ON THE USEFULNESS OF SLAM-II FOR THE MODELING AND SIMULATION OF LARGE TRANSPORT SYSTEMS

by: Andre Graber & Francois E. Cellier
Institute for Automatic Control
The Swiss Federal Institute of Technology Zurich
ETH - Zentrum
CH-8092 Zurich
Switzerland

Since 1980, the new network-oriented simulation language SLAM, primarily useful for discrete system simulation but also applicable to combined continuous and discrete system simulation, is available, and since then has been successfully applied by many Research Groups to a large variety of simulation problems. This software system which was developed by Pritsker & Associates, Inc. certainly is a mile stone in topological simulation software development.

This paper discusses a very large industrial application of SLAM for the simulation of internal transport systems. The aim of this paper is to critically analyse the aptitude of SLAM for the modeling of such types of systems, to show its advantages, and to outline its limitations. By these means, we hope to provide future potential users of SLAM with a decision aid for software selection criteria.

1. INTRODUCTION

This paper describes briefly the modeling of a large internal transport system for stock management which was undertaken for Digitron AG, a Swiss Company producing robot trailers and automated internal transport systems making use of these robot trailers. The particular application, which was studied by us using the SLAM-II simulation software, concerned a large stock management system in which approximately a dozen robot trailers traveled between a delivery conveyor, 8 fully automated high bay ware houses, 6 manned picking stations, a manned inventory and quality control station, and the collecting station from where the stored goods are sent further.

The main objective of this study was to figure out:

1) how many robot trailers were precisely needed to cope with the required good movement, by:

2) determining an optimal disposition of trailers to jobs, and

3) optimally routing the trailers through the system.

The model can be conceived as of having four hierarchical levels (described from bottom to top level):

1) movement of the robot trailers,

2) block reservation mechanism,

3) disposition of paths to trailers, and

4) disposition of trailers to jobs.

Quite obviously, the lowest level is application dependent and thus of not much interest to a larger community of simulation specialists. On the other hand, levels 2 to 4 are quite general and, therefore, deserve to be looked at a little closer. If one replaces each previous occurence of the word "robot trailer" by the word "train", one can immediately see that level 2 of the hierarchy can be applied one to one to this modified situation, while level 3 would apply at least to larger railway stations. Level 4 does not apply, as this step would be taken care of during the evaluation of the time table, that is off-line. If the word "robot trailer" were replaced by "taxi", level 2 could (for obvious reasons) not be directly used, while levels 3 and 4 would apply one to one to the new situation as well.

It is certainly not in our intention to describe the above outlined application to a great detail as this would be of no direct value to others. It is the experience as concerning capabilities and limitations of the SLAM-II software which we want to convey in this article, an experience which we gained during this large-scale modeling process. As the problems mentioned pertain to all transport systems equally, we hope that this experience may become a valuable decision aid for future potential users of the software confronted with the need to develop larger transport system models to decide whether SLAM would be an appropriate vehicle to try their hands on or not.

2) ROBOT TRAILER MOVEMENT

As robot trailers move around with constant speed, it was easiest (and most cost effective) to model the trailer motion by discrete modeling rather than by sets of differential equations. An activity monitoring approach was used for that purpose. A robot trailer leaving one "station" (node) enters an activity. In some cases, the activity duration can be predetermined and can, therefore, be directly assigned, while in others the duration depends on the circumstances (e.g. whether the trailer has to stop at the next node or can drive through). In such places, the USERF option of SLAM-II was used to model the activity duration, that is: the computation of the activity duration is done in FORTRAN.

3) BLOCK RESERVATION

Let us discuss the simplest of all cases, a "transit stop", that is: a point of the network at which the robot trailer may be forced to stop because the following block is currently occopied by another trailer but otherwise would always drive through as it is neither a place where palettes are to be delivered to nor a place from where palettes are to be picked. Fig. 1 models such a "transit stop" in terms of SLAM network elements, while Fig. 2 shows the actual program for one particular transit stop (stop 17).

```
;
;*************************************************
;                    STOP NO 017                  *
;*************************************************
 S017 EVENT,17,1;                  ENTERING STOP 017
       ACT,ATRIB(6);               DRIVE TO STOP
 F017 EVENT,217,1;                 FREE PREV. TRACKS
       ACT,,ATRIB(4).EQ.1.,B017;   NEXT TRACK RES.
       ACT,,ATRIB(4).EQ.0.,A017;   NEXT TRACK OCC.
 A017 AWAIT(17),ALLOC(17),,1;      WAIT FOR NEXT TRACK
 B017 GOON,1;                      CONTINUE
       ACT,USERF(17);              LEAVES STOP 017
 K017 EVENT,417,1;                 FREE STOP 017
       ACT,ATRIB(6),ATRIB(2).EQ.1.,S018;    GOTO 018
       ACT,ATRIB(6),ATRIB(2).EQ.2.,S031;    GOTO 031
       ACT,ATRIB(6),ATRIB(2),EQ,3.,S063;    GOTO 063
;
```

Fig. 2: SLAM Network Program Segment for Transit Stop

The robot trailer (RT) enters the stop at the EVENT-node labeled Si. At this moment, the subroutine EVENT is called by SLAM-II with an event code of i. This EVENT-node represents the interface to the hierarchically higher levels 3 and 4. The target number is compared to the direction code and is possibly adjusted (level 3). At "disposition stops", a job may be assigned to a previously idle RT (level 4). After this has been accomplished, ATRIB(4) and ATRIB(6) obtain their values. ATRIB(4) is used to determine whether the next track could be immediately allocated (=1) or was currently occupied (=0). ATRIB(6) determines the time needed to pass through the stop. This time interval depends on whether the RT has to stop or may drive through, and possibly whether it has to await a new job disposition.

After ATRIB(6) time units (activity duration), the RT enters another EVENT-node labeled Fi. Now, the EVENT subroutine is called by SLAM-II with an event code of (200+i). Here, all previously allocated tracks are freed (possibly with the exception of one stop back, as there exist a few stops which are so close to each other that they may not be simultaneously occupied by two RT's). It is necessary to perform this in an EVENT-node rather than in a FREE-node, as there are possibly several tracks to be returned at once, and as FREE-nodes do not allow to refer to resources by number rather than by name. This would require network programming for each stop individually.

If the following track(s) could not yet be reserved, the RT comes to an AWAIT-node labeled Ai in which subroutine ALLOC is called for resource allocation. This call to ALLOC is unavoidable as it is often necessary to reserve several tracks at once. This cannot happen sequentially, as there may occur deadlock situations otherwise.

After the RT received the required tracks, it passes through the GOON-node labeled Bi and then enters an activity the duration of which is computed in the USERF subroutine (time to speed up in case the RT had to stop). In the EVENT-node labeled Ki, finally, the just left stop is freed (except if the next one is very close). The time to travel to the next stop is computed and stored in ATRIB(6). ATRIB(2) contains the direction code which is used for branching.

This concludes the description of the simple transit stop. Loading and unloading stops are a little more complex, and also the picking places are modeled separately in terms of SLAM-II network elements.

Altogether, the network contains about 120 stops, corresponding to roughly 3000 lines of network description code.

4) DISPOSITION OF PATHS TO TRAILERS

The disposition problem involves basically a lot of sequential logic which is not easily expressible in terms of a SLAM-II network. All disposition logic is, therefore, programmed in FORTRAN, interfaced to the network through the EVENT-nodes labeled Si.

Whenever a branching is possible, the direction code must be recomputed (and stored as ATRIB(2)). This is currently done by table look-up, that is: corresponding to the programmed network structure, there exists also a topological data structure from which it is possible to evaluate for each stop its direction code given the actual stop number and the target code (that is: the number of the stop at which a palette is to be delivered or from where a palette is to be fetched). This disposition is currently entirely static, that is: it does not take into account how much traffic is on the way to determine dynamically alternative paths to the target stop. The program is, however, written in a very modular way in that all table look-up for disposition of paths takes place in one single subroutine which could be easily replaced by a more intelligent one.

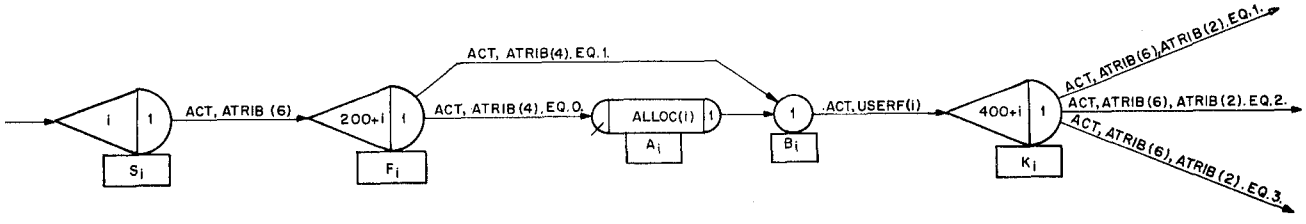

Fig. 1: SLAM Network for Transit Stop

5) DISPOSITION OF TRAILERS TO JOBS

There exist basically two different algorithms to assign trailers to jobs. One approach would be to let each new job "look around" for the next free trailer, while the other algorithm lets each free trailer "look around" for unassigned jobs. It is this second algorithm which was used in our program. Idle RT's travel restlessly in a circular path around the 6 picking places, and, along this path, there exist a few distinct "decision stops" in which idle trailers look for new jobs. Also this disposition problem is interfaced to the network through the EVENT-nodes labeled Si. As for the previous disposition problem, the currently used strategy is still fairly simple, but takes place in a separate subroutine which could easily be replaced by something more elaborate.

6) CRITICAL ASSESSMENT OF SLAM-II FOR THIS PROBLEM

Array Limitations: SLAM-II is programmed in FORTRAN, and thus inevitably places limitations on the size of some "system quantities" (e.g. state variables, resources, etc.). As these limitations are not explicitely mentioned in any SLAM document of which we dispose (we had to read them out from the source listing!), it was decided to list these limitations explicitely in table 1 of this paper.

SLAM quantity	size
number of differential eq.	100
number of state conditions	25
number of entry nodes	25
number of gates	25
number of resources	75
number of histograms	50
number of cells (in histo.)	500
number of collect stat.	50
number of time pers. stat.	50
number of attributes	100
number of files	100
number of plots	10
number of dep. var. per plot	10
number of random streams	10
number of activities	100
number of nodes	500
number of service activities	50
number of global variables	100

Table 1: Size Limits in SLAM-II

For our application, we had to enlarge 7 of these 18 quantities (e.g. the number of activities of which we required 750). It was a very pleasant surprise to learn that all these limitations appear in the program only in terms of array dimensions, while the limit itself is kept as a variable which is assigned precisely once for each of these quantities and then used throughout the program (no more explicit references in DO-loops or similar as with the former GASP system!). In this way, the modifications could be achieved within a few hours and the enlarged SLAM system was operational upon the first trial.

Expressiveness of the Language: The program described in this paper was developed by one student during his diploma thesis (with a duration of 8 weeks). The fact that such a complex study could be performed in this short time by an (admittedly good) student who had no experience in using SLAM before shows better than any detailed analysis one of the highlights of SLAM, namely the ease with which models are developed. We faced no major problems in expressing any of the system components by means of SLAM (which would certainly not have been the case for a software system like Q-GERT or GPSS-V, that is, for a software which does not provide for such a flexible FORTRAN interface).

Program Debugging: As it was to be expected, the program contained many errors in the beginning. We noticed that network errors are extremely easy to debug by means of the (well readable) tracing facility offered in SLAM. FORTRAN errors were much more difficult to trace, and also the use of process interaction routines within the FORTRAN portion of the program created some problems in that these calls often generate entries into the trace which are not so easily interpretable. Debugging the FORTRAN portion required several nights of work of the project supervisor (!), while network errors were in all cases found by the student himself.

Allocation of Resources: The original version of SLAM, as described in [4], does not allow to allocate several different resources at one node (that is: simultaneously). This feature is, however, essential when deadlock situations are to be avoided. Let us assume that there exist two types of resources with a capacity of one each. Two different transactions (possibly at different points in the network) require both resources. If each of them allocates one resource first, they both must wait forever to get the other one! In our problem, this situation may happen with the allocation of multiple intersections which may be required by the RT's coming from different sides. The only way around this problem is by providing a means to allocate several different resources (e.g. a multiple intersection) in one single (that is: uninterruptable) command. SLAM-II, described in [3], provides now the possibilty to achieve this by calling subroutine ALLOC from an AWAIT-node.

Beside of these SLAM-specific (meanwhile resolved) resource allocation problems, there exist also the "usual" shortcomings of commonly used resource allocation mechanisms, as they are described in a companion paper [5]. Resources may readily be freed which have never before been allocated; transactions may leave the system without returning their allocated resources, thus taking them to grave (nice feature to model the behavior of people passing away while leaving their money on a number account in a Swiss bank!).

Execution Efficiency: It is still too early to finally judge the efficiency of SLAM-II as compared to other programs. During the oral presentation, we may have more results available as to this point. One remark can, however, be made already now: The compilation of the network description becomes quite expensive as the network grows in size. For our example (with about 3000 lines of network code), more than 120 sec of CPU time were required on a CDC CYBER installation for the compilation of the network. This is acceptable if there exists a possibility to store the compiled network away for later reuse, like:

```
SAVE,8*
```

to store the network description on tape 8, and

```
LOAD,8*
```

te reload it from there. Remember that the disposition logic is coded entirely in FORTRAN. It makes a lot of sense to try different disposition strategies on one and the same topological model. Unfortunately, such a feature does not exist at the moment.

Macro Facility: Unlike Q-GERT, SLAM does (until now) not provide for a macro facility. In larger networks, such a facility is, however, essential as there tend to exist larger portions of the network which are often repeated (like the stops in our application). Such a facility, when properly designed (which is not really the case in Q-GERT as macros there may not carry formal arguments!), also allows for a much more modular hierarchical structuring of the model. In our case, we decided to develop a (relatively short) PASCAL program to generate the network description out of a few master elements (like the previously sketched transit stop) together with a topological network description of the system (data file). This seemed easier than to use one of the available general purpose macro languages (like ML/I [1]) for which the definition of macro bodies tends to become somewhat clumsy. (An application of that approach was discussed in [2].)

Memory Requirements: Although a macro facility would be very useful, it does not solve all problems either. For our application, we required CM240000 on our CYBER installation which is almost at the upper limit of available core. (ECS could have been used also, but would have required a larger portion of the SLAM software to be modified). Obviously, the operating system on our CYBER is somewhat old fashioned as it does not provide for a virtual memory mechanism (on either our VAX or the IBM 3033 we would have faced no space problems at all!). Nevertheless, it may be preferable to reduce the required memory anyhow for efficiency considerations. This reduction could e.g. be achieved by recycling the RT's again and again through the same standard block. However, to do this, we would have to reprogram even larger portions of the network in FORTRAN. In the end, it may become easier to forget about the network description capability all together and resort entirely to FORTRAN. (Notice that even now most of the utilized network elements are just links to FORTRAN subroutines.) However, this does not mean that the network description becomes useless. Even in that case, SLAM would have helped us through its network capabilities in the structuring of the software to an extent which cannot be overestimated. We are convinced that, without the support from the SLAM software, the student would have had no chance whatsoever to obtain any results for this problem within the short available period of 8 weeks.

Hierarchical Structures: As it can be seen from Fig. 1, it is not easy to separate in this model the hierarchical levels (1 and 2) from each other. Also levels 3 and 4 are coded in two parallel rather than hierarchically structured subroutines. This tendency of "smearing" hierarchy levels is inherent in the network modeling approach. In another project, the process oriented (PASCAL extension) language MODULA-II [6] has been used to control (in real-time) a railway system (childrens' toy). In that project, the hierarchy of train process, train scheduler process, block reservation process, and disposition process could be quite naturally maintained in the software.

Overall Assessment: All of the previously available transaction oriented simulation software systems (like GPSS-V or Q-GERT) were much too inflexible to allow real complex systems (as they are found in industry) to be adequately and conveniently modeled. SLAM-II is the first such system which, as we feel, provides for sufficient flexibility to make it also appropriate for large-scale industrial processes. Even if the final production software (as in our case) has to resort largely to FORTRAN programming, the modeling support available in SLAM justifies by far the inevitable overhead inherent in any "cure-all" system.

ACKNOWLEDGMENTS

We would like to thank Digitron AG for the highly interesting practical application problem and for all the excellent support which we received from them during the execution of this project. We would also like to thank Dr. Hazeghi and Mr. Benninger from the Institute of Operations Research at ETH for their cooperative support of this project.

REFERENCES

[1] Brown P. J.: (1975) "Macro Processors and Techniques for Portable Software". John Wiley.

[2] Cellier F. E.: (1976) "Macro-Handler for Simulation Packages Using ML/I". Proceedings of the 8th AICA Congress on Simulation of Systems, (L. Dekker, ed.), North-Holland Publishing Company, pp. 515 - 521.

[3] Duket S. D., J. J. O'Reilly and R. J. Hannan: (1981) "SLAM-II, Enhanced Simulation Capabilities". Pritsker & Associates, Inc., P.O.Box 2413, West Lafayette, IN 47906.

[4] Pritsker A. A. B. and C. D. Pegden: (1979) "Introduction to Simulation and SLAM". Halsted Press (John Wiley) and Systems Publishing Corporation.

[5] Rimvall M. and Cellier F. E.: (1982) "The GASP-VI Simulation Package for Process-Oriented Combined Continuous and Discrete System Simulation". (this volume).

[6] Wirth N.: (1980) "MODULA-II". Internal Report No 36, Institute for Informatics, ETH-Zentrum, CH-8092 Zurich, Switzerland.