DISTRIBUTED SEQUENTIAL
CONTROL IMPLEMENTATION USING
PEER-TO-PEER APPROACH

Chee-Meng Chew¹, Geok-Soon Hong² and Yoke-San Wong³

ABSTRACT

Conventional sequential control systems usually adopt a centralized control approach. This results in a wiring layout that becomes complex as the number of devices increases. A peer-to-peer distributed control approach is proposed and implemented for sequential control. In the developed system, there is a number of cylinder nodes, each of which consists of a pre-programmed micro-controller interfaced to a pneumatic cylinder system through its local input-output (I/O) port. The cylinder nodes are capable of communicating with one another to achieve the overall sequential control task without the need of a central controller node.

INTRODUCTION

Sequential control is usually implemented using programmable controllers (PLCs). Being typically a centralized controller, complex bundles of wires can always be seen in PLC control systems due to the point-to-point connections to all the I/O devices to the PLC (Scholz and McEntee (1992)).

With rapid advancement in VLSI technology and declining cost of integrated circuits, an approach called fieldbus technology has been introduced in recent years (Raji (1994)). Fieldbus technology is basically a distributed system applied to low-level devices such as actuators, sensors etc.. In this relatively new approach, instead of having point-to-point connections between the controller and all the devices, the control is via network communication. The purpose is to reduce the control wiring within and between machines to a single communication medium which connects every device involved in the control. Due to the simpler physical connections between the devices, the hardware installation, maintenance and cable fault-finding tasks for such distributed system can be performed much more easily. The distributed system is also expandable by simply adding nodes physically and logically to the existing system.

There is also a desired trend to distribute the control functions into the input and output devices so that the nodes can intercommunicate without the need of a host or master module (Gibson (1993)). This peer-to-peer distributed network system has one important advantage over the centralized control type in that it does not have a single point of failure.

A modular approach to sequential control of pneumatic cylinders is presented based on a peer-to-peer distributed system. In this approach, each pneumatic cylinder system is interfaced to a micro-controller to form a cylinder node. The micro-controller consists of built-in processors for control and communication. It has a local I/O port that can be interfaced to low-level devices of sensors and actuators. The cylinder nodes communicate to one another through the network to achieve the overall control task (e.g. a predefined sequence of actions). A hardware implementation of the proposed distributed system using a commercial distributed network system called LonWorks™, a distributed system (Echelon (1995, 1995a); Hollignum (1994)) is described.

SEQUENTIAL SYSTEMS

Sequential systems are commonly found in industrial automation. This type of systems can be represented by a discrete state space where changes in state are caused by occurrences of events. Sequential systems can be categorised into asynchronous and synchronous systems (Pesson (1990)). Asynchronous systems are event-based, which means that a control action begins only after the previous control action is successfully completed. Synchronous systems, on the other hand, are time-based, that is, the system is driven by a clock producing pulses at fixed intervals. These pulses trigger the sequence of control actions. The focus is on the design of asynchronous sequential systems which have predefined program paths, either single-path, multi-paths (simultaneous) or a combinations of both.

Pesson (1989) has proposed a method to design the ladder diagrams for sequencing tasks and implementing them using programmable controllers (PLCs) or hard-wired relay circuits. An implementation based on a peer-to-peer distributed approach will be adopted. The approach will be used to perform the sequential control of a number of pneumatic cylinders involving both single and multiple paths. It is assumed that each cylinder system consists of a pneumatic cylinder actuated by a double solenoid control valve and two limit switches, LS 1 and LS 2, used to detect the position of the piston in the pneumatic cylinder (Fig. 1).
DISTRIBUTED SEQUENTIAL CONTROL
SYSTEM DESIGN

Types of distributed system for control

A distributed system is composed of several autonomous intelligent devices which cooperate to achieve an overall goal (Pimentel (1990)). In a distributed system for control applications, the control architecture can be classified into two types (Fig. 2):

- **Centralized control approach** which is a hierarchical approach that includes a central control node to perform the control and event management for the lower level device nodes; and

- **Peer-to-Peer approach** which does not have a central control node for the system. The device nodes simply interact among themselves to achieve a certain task.

In the centralized control distributed system, I/O functions are usually transferred from the central control node to the device node, thus releasing the central control node from I/O processing to perform mainly logic control functions.

In the peer-to-peer strategy, since the system does not have a central control node, the control logics are distributed among the nodes in the network. Each node is designed to react according to the occurrence of certain events (either external or internal). The events may cause the node to perform certain operations and eventually send a message to other nodes. The receivers of the message will then react according to its events handling algorithms.

An architecture for the peer-to-peer approach is presented where the sequential control task is achieved via communication of messages among the actuator nodes. In subsequent sections, the term "distributed system" will be referred to this proposed peer-to-peer system. This system is also a modular system. No customization of the software algorithm is required when the required sequential task is changed.

The concepts and components of the distributed sequential control system

The proposed distributed system consists of the following types of communication nodes: 1) cylinder node and 2) user interface node. Their implementations are described in the following subsubsections.

In the centralized control distributed system, I/O functions are usually transferred from the central control node to the device node, thus releasing the central control node from I/O processing to perform mainly logic control functions.

In the peer-to-peer strategy, since the system does not have a central control node, the control logics are distributed among the nodes in the network. Each node is designed to react according to the occurrence of certain events (either external or internal). The events may cause the node to perform certain operations and eventually send a message to other nodes. The receivers of the message will then react according to its events handling algorithms.

An architecture for the peer-to-peer approach is presented where the sequential control task is achieved via communication of messages among the actuator nodes. In subsequent sections, the term "distributed system" will be referred to this proposed peer-to-peer system. This system is also a modular system. No customization of the software algorithm is required when the required sequential task is changed.

The concepts and components of the distributed sequential control system

The proposed distributed system consists of the following types of communication nodes: 1) cylinder node and 2) user interface node. Their implementations are described in the following subsubsections.
carried out by the object (pointed by the arc). Initially, the START object sends a message (action: to extend) to cylinder object A. Cylinder object A then performs the required action. Upon completion of the action (event: extended), cylinder object A sends a message (action: to extend) to cylinder object B. After cylinder object B has extended, it sends a turn around message to itself so that it will retract. After retracting (event: retracted), cylinder object B next sends a message (action: to retract) to cylinder object A again, causing cylinder A to retract.

For the implementation, the following mechanism is adopted. Each software object consists of a set of discrete control actions as shown in Fig. 5. Each discrete control action is generally used to realise an operation, for example, cylinder extension or cylinder retraction. That operation is executed when a token is received at its entrance. After completing the operation, it passes the token out from its exit (Fig. 6) which can be linked to the entrance of another control action. As such, a series of operations can be executed by pre-linking the control actions.

In the proposed system, a software object called "cylinder object" which resides in the cylinder node is created. This object acts as a virtual device in which one side of the object is interfaced to the pneumatic cylinder system and the other side of the object provides the communication interface to other cylinder objects. The details of the physical interface are transparent to the end-user so that the internal implementation of this object can be encapsulated (Rumbaugh, et al. (1991)). The behaviours and functions of the cylinder object can be understood by looking at the communication interface.

Each cylinder object handles the following types of actions: a) single-stroke and b) reciprocating actions. The single-stroke action can be further divided into the retraction and extension actions of the cylinder. When a token is at the entrance of the extension action object, it causes the microcontroller to send out an output signal to energise the solenoid which causes the cylinder to extend. Once the limit switch at the extended position is activated, it will cause the object to send off the token through the token exit corresponding to that extension action. The retraction action works in the same manner.

The reciprocating action allows the cylinder to perform repeating operations for a predefined number of cycles before calling the next action. When a token is received at the token entrance of the control action, it starts the reciprocating operation of the cylinder until a monitoring counter has reached a preset value. Then, it sends the token off through the control action exit. The number of cycles is a configurable attribute. A cylinder object can communicate either with itself (turn around, as in the cylinder object B shown in Fig. 4) or with the other objects.

The user interface node

This node consists of with three types of software objects: a) user interface, b) timer, and c) ANDing objects (Fig. 7b). The user interface object provides the interface to the physical start/stop buttons for starting and stopping of the sequential control respectively. When the start button is actuated, a token is sent off from this object to start the sequential control process. When the stop button is actuated during the operation, this object generates tokens to all actuator nodes to inhibit the propagation of the tokens in the network thus terminating the sequential control operations.

Time delay is often encountered in sequential control. The timer object is designed to provide such a service. It has a configurable timer attribute which determines the interval of time delay. When a token is received at the token entrance of this object, it starts the timer. It sends the token off through the token exit after the preset time delay.

Some sequential control tasks may have parallel sequential paths. When the termination of these parallel operations is followed by a single operation, the last operation of each parallel path has to be completed before proceeding to that operation in the sequence. The ANDing object is designed to handle such a circumstance. It has a configurable attribute which corresponds to the number of simultaneous converging paths. When each of the parallel operations is completed, the associated control action sends a token to the token entrance of the ANDing object, causing an internal flag to be turned on. When the internal flag is turned on, the ANDing object then sends a token off through its token exit.
Binding process

For the objects described thus far, each has a set of token entrances and a set of token exits. By directing the token exit towards another token entrance, the token flow can be controlled. The process of directing a token exit towards one or more token entrances is called the binding. The binding process logically connects the message channels among the cylinder nodes. Thus, the sequence in the sequential control system can be setup or modified by the binding process.

IMPLEMENTATION OF THE DISTRIBUTED SEQUENTIAL CONTROL SYSTEM USING THE LONWORKSTM SYSTEM

Based on the concept explained in the previous section, the distributed sequential control system has been implemented using a commercial distributed system called the LonWorks system. A description of LonWorks system can be found in Echelon (1995). A workbench consisting of four cylinder nodes is used to demonstrate the operation of the distributed control system. The nodes are connected by twisted pair cable to facilitate communication among the nodes. A network management node (residing on a PC) is used to perform the configuration and binding process. It can be disconnected from the system after each installation.

Introduction of LonWorks system

The LonWorks system consists of intelligent nodes that communicate over a network using a set of communication protocol built into the firmware. In the LonWorks system, the operating system in each node is an event-driven system. Each node includes local processing capability and I/O interface, and can be programmed to respond to a set of events that include I/Os, network, timer events, etc. Network variables (NVs) are used in a node to define the input and output of a node from a network point of view. A node’s object typically consists of a set of user-defined NVs that are specific to the object. These can be linked logically (using the network management node) to the NVs of other nodes for data communication. An output NVs (prefix ‘nvo’) from a node’s object can be bound to one or many input NVs (prefix ‘nvi’) of the other or the same node’s object(s). An output NV will propagate its data if there is an assignment of data to the network variable. When there is an update from an output network variable, the bound input network variables will receive the updated value and specific tasks can be linked to this event. Upon completion of the tasks, the object can send out data through its respective output NVs, which may be bound to input NVs of the same or other node’s object(s). The schematic diagram of a LonWorks node is shown in Fig. 8.

![Cylinder object](image1)

**Figure 9:** Graphical representation of the cylinder object in the cylinder node.

Implementations for the cylinder and the user interface nodes

Two application codes have been developed for the cylinder and user interface nodes respectively. No programming is required for the end-user. The end-user needs only to understand the functional specifications of the nodes’ objects to install any sequence for the cylinders. All the I/O variables within the objects of the node are encapsulated and not shown. The behaviours of an object can be examined by merely looking at its network variables.

The definitions of the network variables in the cylinder object are shown in Table 1. This includes the network variables to handle extraordinary or diagnostic tasks, for example, emergency stop. The graphical representation of the cylinder object is shown in Fig. 9.

![Node’s object](image2)

**Figure 8:** A schematic diagram of a node’s object in LonWorks system.

![User Interface Object](image3)

![Timer Object](image4)

![ANDing object](image5)

**Figure 10:** Graphical representations of the software objects in the User Interface node: (a) the user interface object; (b) the timer object and (c) the ANDing object.

The definitions of the set of network variables in the user interface, timer and ANDing objects are shown in Table 2 and represented graphically in Fig. 10. There are five timer objects in the user interface node. These timer objects can be used by any of the cylinder node objects to include time delay in their actions. Similarly, the user interface node has five ANDing objects to handle concurrency.
Table 1: Network variables in the cylinder object.

<table>
<thead>
<tr>
<th>Network Variable</th>
<th>Direction</th>
<th>Data type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvo_plus[0..n]</td>
<td>Out</td>
<td>Boolean</td>
<td>When cylinder is at the extending state and has reached extended position, this variable will be assigned value '1' and propagated.</td>
</tr>
<tr>
<td>nvo_minus[0..n]</td>
<td>Out</td>
<td>Boolean</td>
<td>When cylinder is at the retracting state and has reached the retracted position, this variable will be assigned value '1' and propagated.</td>
</tr>
<tr>
<td>nvi_plus[0..n]</td>
<td>In</td>
<td>Boolean</td>
<td>When updated, the cylinder extends if the updated value equals to '1'.</td>
</tr>
<tr>
<td>nvi_minus[0..n]</td>
<td>In</td>
<td>Boolean</td>
<td>When updated, the cylinder retracts if the updated value equals to '1'.</td>
</tr>
<tr>
<td>nvi_R_Start</td>
<td>In</td>
<td>Boolean</td>
<td>When updated with value '1', the cylinder will start with the reciprocating motion.</td>
</tr>
<tr>
<td>nci_R_Data</td>
<td>In</td>
<td>unsigned long integer</td>
<td>A configurable network variable to initialize the number of reciprocating cycle for the cylinder before nvo_R_Completed is assigned with value '1'.</td>
</tr>
<tr>
<td>nvo_R_Completed</td>
<td>Out</td>
<td>Boolean</td>
<td>When the cylinder has reciprocated the preset number of times, this network variable will be assigned value '1' and propagated.</td>
</tr>
<tr>
<td>nvi_Stop</td>
<td>In</td>
<td>Boolean</td>
<td>When updated with value '1', it will cause the cylinder node to be at the suspended state. It serves as an emergency stop for the cylinder node and after which, the cylinder node will not respond to any network variables updates until hard reset is carried out.</td>
</tr>
</tbody>
</table>

Figure 11: Logical binding of network variables to achieve the sequence in example one.

'b' The reason of using array type for these network variables will be shown in the example one of this paper. 'n' was chosen to be 4, which corresponds to five set of network variables for single stroke actions of the cylinder.
Table 2: Network variables in the user interface node: a) the user interface object; b) the timer object and; c) the ANDing object.

<table>
<thead>
<tr>
<th>Network Variable</th>
<th>Direction</th>
<th>Data type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvo_Start</td>
<td>Out</td>
<td>Boolean</td>
<td>When the Start button is pressed, this network variable will be assigned a value of ‘1’ and will be propagated to start the sequence. This event is inhibited while the sequential process is in operation (see also nvi_Last).</td>
</tr>
<tr>
<td>nvo_Stop</td>
<td>Out</td>
<td>Boolean</td>
<td>When the Stop button is pressed, this network variable will be assigned a value ‘1’ and propagated to stop the sequence. This event is valid even if the sequential process is in operation. It serves also as an emergency stop to the sequential process. When this network variable is updated and equal to “1”, it causes the inhibition to propagate nvo_Start to be reset, thus allowing the Start button to initiate another cycle of the sequence.</td>
</tr>
<tr>
<td>nvi_Last</td>
<td>In</td>
<td>Boolean</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Variable</th>
<th>Direction</th>
<th>Data type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvo_TimerUp[0..n]c</td>
<td>Out</td>
<td>Boolean</td>
<td>When timer expires, this network variable will be assigned value ‘1’ and propagated.</td>
</tr>
<tr>
<td>nvi_TimerOn[0..n]c</td>
<td>In</td>
<td>Boolean</td>
<td>When receiving an update of value ‘1’, the timer object will start the timer object.</td>
</tr>
<tr>
<td>nci_TimerData[0..n]c</td>
<td>In</td>
<td>unsigned long integer</td>
<td>This network variable is used to configure the timing value, for example, a value of 5 for 5 sec timer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Variable</th>
<th>Direction</th>
<th>Data type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvo_ANDOut[n]d</td>
<td>Out</td>
<td>Boolean</td>
<td>When the internal counter has reach zero (that means all the last nodes of the branches have completed their action), this network variable will be assigned a value of ‘1’ and propagated.</td>
</tr>
<tr>
<td>nvi_ANDIn[n]</td>
<td>In</td>
<td>Boolean</td>
<td>When an update value of ‘1’ is received, the internal counter will reduce by one.</td>
</tr>
<tr>
<td>nci_ANDData[n]</td>
<td>In</td>
<td>integer</td>
<td>This network variable is used to configure the number of branches that will be bound to nvi_ANDIN.</td>
</tr>
</tbody>
</table>

Table 3: Binding table to achieve the sequence in example one.

<table>
<thead>
<tr>
<th>(OUT) Node Name/ &lt;network variable&gt;</th>
<th>(IN) Node Name/ &lt;network variable&gt;</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface/nvo_Start</td>
<td>Cylinder A/nvi_plus[0]</td>
<td>START → A+</td>
</tr>
<tr>
<td>Cylinder A /nvo_plus[0]</td>
<td>Cylinder B /nvi_plus[0]</td>
<td>A+ → B+</td>
</tr>
<tr>
<td>Cylinder B /nvo_plus[0]</td>
<td>Cylinder C /nvi_plus[0]</td>
<td>B+ → C+</td>
</tr>
<tr>
<td>Cylinder C /nvo_plus[0]</td>
<td>Cylinder A /nvi_minus[0]</td>
<td>C+ → A-</td>
</tr>
<tr>
<td>Cylinder A /nvo_plus[1]</td>
<td>Cylinder B /nvi_minus[0]</td>
<td>A+ → B-</td>
</tr>
<tr>
<td>Cylinder B /nvo_minus[0]</td>
<td>Cylinder C /nvi_minus[0]</td>
<td>B- → C-</td>
</tr>
<tr>
<td>Cylinder C /nvo_minus[0]</td>
<td>Cylinder A /nvi_minus[1]</td>
<td>C- → A-</td>
</tr>
<tr>
<td>Cylinder A /nvo_minus[1]</td>
<td>User Interface/nvi_Last</td>
<td>A- → END</td>
</tr>
</tbody>
</table>

c ‘n’ is from 0 to 4, which corresponds to having five timer objects in the user interface node.

d ‘n’ is from 0 to 4, which corresponds to having five ANDing objects in the user interface node.
Procedure for implementing the sequential control of pneumatic cylinders

To achieve a sequential control of a number of cylinders, the network variables are bound in a specific way as demonstrated by the following examples. The binding process can be carried out through the network services of the network management node.

Example one

This example is used to demonstrate the basic principle of the proposed system. The following sequence: START, A+, B+, C+, A-, A+, B-, C-, A- (where A+ denotes “cylinder A extends” and A- denotes “cylinder A retracts”) is to be implemented. The network variables of the cylinder nodes are required to be bound as in Table 3 and Fig. 11. The solid lines in Fig. 11 represent the required binding to achieve the sequence and the detached lines represent the binding for the emergency stop control.

Note that the extension and retraction of cylinder A happen twice in the sequence. In this case, more than one element of the network variables in cylinder object of cylinder node ‘A’ are used to eliminate any ambiguity.

Example two

This example demonstrates the usage of the timer and ANDing objects. The following sequence: “START, A+, (B+, B-) repeated 20 times, C+, B+, 5 sec delay, (      ), C-” is to be implemented. The network variables should be bound as in Table 4 and Fig. 12 to achieve the sequence.

DISCUSSIONS

In the proposed system, each node is well defined before installation and no modification of the application code is required in each node. In this way, the task of reprogramming is eliminated in the implementation of a new sequence. The sequential control can be performed just by binding the relevant network variables. To include an additional cylinder into the system, the corresponding cylinder node is simply connected physically to the communication medium and a new sequence can be achieved by revising the binding of the affected nodes.

Although pneumatic cylinders have many forms of actuation methods, for example, single acting with spring return, double acting etc. (Pessen (1990)), the user does not need to worry about these differences. He simply needs to know that a linear pneumatic cylinder has two basic functions, viz.: extend and retract. The micro-controller can be customised by pre-programming to take care of the actual hardware (I/O devices) differences (encapsulation concept). This can be contrasted with conventional relay logic approach or PLC approach, where the end-user needs to understand the hardware differences before programming (Fig. 13).

In the proposed system, it is important to ensure that all the network messages (for sequential control) sent among the nodes are successful. Any failure in the sending of the network message will cause the system to malfunction since it corresponds to a broken path in the sequence. This is an inherent disadvantage of using the peer-to-peer distributed
system to implement sequential control. So, it is important that the network system used to implement such a system must have reliable communication ability. One way to overcome this is to provide redundancy.

Although the presented implementation is for the sequential control of linear actuators, this concept can be extended to the sequential control of a system which may consist of many concurrent and asynchronous operations which have two discrete states viz.: start of the process (or action) and the completion of the process.

The proposed system does not include the node objects for input devices like discrete sensor devices. The next step in the research is to generate more node objects, such as discrete sensors and fault detectors. For a huge system with many devices, it may be worthwhile to automate the binding of the nodes to further reduce the installation task.

CONCLUSIONS

A peer-to-peer distributed approach for the sequential control of pneumatic cylinders is proposed. Using an experimental setup built on the LonWorks system platform, two examples on the sequential control of pneumatic cylinders have been used to demonstrate the feasibility of using peer-to-peer distributed control approach to implement sequential control. Although the implemented system is applied to pneumatic cylinders, this approach can

### Table 4: Binding table to achieve the sequence in example two.

<table>
<thead>
<tr>
<th>(OUT) Node Name/ &lt;network variable&gt;</th>
<th>(IN) Node Name/ &lt;network variable&gt;</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface/ nvo_Start</td>
<td>A / nvi_plus[0]</td>
<td>START → A+</td>
</tr>
<tr>
<td>A / nvo_plus[0]</td>
<td>B / nvi_R_Start(^e)</td>
<td>A+ → (B+,B-) repeated 20 times</td>
</tr>
<tr>
<td>B / nvo_R_Completed</td>
<td>C / nvi_plus[0]</td>
<td>(B+,B-) repeated 20 times → C+</td>
</tr>
<tr>
<td>C / nvo_plus[0]</td>
<td>B / nvi_plus[0]</td>
<td>C+ → B+</td>
</tr>
<tr>
<td>B / nvo_plus[0]</td>
<td>User Interface/ nvi_TimerOn[0](^f)</td>
<td>B+ → 5 sec delay</td>
</tr>
<tr>
<td>User Interface / nvo_TimerUp[0]</td>
<td>A / nvi_minus[0]</td>
<td>5 sec delay → ( \begin{pmatrix} A - \ B - \end{pmatrix} )</td>
</tr>
<tr>
<td>B / nvo_plus[0]</td>
<td>User / nvi_ANDIN[0](^g)</td>
<td>A- → ANDing</td>
</tr>
<tr>
<td>B / nvo_minus[0]</td>
<td>User / nvi_ANDIN[0]</td>
<td>B- → ANDing</td>
</tr>
<tr>
<td>User Interface/ nvo_ANDOUT[0]</td>
<td>C / nvi_minus[0]</td>
<td>ANDING → C-</td>
</tr>
<tr>
<td>C / nvo_minus[0]</td>
<td>User Interface/ nvi_Last</td>
<td>C- → END</td>
</tr>
</tbody>
</table>

\(^e\) The reciprocating object has to be configured before the operation.
\(^f\) The timer object has to be configured before the operation.
\(^g\) The ANDing object has to be configured before the operation.

Figure 13: Different ladder diagram designs corresponding to pneumatic cylinders controlled by: (a) single-solenoid control valve and (b) double-solenoid control valve respectively.
be adopted for other discrete event system devices, for
examples, motors driving a conveyor to move parts between
two positions, or a pump filling liquid into bottles, which
have two discrete states (on-off). This method can also be
extended to systems which consist of sequential control of
complex tasks, such as, machining and material handling
processes. In these cases, only the "start of process" and "end
of process" states are considered. The peer-to-peer
distributed control approach has the advantages over the
point-to-point centralized control approach in that: 1) it has
lower wiring costs; 2) is easier to expand; and 3) is easier to
maintain. The setup task for the distributed sequential
control has also been shown to be simple in terms of concept
and effort.

ACKNOWLEDGEMENTS
The authors wish to thank Dr. Robert Harrison of the
Loughborough University of Technology, U.K., for initiating
the distributed control project and the setting up of the
LonWorks system during his sabbatical leave at NUS. The
concept and implementation of the distributed sequential
control system have been inspired by him. He has also
provided invaluable feedback and comments.

REFERENCES
29200, Echelon Corporation, USA.
No. 29300, Echelon Corporation, USA.
networks in industrial applications, Advances in
Instrumentation and Control: International Conference
and Exhibition v 48 pt 1 1993, Published by Instrument
Society of America, Research Triangle Pk, NC, USA. 367-
374.
Assembly Automation 14:1, 25-27.
Components, John Wiley & Sons.
Manufacturing, Prentice-Hall.
Raji, R. (1994). Smart networks for control, IEEE Spectrum,
31:6, 49-55.
Rumbaugh, J., Blaha, M., Premerlani, W. and Lorenzen, W.
Hall International.
Electropneumatic Control, Machine Design, 64:18,
38-44.