RSD-TR-14-83

PRODUCTION SYSTEMS FOR MULTI-ROBOT CONTROL

A TUTORIAL

M. C. Maletz

Department of Electrical and Computer Engineering

The University of Michigan

Ann Arbor, Michigan 48109

November 1983

CENTER FOR ROBOTICS AND INTEGRATED MANUFACTURING

Robot Systems Division

COLLEGE OF ENGINEERING

THE UNIVERSITY OF MICHIGAN

ANN ARBOR, MICHIGAN 48109

ABSTRACT

Current robot programming languages belong to the class of procedural systems. They can be characterized by elaborate flow of control, primarily serial processing, and substantial syntactic constraint. Such languages are designed for completely specified tasks, and are poorly suited to the problem of providing robots with a local intelligence capability. In direct contrast, the class of production system languages do not require elaborate control constructs, are highly parallel, and are particularly valuable for implementing heuristically specified tasks.

This paper will introduce production systems and develop a formalism for applying such systems to the area of Multi-Robot Control (MRC). The structure of production systems will permit the construction of MRC systems which are highly parallel and internally conflict-free.

I. INTRODUCTION

Current robot control systems and their programming languages are designed to handle completely specified tasks. They can be characterized by elaborate flow of control, primarily serial processing, and substantial syntactic constraint (e.g., AUTOPASS[4], AL[2], WAVE[5], etc.). Such systems are poorly suited to the problem of providing robots with a local intelligence capability. As the complexity of the task that the system is expected to perform increases, there is a corresponding increase in the complexity of the robot programming and control system design. In direct contrast to current robot programming systems, which belong to the class of procedural systems, is the class of production system languages. This paper introduces production systems, which are highly parallel, do not require elaborate control constructs, and are particularly valuable for implementing heuristically specified tasks.

The application of the production system methodology to the area of multi-robot control develops a formalism for pursuing artificial intelligence research in the robotics area. Among the areas for possible research using the production system formalism are robot synchronization and collision avoidance, the introduction of expert advice, and adaptive modification of the underlying production using genetic algorithms.

Section Two provides a general introduction to production systems. In Section Three, the application of production systems to robot control in multi-robot environments is discussed. This discussion includes the description of a prototype multi-robot environment. The application of production systems to robot control necessitates changes to

the basic production system. These changes are also introduced in Section Three. Section Four contains a detailed example of the way in which the production system-based multi-robot control system functions. Finally, Section Five presents conclusions and areas for further research.

II. INTRODUCTION TO PRODUCTION SYSTEMS

Two classes of programming languages can be identified: procedural systems and production systems. Almost all traditional programming languages are procedural in nature. They are characterized by the importance of flow of control, which is often elaborate, and by an algorithmic specification of the tasks to be undertaken. Procedural systems are generally processed serially, although a degree of parallel processing is occasionally permitted (such as tasking). This parallelism, however, must be carefully specified and is subject to substantial syntactic constraint. Even where parallelism is permitted, flow of control concerns still dominate. In direct contrast, is the class of production systems, which are highly parallel, lack elaborate control constructs, and are particularly valuable for implementing heuristically specified tasks.

Traditional programming languages are heavily dependent on syntax requirements. A change of even a single bit is likely to prevent the program from functioning correctly, perhaps even from functioning at all. This dependence on syntax makes automatic programming and the implementation of adaptive algorithms particularly difficult. The highly parallel nature of production systems makes their "instructions" less interactive than those of traditional programming languages. This results in the graceful degradation property which characterizes production systems and permits the alteration of individual "instructions" without significantly degrading system performance.

A production system can be completely specified by a set of productions and a common data structure known as a message list as follows:

production system ::= {production_set, $message\_list_m$}
production_set ::= <production>$^+$
<production> ::= <condition> /<action>
<condition> ε $\{0,1,*\}^n$ | '$\{0,1,*\}^n$
<action> ε $\{0,1\}^n$
$message\_list_m$ is a data structure that can contain a maximum of m messages, where each message is an n-bit string over {0,1}

Each production in the production_set contains a condition part and an action part. The + superscript indicates that

the superscripted element may appear one or more times. This means that productions are composed of one or more conditions comprising the condition part, followed by exactly one action in the action part. Conditions are n-bit strings over the {0,1,*} alphabet, where 0 and 1 are the ordinary binary digits, and * is a don't care character (i.e., it will match both a 0 and a 1). The actions are n-bit strings over the {0,1} alphabet. The relative simplicity of the message list belies a great versatility which will be developed.

Given the above specification, a production system functions as follows:

1. The system is first initialized by placing one or more messages on the message list (up to a maximum of m messages).

2. Each of the productions is checked against the message list to determine whether the messages on the message list satisfy that production's conditions.

   A production's conditions are **satisfied** if and only if every condition matches a message on the message list, where matching is done on a bit-by-bit basis. The matching process is performed bit-wise for every condition in the condition part of the production, and therefore, the condition part is treated as a conjunction of conditions. Only the 0 and 1 digits in a condition must match the corresponding digit in a string on the message list since the don't care character (*) will match either a 0 or a 1 digit.

3. After every production has been checked against the message list, the message list is erased. Notice that the production checking procedure can be performed in parallel if the hardware permits. In fact, each production could potentially be processed by a dedicated microprocessor. Because of the message list erasure at the end of each time step, all messages have a single time step lifespan. To post messages for longer periods, repeated postings are required. A simple message renewal construct involves the use of a production whose condition and action parts are identically equal to the message to be renewed. This results in perpetual message renewal for the associated message once it appears on the message list, unless the production is not selected to post its message because more

than m productions with distinct messages attempt to post their messages to the message list. A more general approach would be to use additional conditions in the condition part of the production to indicate when message renewal should occur. Using this approach, when the message to be renewed appears on the message list it will match one of the conditions, and when the other conditions are matched signalling message renewal, the message is re-posted.

4. A new message list is then constructed from the action parts of every production that was satisfied by the old message list. As long as no more than m productions with distinct messages are satisfied in step 3, every production can post its action part to the new message list.

> The action part of a production is **posted** to the message list by placing the n-length bit string that comprises the action part on the message list.

5. The processing procedure is iterated by returning to step 2.

As long as the number of distinct messages to be placed on the new message list in step 4 is not greater than m, all productions whose conditions are satisfied by the old message list may post their actions on the new message list. This results in a parallel system that is internally conflict free due to the fact that messages, from the message list viewpoint, are simply bit strings and cannot conflict with one another. Conflict can occur when the messages are interpreted if they generate contradictory interpretations (e.g., two messages might generate two robot control commands, one which instructs the robot to move to the right and the other which requires a left rotation). Interpretation conflict resolution is, however, exogenous to the production system. Another type of conflict is due to the fixed length message list. If more than m productions attempt to post distinct messages to the new message list, then m of the new messages must be selected. This problem can be resolved by removing the length constraint from the message list. This could be done by setting the message list length (m) sufficiently large, since the system itself imposes a maximum message list length equal to the number of distinct actions in the production set.

The production system processing can continue indefinitely. There are three possible terminal conditions:

1. During some iteration, it is possible that no productions will be satisfied by the current

message list. This results in no messages being posted to the new message list. Once an empty message list appears, the system stops processing.

2. A message list cycle may appear. This represents a type of dynamic equilibrium in which the system cycles through a set of message lists. It corresponds to the control of repetitive tasks.

3. A special case of message list cycling occurs when the cycle length is equal to one. In this case, the same message list appears during all future time steps.

An empty message list brings the system to a halt and should be avoided. Other control constructs for halting the system can be provided if necessary. A simple mechanism that will prevent an empty message list from occurring is the inclusion of a set of productions in the production_set that covers the n-bit string space over {0,1}.

> A <u>cover</u> of $\{0,1\}^n$, the n-bit string space over {0,1}, is a collection of strings in $\{0,1,*\}^n$ such that every string in $\{0,1\}^n$ is contained in a string in the cover. A string in $\{0,1\}^n$ is contained in a string in $\{0,1,*\}^n$ if and only if the 0 and 1 digits in the $\{0,1,*\}^n$ string exactly match the corresponding digits in the $\{0,1\}^n$ string.

A simple example of this would be the use of a production whose condition consisted of the string **...*, which would match any string on the message list. This universal condition is particularly useful in systems where the message list is a limited resource and not all productions whose conditions are satisfied can post their actions to the new message list. In such systems, the universal condition production could be assigned a low priority so that it would only post its action to the new message list if there were few other satisfied productions during the current time step. In a system where all satisfied productions are guaranteed a space on the new message list for their actions, the use of universal condition productions is discouraged since their actions would perpetually appear on the message list. In such systems, it is better to use a larger set of productions that cover the n-bit string space over {0,1}. The universal condition production is the smallest cover. The largest possible cover contains 2^n productions, where the condition part of each production is a distinct string in $\{0,1\}^n$.

A message list cycle can be beneficial in systems which are used to control cyclic or repetitive tasks. The steps of

the task cycle would then correspond to the cycles of the message lists. The single time step cycle is a special, and generally degenerate, case of these cycles. Although cycles can be programmed using production systems, they are better handled by procedural programming systems because of the algorithmic nature of cycles and the advantages inherent in procedural systems for algorithmic programming.

The don't care (*) character has been permitted to appear in the condition part of productions, but not in the action part. A don't care character in the action part of a production corresponds to a pass-through character. A pass-through character can only be used with productions whose condition part consists of a single condition.

> A **pass-through** on the action side of a production involves passing the corresponding bit of the message that matched the condition through to the action that is posted. If the corresponding character in the production's condition is a 0 or 1, then that digit is always passed-through. In such situations, it is easier to simply place that digit in the action string rather than the pass-through character. More often, the condition character corresponding to a pass-through is the don't care character. In this case, the corresponding character in the message that matches the condition will be passed through. Recall that the don't care character can match either a 0 or a 1 digit in a message. The pass-through character will not be permitted in productions because of computational difficulties that result. The most prominent difficulty is the resolution of a potential conflict when more than one message matches the condition of a production containing pass-throughs in the action part. Consider the following example:
>
> > The production involved is 01*11/11*10. Suppose that both of the messages that can match the condition part of this production appear on the message list (01011 and 01111). The pass-through would then attempt to pass through both the 0 and the 1 digits, and to post both 01010 and 01110. This violates the single action requirement

imposed on the action part of productions.

Rather than introduce this potential conflict, the pass-through character will be excluded from the action parts of productions. This does not limit the computational power of the production system, since the action part pass-through can be emulated by using more than one production. This emulation requires the replacement of don't care characters in the condition part by both 0 and 1 digits, and replacement of the action part pass-through by the same digits.

Greater flexibility and programming power can be attained in production systems if communication with the outside world is permitted. This communication must be introduced without disturbing the parallelism or internal conflict-freeness of the production system. This is most easily accomplished using the message list by permitting messages to be posted from the outside world. The outside world can also be permitted to read the message list. In this way, production systems can be provided with a primitive I/O capability.

III. ROBOT CONTROL USING PRODUCTION SYSTEMS

The application of production systems to the problem of robot control in a multi-robot environment will now be discussed. The robots in these multi-robot control (MRC) systems will initially consist of a microprocessor-based robot arm, a television camera or other imaging device and an associated image processor, and a locomotion drive unit. Alternate robot configurations are possible with relatively minor alterations to the MRC system command structure. No modification of the production system processing is required when the robot configuration is altered. The robots are able to interact with their environment using the robot arm which can execute a predefined set of control subroutines, view portions of the environment, and move through the environment. Associated with each robot is a robot control processor which is responsible for local processing. Each individual robot processor communicates with its respective robot and with the system supervisor.

An example configuration is presented in Figure 1. The robots consist of PUMA robot arms mounted on a locomotion platform along with a television camera which can be rotated to the left or right, but which is set at a fixed height. The robot processors are LSI 11/23 microcomputers. Each LSI 11/23 is connected to the associated robot and to the supervisory computer, a DEC VAX 11/780. Two channels from the VAX 11/780 to each robot processor are required. Therefore, the number of robots that can operate within the

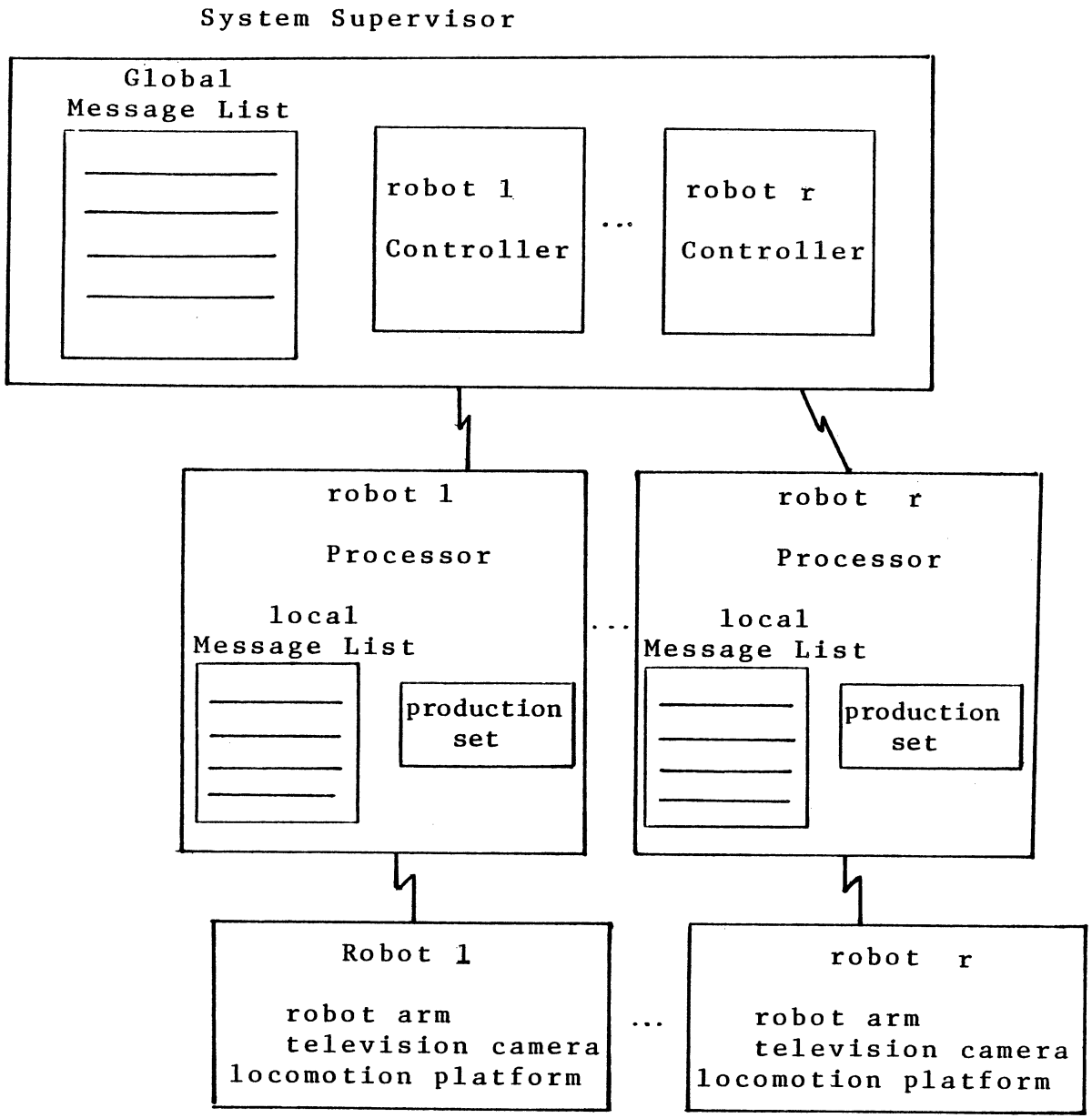


Figure 1. Example MRC System Configuration

system is entirely determined by the number of available channels to the VAX 11/780.

Bidirectional communication between robot and robot processor permits robot control and television camera data transfer. The television camera has a local dedicated processor which converts images within the scope of its view into messages which can be placed on the message list. These messages are transferred along the communication channel to the robot processor. No other data originates at the robot end of the channel. Again, alternate robot configurations could be constructed which would require more extensive robot/robot processor data transfer. One example would be data generated by a wrist force sensor or other sensory devices. As with any alteration to the robot configuration, these changes would require MRC system modification. The robot processor sends robot control instructions to the robot. Initially, a limited set of robot control commands (RCC's) will be permitted. These initial commands are summarized in Table 1. Although relatively few commands are available, the set of Execute commands {En} permit the execution of any one of N robot control subroutines. These pre-programmed subroutines could be designed using task-specific robot control requirements. They therefore provide a large degree of flexibility in specifying the robot control capabilities of the system.

Table 1. Robot Control Commands

Command		Description
{S}	SCAN	Vision vector performs 360° scan
{A}	ALIGN	Align vision vector with nearest object
{T}	TURN	Rotate vis. vec. 180°
{L}	LEFT	Move vis. vec. left ε°
{R}	RIGHT	Move vis. vec. right ε
{M}	MOVE	Move in the direction of the motion vector
{F}	FAST	Move Quickly in the motion vector's direction
{H}	HALT	Stop moving
{C}	COUPLE	Couple motion vector with vision vector
{U}	UNCOUPLE	Uncouple motion vector from vision vector
{G}	GRASP	Grasp object aligned in vision vector
{En}	EXECUTE	Execute robot arm subroutine n, for $n \leq N$

The set of RCC's has been divided into three general categories. The first five commands relate specifically to the vision vector. The next five commands are used to control the robot's locomotion, and the last two commands are used to manipulate the robot arm.

Implicit in Table 1 are several assumptions concerning the MRC system:

1. Each robot is equipped with a single television camera which corresponds to the vision vector, a robot arm which can grasp objects centered by the vision vector, and a primitive locomotion capacity which must be in the direction of the vision vector.

2. Robots move at a fixed speed in the direction of the vision vector. The motion vector must always be coupled with the vision vector before movement. The vision vector may be rotated during robot movements, but this will alter the robot's course.

3. The vision vector may only be uncoupled from the motion vector after the robot has been halted. This permits the robot to scan the environment by moving its camera while leaving its locomotion platform fixed. Because of this requirement, the SCAN RCC can only be performed after the robot halts.

4. The robot can grasp an object that has been aligned in the vision vector. A primitive subroutine execution facility {En} has been included in the RCC set that can be used to code robot arm subroutines, in addition to the other robot control subroutines that {En} may encode. Robot arm control EXECUTE RCC's may be performed either before or after grasping an object. They are presumably related to the task that the robot is to perform. The availability of multiple robot arm subroutines using more than one {En} RCC corresponds to multi-task capabilities.

Two additional RCC's are provided for use by the system's "expert" users (the individuals who provide "expert" advice to the production system):

SHOW R	Show the portion of the environment contained in robot R's vision vector on the television monitor.
IMAGE R	Show all image related messages

posted to the message list by robot R during the current time step. These messages are created by the image processor associated with the robot's television camera.

These commands will never be invoked by the production system. They are used exclusively by "expert" users to help them understand the internal processing of outside world image data by the production system.

3.1 Production System Modification

The class of production systems specified by { production_set, message_list$_m$ } can be used to control a MRC system, but only if robots have complete autonomy or no autonomy. In the case of complete autonomy, each robot would have its own production system and would never need to communicate with other robots. In the no autonomy case, all processing and control would be performed by a single production system which would reside at the supervisor level. Clearly, both of these options are inefficient. Because the robots operate in the same environment, some implicit communication and cooperation among the robots is desirable. However, because the robot tasks may be vastly divergent, centralized control is inappropriate. This indicates the need for a modification of the production system definition.

To isolate robot control, each robot should be equipped with its own production_set. This will permit differentiation of robots according to task and behavior specific requirements through the distinct production_sets. To permit implicit communication and cooperation, a single message list will serve as the common data structure. In terms of the example system configuration, this means that each local robot processor (LSI 11/23) will have a production_set which is processed and updated separately from all other robot's production_sets. This will permit each robot's production_set to evolve separately when adaptation is introduced. The single message list for the MRC system will reside on the system supervisor (VAX 11/780), and be accessible to all robot processors. These modifications to the production system definition are sufficient to permit control of a MRC system. The new production system is specified by the following r+1-tuple:

production system ::= { production_set$_1$,
production_set$_2$, ...,
production_set$_r$,
message_list$_m$ }

Where there are r robots in the multi-robot environment.

Figure 2 illustrates the example MRC system, including the locations of production_sets and the message list.

3.2 Message Differentiation Through Tagging

In the multi-robot environment, each robot is characterized by its individual production_set. These production_sets are processed using the common message list during each time step. The shared access to the message list necessitates segmentation of the message list. The message list segments correspond to the various interpretations of messages on the message list, and are summarized as follows:

1. Messages can be generated externally to provide the MRC system with "outside world" information. In the example system, these messages can only be generated by the television camera image processor. The messages are therefore specific to the robot whose television camera generated them. For this reason, other robots need not use these messages in production processing. Future MRC systems may include an external source of outside world data. This external source could then make its data available to all robots. An example of this would be a motion detection device which could identify all robot's that were in motion.

2. Internal processing messages can be generated by the activation of a production belonging to a particular robot for processing associated uniquely with that robot. These are the messages that permit each robot's production processing to occur, and need not be interpreted by other robots.

3. Control messages can be generated internally by production activation for a particular robot. These messages are used both for production system processing and for robot control. The robot control function is implemented by translating the message into a robot control command, which is then passed to the robot for interpretation. These control messages are also robot specific, and need not be interpreted by other robots.

4. Coordination messages are accessed by all robot production_sets and are used to permit robot coordination and joint processing capabilities.

Figure 3 illustrates the segmentation of the message list into these four segments, and describes the degree of shared access to each segment among robot processors. This message

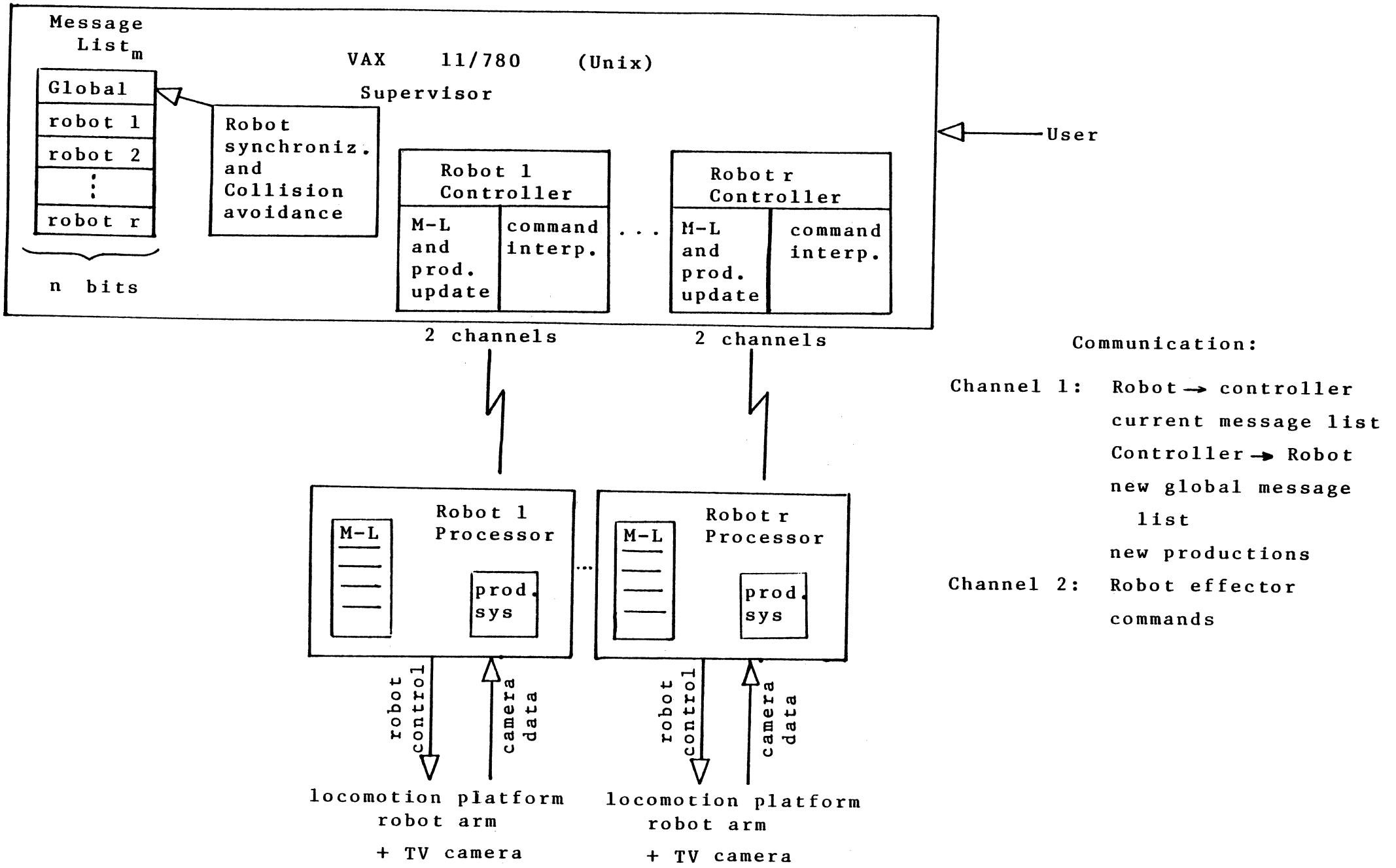
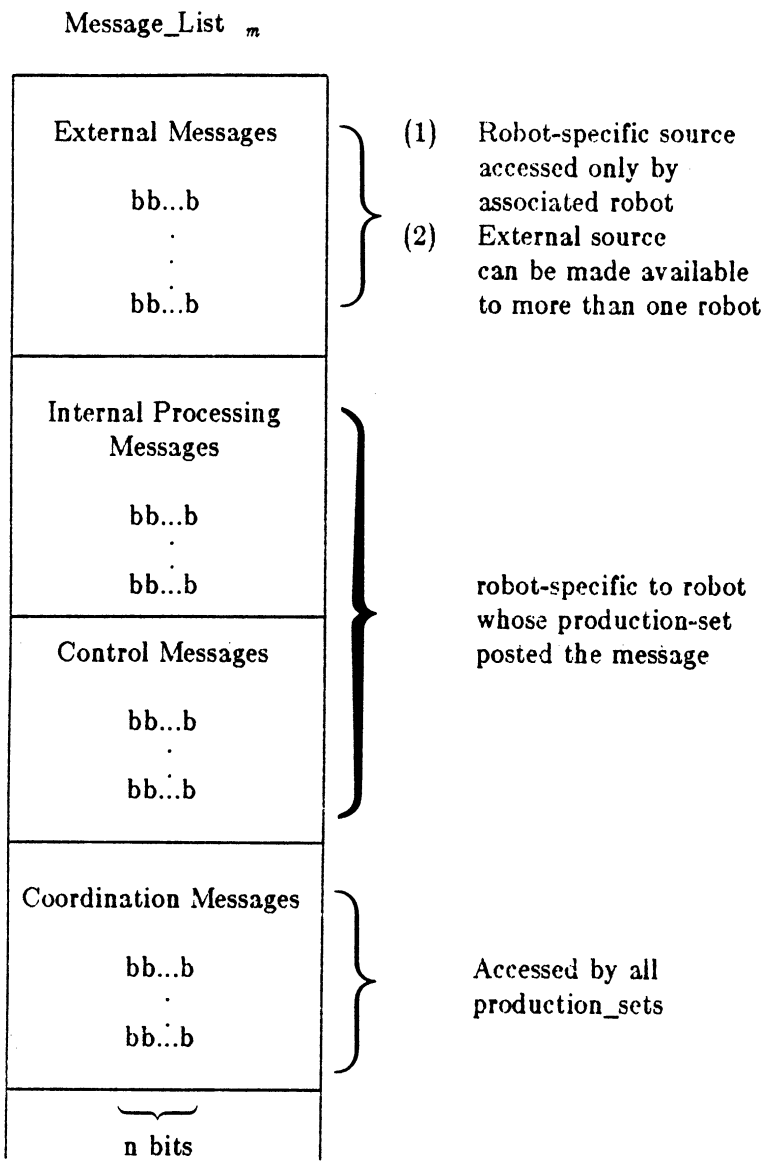


Figure 2. Example MRC System with Production_Sets and the Message List



where $b = bit$

$bb...b$ is a fixed length bit string (n bits)

Figure 3. Four Segment Message-List Segmentation

list segmentation will be further refined for MRC system usage. The first category of messages provides the production system with a primitive input capability. These are the messages that are generated by the image processor associated with each robot's television camera. Encoded in these messages is information concerning the image currently contained in the vision cone. The third message category provides the production system with a primitive output capability, in the form of robot control commands which are executed by the robot. These commands permit the robot to interact with its environment. The final message category represents the potential for I/O among robots in the form of messages which are accessed by all robots.

Segmentation of the message list into the four categories of messages is accomplished through the use of message tags.

> A **tag** can either be a prefix or a suffix consisting of a fixed number of bits. These bits can be used to identify message list segments.

In this case, prefix tags will be used. The number of bits dedicated to the tag depends on the number of distinct message list segments desired. By subdividing the prefix bits into fixed length substrings, it is possible to provide sub-segments for the message list. In this way, the message list may first be segmented into principal components, and then each of these components may be further subdivided as necessary. For the MRC system, primary message list segmentation will be used to separate the global portion of the message list and the robot-specific segments. The global segment of the message list contains messages from category four. These messages are required for robot coordination, and must be accessible to all robot processors. Each robot-specific segment of the message list contains external messages, internal processing messages, and control messages for the associated robot. If there are r robots, then the message list must have r+1 segments, which requires INT_CEILING $[\log_2(r+1)]$ bits dedicated to the tag prefix (where INT_CEILING [x] is the integer ceiling function; the largest integer greater than or equal to x).

No further subdivision is required within the global portion of the message list. The robot-specific segments, however, must be further divided into three sub-segments according to message category. This further segmentation requires implicit segmentation of the tags.

> Each tag used by the production system will be divided into two components: a **primary tag** and a **secondary tag**. The **primary tag** is used to segment the message list into its primary segments; global and

robot-specific. This requires r+1 distinct tags, and hence INT_CEILING $[\log_2(r+1)]$ bits will be dedicated to the primary tag. The primary tag will always be represented by the initial bits in a message. A **secondary tag** is used to establish message list sub-segments, within segments defined by the primary tag. The number of bits required for the secondary tag are computed in the same way as the number of bits required for the primary tag. The secondary tag bits immediately follow the bits dedicated to the primary tag. While only primary and secondary tags will be used here, higher order tags can also be defined.

The first sub-segment of the robot-specific message list segments identifies external messages. These messages correspond to images in the vision cone and can therefore be interpreted as more than just a sequence of bits for production system control. The second sub-segment contains internal processing messages. The third sub-segment contains control messages which can be converted into robot control commands and transferred to the robot. These control messages are also used by the production system for internal processing along with internal processing messages. By distinguishing between internal processing messages and control messages, a substantial amount of command conversion overhead is eliminated, since the command conversion routine need only process messages tagged as control messages.

The following tagging convention will be adopted:

Suppose that there are r robots in the MRC system, and that the robots have been indexed 1 to r. i = INT_CEILING $[\log_2(r+1)]$ bits are required by the primary tag, and 2 bits are required by the secondary tag.

1. The global segment of the message list will always be assigned the tag: 00...0 (i+2 bits).

2. Each robot-specific segment of the message list will be assigned an i-bit primary tag whose decimal value is equal to the index number of the associated robot.

3. The 2 secondary tag bits will immediately follow the primary tag bits and will be encoded as follows:

00	External messages
01	Internal processing messages

10 Control messages
11 Not currently used

The 11 secondary tag could be used for global messages posted by a particular robot. This could, for example, be used if image data collected by a particular robot was required by other robot processors. Figure 4 displays the complete message list segmentation used by the MRC system. This tagging scheme segments the message list in a manner that requires little system overhead. Distinct message list segments can be identified by using a bit mask to select the messages of interest. As an example, a bit mask of 00...011 (i bits) will select all messages specific to robot number 3, and a mask of 00...01110 (i+2 bits) will select all control messages for robot number 3.

Tag manipulation using bit masks and assembly language is fast and efficient. Tags represent a low overhead mechanism for partitioning the MRC system's common data structure into both shared and dedicated components. In future applications, tags can be used to perform even more sophisticated functions. One example involves message addressing using tags. In such a system, robot j might post a message on the message list, and address that message to robot k through the use of a tag. This tag would have two address components, one for the sender, and the other for the addressee. Although the message list is a relatively simple data structure, information coding using tags can permit sophisticated behavior, and the emulation of complex data structures.

IV. MRC System Example

The example system will be configured with a single robot, consisting of a robot arm, a television camera, and a locomotion platform. The robot will reside in a contrived environment in which there are humans, ducks, rats, and a single elephant (Clyde, of course). The robot's desired behavior is to avoid the elephant, chase the rats, and ignore the ducks and humans.

The simplicity of the environment and task results in a relatively simple production system. The external messages generated by the television camera's image processor need only distinguish the four types of occupants of the contrived environment. This can be accomplished using two features: size and number of legs. A third feature which will prove to be useful is the range of the object, since the robot will react differently to the elephant depending on whether it is close or far away. The three features can be coded as follows:

size 0=large 1=small

# legs	0=2 legs	1=4 legs
range	0=close	1=far

Using these features, the external messages corresponding to the four residents of the robot's environment can be constructed. Each of these messages requires five bits: the two bit 00 prefix identifies the message as an external message, and the last three bits correspond to the size, number of legs, and range features, respectively. The area inhabitants are identified as follows:

Humans	0000_
Ducks	0010_
Rats	0011_
Elephant	0001_

Note that in each case, the range feature is left undetermined, since each of the area inhabitants can be either close or far away.

To obtain the desired behavior from the robot, a subset of the Robot Control Commands is sufficient. These RCC's are coded as control messages using the control message 10 prefix as follows:

10000	Halt
10001	Rotate vision vector ε°
10010	Rotate vision vector 180°
10011	Move fast
10100	Align vision vector with nearest object

For this example, five RCC's were sufficient, and since the external messages required three bits for feature mapping, the RCC's could be coded using the message length determined by the external messages. In fact, one could have coded up to eight RCC's using the five bit message format. If more than eight RCC's were required, then it would have been necessary to expand the message length, and to pad the external messages with dummy features.

Because of the task's simplicity, a rough approximation of the desired behavior can be obtained using only seven productions:

00*0*/10001	Ignore humans and ducks by rotating the vision vector
00011/10000	Halt when the elephant is sighted at a distance
10000/10001	Rotate vision vector away from elephant after halting
00010/10010	Rotate vision vector 180°

	when elephant is close
10010/10011	Move fast to flee from close elephant
0011*/10100	Align vision vector on rats
10100/10011	Move fast to chase rats

The above production system fragment could be expanded to provide the exact behavior desired, but it is intended only to serve as an example and to help develop intuition about production systems. A particularly significant observation concerns the third and fifth productions. In both of these productions, a control message serves as an internal processing message. In the second production, the 10000 control message is posted to the message list and is interpreted as an RCC to halt the robot. In the third production, the 10000 control message serves as a condition to the production. The 180° rotation RCC (10010) found in the fourth and fifth productions functions in a similar manner.

V. Conclusions

A formalism for applying production systems to the area of Multi-Robot Control has been developed. The example system demonstrated the use of such a system in a single robot environment. More sophisticated multi-robot environments can be developed, in which the robots perform robot-specific tasks and also communicate with each other through the global message list.

In the future, the formalism developed here will be applied to several problems in the area of robot control:

1. Robot Synchronization - the fact that all robots access the global message list can be used by the system supervisor to handle such robot synchronization problems as collision avoidance. One simple approach to this problem would be to divide the environment into quadrants, and use the global message list to Lock and Unlock these quadrants so that only one robot would be permitted to be in motion in any quadrant.

2. Production Differentiation - assuming that the production system does not completely correspond with the desired behavior (either due to faulty coding, inaccurately specified behavior, or changes in the task itself) it is helpful to identify productions that contribute positively and negatively to the robot's completion of the

specified task. This will involve assigning a strength parameter to each production, and modifying the strength based on the production's contribution to robot behavior.

3. Adaptive Modification of Production System - once the productions have been assigned strength parameters, it is possible to use genetic algorithms to modify the production system so that the robot's behavior will converge to the desired task.

4. Introduction of Expert Advice - expert advice is required to differentiate productions. It can also be used in conjunction with genetic algorithms to modify the task assigned to the robot.

These areas of future research will involve the development of simulations of the MRC systems, and then prototype systems using actual robots with simulated input data. Finally, a prototype system in which the robots and the television cameras are fully operative will be constructed.

Acknowledgments. The author is grateful to C.S. George Lee for his comments and advice during the development of this paper.

References

{1} Bonner, S., and Shin, K.G. A Comparative Study of Robot Languages, *Computer*, 15,12 (December 1982), pp. 82-96.

{2} Finkel, R. AL, A Programming System for Automation, Stanford Artificial Intelligence Laboratory Memo AIM-243, Stanford University, Palo Alto, California, November 1974.

{3} Holland, J.H. *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Michigan, 1975.

{4} Lieberman, L.I., and Wesley, M.A. AUTOPASS, An Automatic Programming System for Computer Controlled Mechanical Assembly, Computer Sciences Department RC 5925 (25653), T.J. Watson Research Center, Yorktown Heights, New York, March 1976.

{5} Paul , R.P.C. Wave: A Model-based Language for Manipulator Control, *Industrial Robot*, vol. 4 (1977), pp. 10-17.

{6} Takase, K., Paul, R.P., and Berg, E.J. A Structured Approach to Robot Programming and Teaching, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11,4 (April 1981), pp. 274-289.

{7} Winston, P.H. *Artificial Intelligence*, Addison-Wesley, Massachusetts, 1977.