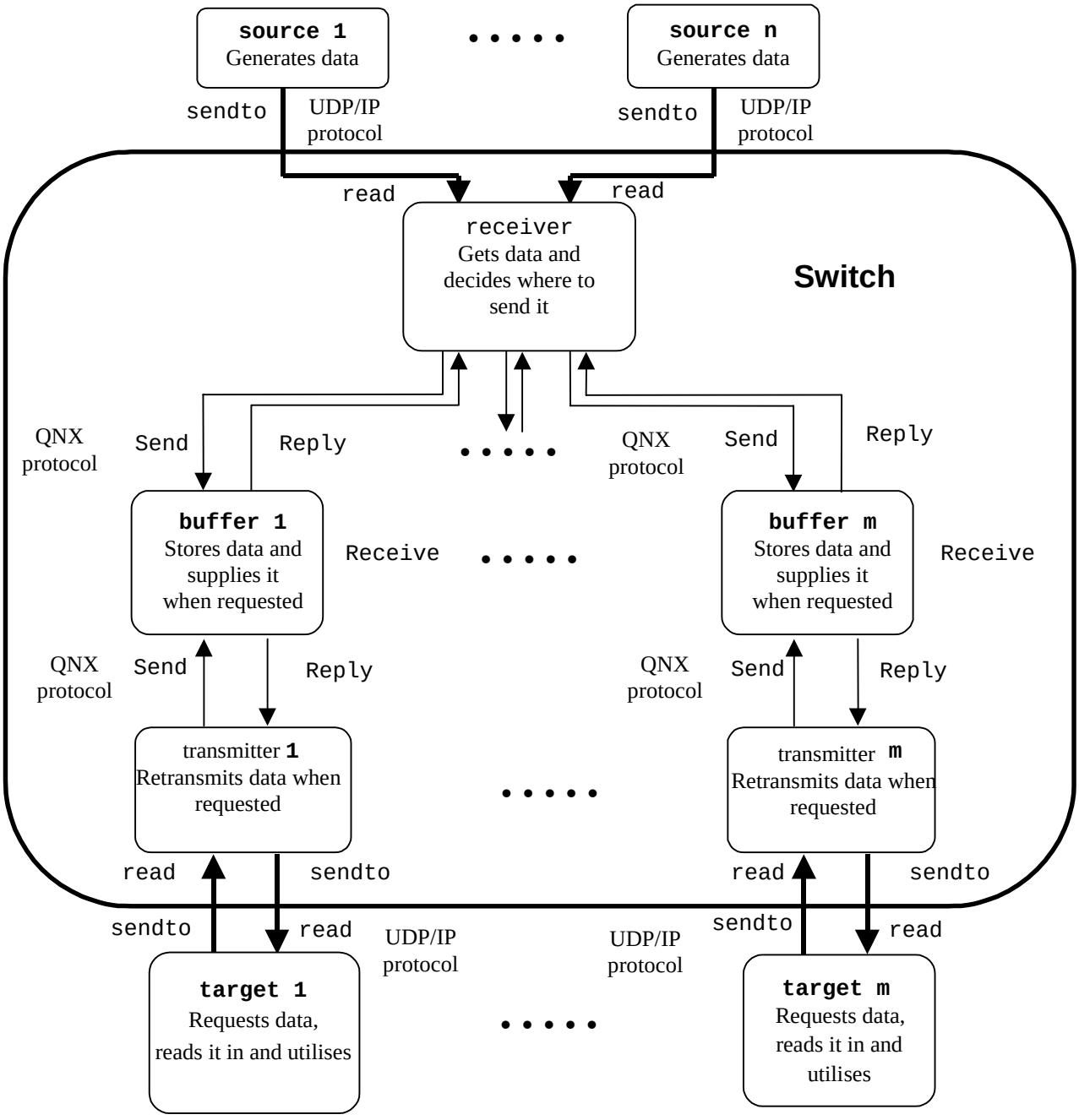

Fig.2 Internal Structure of the Communication Switch

Fig. 3. The Super Safir URV

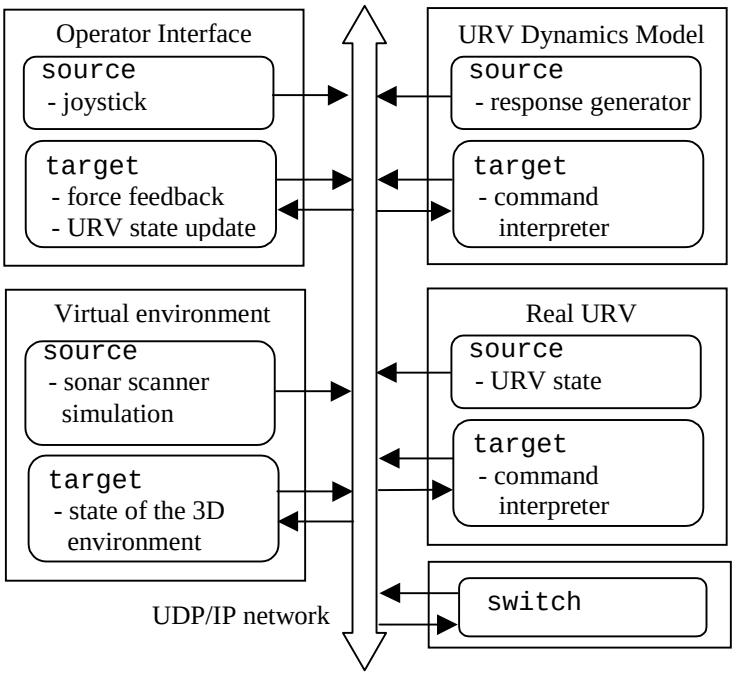

Fig.4. URV control and simulation system structure

Each of the components uses a different software platform for its implementation. In the case of operator interface, instead of hardwired consoles, reconfigurable LabVIEW based panels were developed. LabViEW is a development environment based on the graphical programming language G [6]. WorldToolKit by Sense8 Corporation was used for the development of the virtual environment. A non-immersive system with a 21" CRT display was chosen for presenting the 3D real-time simulation. Stereoscopic viewing was made possible with the use of CrystalEyesVR LCD-shutter glasses that is synchronised with the high-frequency monitor. To further enhance the sense of presence within the virtual environment, both auditory feedback and tactile feedback was incorporated into the system [7]. Sound files may be played in response to events such as collision. Force feedback was achieved with the use of the Microsoft SideWinder force-feedback joystick [8]. Different behaviours may be programmed into the joystick and activated upon request. The force-feedback joystick has two functions. As an input device it accepts motion commands for the URV much like a normal joystick. In the camera control mode, it can also be used to direct the orientation of the pan-tilt mechanism of the camera. In the output mode, the joystick can produce force feedback effects in response to commands from the virtual environment computer. In an effort to improve operator dexterity, the force-feedback joystick can be programmed to be in one of the following modes: vibratory effect for URV collision detection, spring force effect for increasing resistance to motion at higher velocity and damper effect for increasing resistance to motion at higher acceleration.

The WorldToolKit was chosen as the platform on which to develop the virtual environment system. It has a programming library of over 1000 functions written in the C language enabling the programmer to develop high-performance, real-time 3D graphical applications with relative ease. The system was designed for use both in the actual vehicle deployment and as a training aid during mission rehearsal. Thus, it is required that the virtual world closely resembles that of the actual work environment. Huge static objects such as underwater structures were created based on exact dimensions and at precise location (fig.5). Several models were included into the underwater virtual world to create a realistic representation of the actual conditions. These include the environment, the vehicle, and the sensor systems onboard the URV.

Both the graphical and dynamic representation of the vehicle are modelled. In the actual deployment of the URV, the its graphical representation within the virtual world is updated using the sensor feedback from the vehicle. In the training mode the simulation module provides the feedback data.

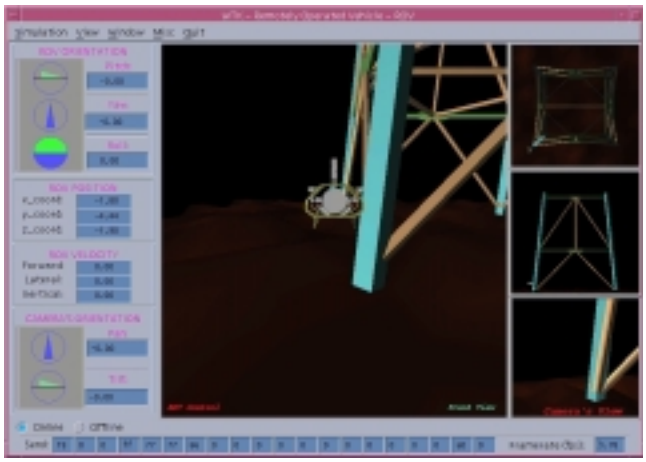

Fig. 5. The Virtual Environment Display

The URV is connected directly to the controller through an RS-232 serial link. The panel provides direct control of the following URV parameters: thruster speed and direction, light intensity and camera movement (pan, tilt, focus, zoom). Included in the panel are a number of indicators providing feedback on the following: camera pan/tilt, URV heading from the onboard magnetic compass, URV depth from the onboard pressure sensor, and fault indicators (high temperature, water and electricity leakage).

4 Conclusions

The software architecture of the communication switch enabling UDP/IP datagram transmission with data buffering has been presented. The switch can communicate with any software using UDP/IP protocol. No assumption has been made as to the number of data sources or targets or the relative speed of data production and consumption. The software of the switch is written in a parametric way using the C language. The switch has as many buffer-transmitter pairs as there are targets. The number of these pairs is a parameter. By changing the value of this parameter and recompiling the source code of the switch a custom designed system tailored to the number of targets is produced. Currently the switch is utilized in the feasibility studies of robotic systems created in the Robotics Research Center of Nanyang Technological University, Singapore. Its efficiency has been tested on a URV controller/simulator system.

5 References

[1] Zielinski C.: Distributed Software for Robots Systems. International Journal of Intelligent Robot Design and Production. Vol.1, No.1, 1994. pp.11-24.

[2] Hunt C.: TCP/IP Network Administration. O'Reilly, Cambridge, 1998

[3] ---: TCP/IP Programmers Guide. QNX Software Systems Ltd. Canada, 1998.

[4] ---: TCP/IP User's Guide. QNX Software Systems Ltd. Canada, 1998.

[5] ---: QNX Operating System - System Architecture. QNX Software Systems Ltd. Canada, 1997.

[6] ---: G Programming Reference Manual, National Instruments, January 1998 Edition

[7] Barfield, W., Furness III, T. A.: 'irtual Environments and Advanced Interface Design, Oxford University Press, New York, 1995.

[8] ---: SideWinder Force feedback SDK – Programmer's Reference, Microsoft Corporation. USA, 1997