Avoiding shared resource conflicts in mobile sensor networks with multiple missions[1]

Koushil Sreenath, Vincenzo Giordano, Frank Lewis
Robotics and Automation Research Institute,
University of Texas at Arlington, J. Newell Blvd. 7300, Fort Worth, TX 76118 USA

***Abstract*——As sensor networks become increasingly autonomous and grow to include mobility and actuation, the need for predictability in the execution of complex missions becomes crucial. In this perspective, we propose a discrete event controller (DEC) as an effective framework for modelling task constraints, priority orders and task schedules of mobile sensor networks in charge of executing multiple missions. The focus of this paper is to show that for such systems risks of deadlock arise and that the proposed DEC is able to easily implement effective deadlock avoidance algorithms. Several simulations and experiments of coordination policies for mobile sensor networks with shared heterogeneous resources are presented.**

***Index Terms*——Mobile sensor network, deadlock avoidance, discrete event control**

1. INTRODUCTION

A mobile sensor network (MSN) is a geographically distributed network whose heterogeneous nodes are able to perform certain tasks, such as measuring, manipulating or moving [1, 10, 4, 15]. Applications range from environmental monitoring to rescue operations in the event of calamities and exploration of dangerous or unknown environments. As for generic multi-robot systems, the key research issue in this field is to endow the MSN with the sufficient intelligence to automatically react to stimuli of external environments according to a predefined set of cooperation rules. In related literature, given the distributed nature of MSN, decentralized approaches are usually preferred [3, 5, 2, 6, 19]. All these approaches usually refer to the achievement of a single mission and require substantial modifications if the mission goal changes. Also, a MSN with homogeneous resources is usually considered.

If more complex scenarios are envisioned and multiple missions (i.e. multiple sequences of interconnected tasks) with multiple goals have to be implemented, the risk that the system gets stuck into deadlocks arises. In fact distributed MSN may have numerous heterogeneous resources that are shared by different tasks in different missions. Task sequencing and resource assignment in such MSNs is challenging and improper assignment of shared resources can lead to blocking phenomena and, in extreme cases, to deadlock. The implementation of deadlock avoidance policies in autonomous distributed robotic systems such as MSN has not been thoroughly investigated yet, even if it is apparent that a purely distributed control approach cannot solve this problem.

In a previous paper [7], we have shown through simple simulation studies how a matrix-based discrete event controller allows one to easily implement efficient deadlock avoidance policies for shared resources in heterogeneous MSN with multiple missions. This control architecture also provides an intuitive tool for easily programming the mission goals and priorities, which is a major concern in MSN if external conditions change or more information are available to a human operator (e.g see [12]).

[1] Supported by ARO grant DAAD 19-02-1-0366; ARO grant ARO W91NF-05-1-0314; NSF grant IIS-0326505; NSF grant CNS-0421282, Singapore SERC TSRP grant 0421120028, NI Lead User grant

In this paper we will significantly extend our previous study to more challenging and realistic scenarios that include circular wait relations among shared resources in different missions. If shared resources are not appropriately assigned, such circular waits can lead to various blocking phenomena, especially deadlock [18], where the MSN resources are occupied in such a manner that they will never be released, and all activity in the MSN comes to a halt. Also we will show through experimental implementations on an actual MSN test-bed (composed of mobile sensors and unattended ground sensors) the practical feasibility and effectiveness of the proposed deadlock-free coordination policy for MSNs with heterogeneous resources in charge of executing multiple complex missions.

2. MATRIX-BASED DISCRETE EVENT CONTROLLER

The matrix-based discrete event controller proposed in [17] provides a rigorous, yet intuitive mathematical framework to represent the dynamic evolution of DE systems according to linguistic *if-then* rules:

Rule i: If <conditions^i hold > *then* <consequences^i>

In particular for a MSN, we can define the mission planning in the following way:

Rule i: If <sensor1 has completed task1, robot2 is available and a chemical alert is detected > *then* <robot 2 starts task4>

We now show how our DE controller allows one to easily represent these linguistic rules in a rigorous fashion.

Let r be the vector of resources used in the system (e.g. mobile robots and unattended ground sensors(UGS)), v the vector of tasks that the resources can perform (e.g. *go to a prescribed location*, *take a measurement*, *retrieve and deploy UGS*), u the vector of input events (occurrence of sensor detection events, scheduled time instant, etc.) and y the vector of outputs (completed missions). Finally, let x be the logical state vector of the rules of the DE controller, whose entry of '1' in position i denotes that rule i of the DE control policy is currently activated.

Then we can define two different sets of logical equations, one for checking the conditions for the activation of rule i (matrix controller state equation), and one for defining the consequences of the activation of rule i (matrix controller output equation). In the following, all matrix operations are defined to be in the or/and algebra, where + denotes logical *or* and 'times' denotes logical *and*.

The matrix controller state equation is (see [17] for more details):

$$\bar{x} = F_v \bar{v} + F_r \bar{r} + F_u \bar{u} + F_{ud} \bar{u}_d \tag{1}$$

where x is the task or state logical vector, F_v is the task sequencing matrix [16], F_r is the resource requirements matrix [9], F_u is the input matrix. F_{ud} is the conflict resolution matrix and u_d is the conflict resolution vector. The current status of the DE system includes task vector v, whose entries of '1' represent 'completed task', resource vector r, whose entries of '1' represent 'resource (robot or UGS) currently available', and the input vector u, whose entry of 1 represent the occurrence of a certain predefined event (fire alarm, intrusion etc.). The overbar in equation (1) denotes logical negation so that tasks complete or resources released are represented by '0' entries.

The activated rules determine the commands to the MSN that the DEC has to sequence in the next iteration, according to the matrix controller output equations:

$$v = S_v x \tag{2}$$

$$r = S_r x \tag{3}$$

$$y = S_y x \tag{4}$$

S_v is the task start matrix, S_r is the resource release matrix and S_y is the output matrix (see [17] for more details]). The task start equation (2) computes which tasks are activated and may be started, the resource release equation (3) computes which resources should be released (due to completed tasks) and the mission completion equation (4) computes which missions have been successfully completed.

It is worth mentioning that all the coefficient matrices in equation 1-4 are composed of Boolean elements and are sparse, so that real time computations are easy even for large interconnected DE systems.

Finally in order to provide a complete dynamical description of the DE system, we define the following quantities (equivalent to the marking vector, the output incidence matrix and input incidence matrix of a PN, see e.g. [14]):

$$m(t) = [u(t)', v(t)', r(t)', u_d(t)']'$$
$$S = [S_u', S_v', S_r', S_{u_d}', S_y']'$$
$$F = [F_u', F_v', F_r', F_{u_d}', F_y']'$$

where t represents time. Then, in order to take into account the time durations of the tasks and the time required for resource releases, we can split $m(t)$ into two vectors, one representing available resources and current finished tasks ($m_a(t)$) and the other representing the tasks in progress and busy resources ($m_p(t)$)

$$m(t) = m_a(t) + m_p(t) \tag{5}$$

As a consequence, considering equations 1-4 which represent the rule-base of our DE supervisory controller, we have

$$m_a(t+1) = m_a(t) - F' \cdot x(t) \tag{6}$$
$$m_p(t+1) = m_p(t) + S \cdot x(t) \tag{7}$$

It results that when a rule is activated (equation 1) some tasks end and some resources become available (equation 6), whereas some other tasks start and some other resources become busy (equation 7).

Equations (1), (6) and (7) represent a complete description of the dynamical behavior of the discrete event system [17] and can be implemented for the purposes of computer simulations using any programming language (e.g. MATLAB® or C). This is a crucial result for mobile wireless sensor networks where direct experimentation on the hardware can be indeed challenging and expensive.

As shown in [8], the practical implementation of the DEC as a framework for coordination of MSN with multiple concurrent missions follows few simple steps. For each mission i we implement the task constraints (using F_v^i and S_v^i), the schedule according to which certain missions have to be started (updating vector u^i), and any decentralized task allocation algorithm (updating F_r^i and S_r^i matrices). Multiple missions are then implemented by conveniently stacking together the sets of vectors and matrices of each mission.

In this paper we will extend the results proposed in [8] to more complex scenarios, in which the implementation of multiple missions determine shared resource conflicts (i.e. conflicts deriving by the simultaneous activation of rules which start different tasks requiring the same resource) and deadlocks which have to be avoided. In equation 1, matrix F_{ud} and vector u_d are used to resolve conflicts of shared resources. Briefly, an entry of '1' in position j in u_d, determines the inhibition of logic state x_i (rule i cannot be fired). It results that, depending on the way one selects the conflict-resolution strategy to generate vector u_d, different dispatching strategies can be selected to avoid resource conflicts or deadlocks. As shown in section 3, this result will be exploited in this paper to implement a real-time deadlock avoidance policy for MSNs.

3. MATRIX-BASED DEADLOCK AVOIDANCE POLICY

As shown in [11, 13], the matrix constructions presented in section 2 can be efficiently used to implement deadlock avoidance policies for discrete event systems. In the following we will consider the following assumptions:

- No resource fails during a mission
- A resource always complete its current task before starting a new one

- Every resource performs one task at a time
- After the task is completed, the resource is immediately available for a new task
- Each task requires one resource to be executed

For any two resources r_i and r_j, r_i is said to wait for r_j, denoted $r_i \rightarrow r_j$, if the availability of r_j is an immediate requirements for the release of r_i. Circular waits (CW) among resources are a set of resources r_a, r_b,… r_w whose wait relationship among them are $r_a \rightarrow r_b \rightarrow \ldots \rightarrow r_w$ and $r_w \rightarrow r_a$. The simple circular waits (sCW) are primitive CWs which do not contain other CWs. For a complete analysis of the deadlock structures, all the CWs need to be identified, not only the sCWs.

The matrix formulation of (1)–(4) provides a very direct computational method for deadlock avoidance.

First of all, we need to calculate the digraph matrix

$$W = \left[S_r \cdot F_r\right]^T \tag{9}$$

which is a square matrix whose dimension is equal to the number of resources in the system. Each '1' in the w_{ij} elements in W, means that the digraph has an arc from resource i to resource j, indicating that resource i waits for resource j. Using the digraph matrix W with the binary algorithm to identify loops and with Gurel's algorithm described in [13], we can obtain matrix C_{out} which provides the set of resources which compose every CW (in rows). An entry of one on every (i,j) position of C_{out} means that resource j is included in the i^{th} CW.

Since deadlock conditions depend on the number of available resources in every CW, we also need to calculate the set of rules which, when fired, increase or reduce the number of available resources in each CWs (input and output rules).

The input and output rules of a CW are calculated as follows:

$$_d C = C_{out} \cdot S_r \tag{10}$$

$$C_d = C_{out} \cdot F_r^T \tag{11}$$

where the *(i,j)* element of $_dC$ (C_d) is 1 if the *jth* rule increases (reduces) the number of available resources in the *ith CW*.

In order to avoid deadlocks, we have to monitor those tasks of the MSN whose completion activate rules which consume resources in a CW. The task set of a CW C, $J(C)$, is the set of tasks which need at least one of the resources of C to be started. A *siphon* is a set of tasks and resources which if gets empty (none of its tasks are in progress and none of its resources are available) after a certain rule fires, then it will remain empty under any successor rule. The critical siphon of a CW C is the smallest siphon containing the CW. The siphon-task set $J_s(C)$ is the set of tasks which, when added to the set of resources contained in CW C, yields the critical siphon. The critical subsystem of a CW C, $J_o(C)$, is the set of tasks from $J(C)$ not contained in the siphon-task set $J_s(C)$. If the number of activated tasks of the critical subsystem is equal to the resources of the CW, it means that all the resources of the CW are busy, i.e. the CW is empty. Since, by construction, the tasks of the critical subsystem, when completed, never increase the number of the available resources of a CW, the CW remains indefinitely empty and the activity of the MSN comes to a halt. Under the assumptions previously presented, a deadlock condition occurs if and only if there is an empty circular wait [11, 13, 14, 18]. For these systems, an empty CW can only be caused by activation of tasks of the corresponding critical subsystem, whose matrix formulation can be calculated as follows ([13]):

$$J_o = (_d CF_v) \wedge (C_d F_v) = (C_d S_v^T) \wedge (C_d F_v) \tag{12}$$

where each entry of one in position (i,j) means that task j is included in the critical subsystem of CW i.

For sake of completeness we also report the matrix formulation of the critical siphon:

$$J_s = _d CF_v \wedge \left(\overline{C_d F_v}\right) = C_d S_v^T \wedge \left(\overline{C_d F_v}\right) \tag{13}$$

where each entry of one in position (i,j) means that task j is included in the critical siphon of CW i.

A simple deadlock avoidance strategy (which has been so far evaluated only in simulation) consists in not allowing the number of activated tasks of the critical subsystem to become equal or greater than the number of available resources in the *ith* CW C_i (MAXWIP policy [11, 13]).

$$m(J_o(C_i)) < m_o(C_i) \tag{14}$$

Therefore, we can conveniently update the conflict resolution input u_d to inhibit rules which, if activated, would violate condition 14 and lead to deadlock conditions.

Our dispatching policy follows three main steps:

1. Based on the structure of the system defined by matrices F and S, we calculate the CWs, their corresponding critical subsystem and the number of available resources $m_o(C_i)$ in the *ith* CW C_i (off-line computation).
2. For every DE-iteration, we calculate from the current marking vector, $m_{current}$, the corresponding possible successor-marking vector, $m_{possible}$. Equation (6) provides this possible successor $m_a(t+1)=m_{possible}$; $m_a(t)=m_{current}$; $m_{possible}$ is readjusted keeping into account possible shared resource conflicts (on-line computation).
3. If the selected $m_{possible}$ does not satisfy condition (14), then it is necessary to eliminate the task that is attempting to cause a deadlock, inhibiting the corresponding rule. This is done by conveniently updating vector u_d. Then the algorithm restarts from step 2 (on-line computation).

4. Implementation of DEC on WSN Testbed

The Mobile Sensor Network Test-bed at the Automation and Robotics Research Institute, University of Texas at Arlington, consists of mobile sentry robots, Unattended Ground Sensors, and a centralized control unit where the DEC runs under LabView programming environment. Every resource is connected to the control unit through transceivers (figure 1).

4.1 Mobile Sentry Robots

Two cybermotion SR2 mobile sentry robots (donated by JC Penney, Inc.) formerly used to patrol a warehouse in Dallas, Texas are employed as mobile sensing units. They have an extensive sensor suite including ultrasonic intrusion, optical flame detector, dual passive IR, microwave intrusion, smoke, temperature, humidity and light sensors, and gas sensors including oxygen, NOx, and CO. Each robot's task is executed through an ad hoc LabVIEW® VI. For sake of simplicity manipulation tasks have been implemented just as time delays. However, this does not affect the behavior of our DEC which is the focus of this work.

4.2 Unattended Ground Sensors

A set of six Berkeley Crossbow unattended ground sensors (UGSs) has been incorporated into the Secure Area Test-bed at ARRI. They can measure various quantities such as Light, Acceleration, Temperature, Magnetism and Sound. The UGSs form a star network and communicate through a wireless link with the base station connected to one of the serial ports of the microcontroller board.

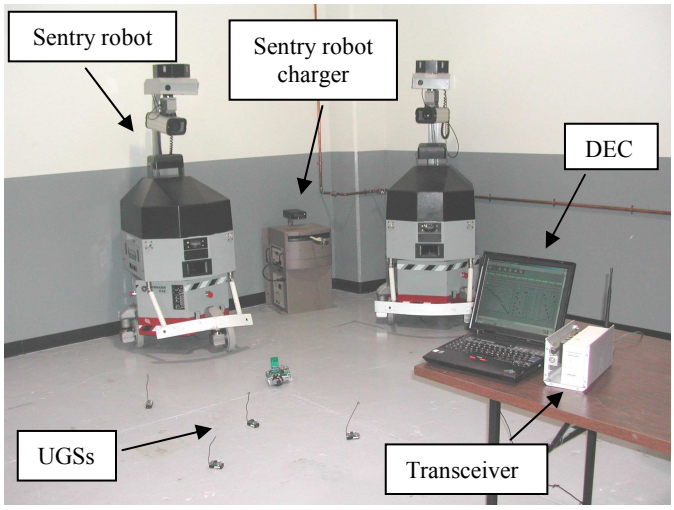


Figure 1- The MSN test-bed at ARRI

4.3 Implementation

In our motivating scenario, a MSN is in charge of monitoring a warehouse where dangerous chemicals are handled. Based on a fair knowledge of the environment and of the possible operating conditions, it is possible to come up with predefined sequences of tasks that the robots have to accomplish in response to external threats or programmed events.

A virtual MSN test-bed has been created to illustrate various mobile robot movements as the MSN topology reconfigures to handle various missions (figure 2).

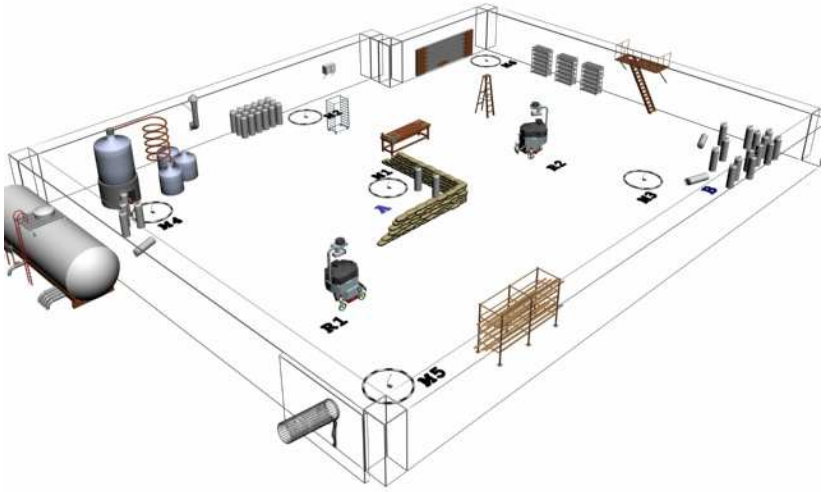


Figure 2- Perspective view of the virtual MSN test-bed with the initial network configuration

5 SIMULATION AND EXPERIMENTAL RESULTS

The results presented in this paper have been obtained using Matlab and Labview programming environments. Matlab has been used for initial simulation of the missions, followed by a Labview implementation of the missions with *simulated resources*. With satisfactory performance of the deadlock resolution algorithm, the simulated resources were replaced by *real resources* and the missions were actually implemented in our lab. Thus the same code has been used for simulation and real implementation of all missions. The similarity and fidelity of the dispatching sequences in both the simulation and

experimental phases was indeed satisfactory. This is a key result since it shows that the proposed DEC allows one to perform a "simulate and experiment" approach for a MSN, with noticeable benefits in terms of cost, time and performance.

Three separate missions have been implemented to illustrate the proposed control architecture. The first mission aims at patrolling the warehouse, the second mission aims at recharging the UGSs batteries, the third mission aims at transporting dangerous chemicals to a predefined location. The triggering events of the three missions are an intruder alert, a low-battery alert, and a prescheduled instant of time respectively. Once the alert is received, the network physically reconfigures its topology to react to the alert. For sake of completeness, the Petri net representation of *mission 1*, *mission 2* and *mission 3* is illustrated in figure 3.

The procedure for implementing the proposed control policy consists of three different steps.

First, we define the vector of resources r of the system. In this example we have two robots R and six stationary sensors M. The resource vector is $r = [R_1, R_2, M_1, M_2, M_3, M_4, M_5, M_6]$.

Then for each mission i, we define the vector of inputs u^i, of outputs y^i and of tasks v^i, and the task sequence of each mission (see Table 1, Table 3, and Table 5 for missions 1, 2, and 3 respectively) and write down the *if-then* rules representing the supervisory coordination strategy to sequence the programmed missions (see Table 2,Table 4, and Table 6 for missions 1, 2, and 3 respectively).

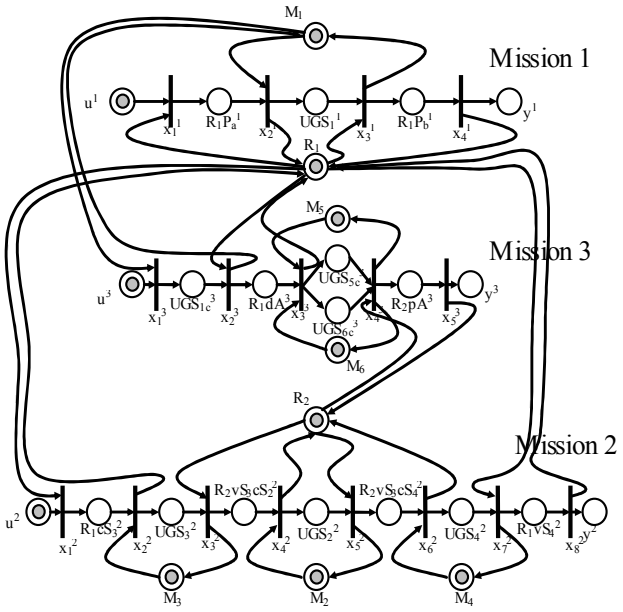


Figure 3- Petri net representation of *mission 1* (Patrol and Sensing of the Warehouse), *mission 2* (Charging of the UGSs) and *mission 3* (Transportation of dangerous chemicals).

Finally, we translate the linguistic description of the coordination rules into a more convenient matrix representation, suitable for mathematical analysis and computer implementation. As an example, matrices F_v^1, F_r^1 relative to mission-1 are reported (figure 4).

Table 1: *Mission 1* - Task Sequence

Mission-1	Notation	Description
Input	u^1	Intruder Alert from any UGS
Task 1	$R_1P_a^1$	i.R_1 navigates to M_2 ii.R_1 takes measurement at M_2 iii.R_1 navigates from M_2 to M_1 iv.R_1 takes measurement at M_1
Task 2	UGS_1^1	i.M_1 takes measurement
Task 3	$R_1P_b^1$	i.R_1 navigates to M_1 ii.R_1 takes measurement at M_1 iii.R_1 navigates from M_1 to M_3 iv.R_1 takes measurement at M_3
Output	y^1	i.Patrol and sensing of warehouse

Table 2: *Mission 1* – Rule-base

Mission 1 – Operation Sequence	
Rule 1 x_1^1	If u^1 *occurs* and R_1 *available* then *start* $R_1P_a^1$
Rule 2 x_2^1	If $R_1P_a^1$ *completed* and M_1 *available* then *release* R_1 and *start* UGS_1^1
Rule 3 x_3^1	If UGS_1^1 *completed* and R_1 *available* then *release* M_1 and start $R_1P_b^1$
Rule 4 x_4^1	If $R_1P_b^1$ *completed* then *release* R_1 and *terminate* mission-1 by producing output y^1

Table 3: *Mission 2* - Task Sequence

Mission-2	Notation	Description
Input	u^2	Low battery warning from an UGS
Task 1	$R_1cS_3^2$	i.R_1 navigates to M_3 ii.R_1 charges M_3
Task 2	UGS_3^2	i.M_3 takes measurement
Task 3	$R_2vS_3cS_2^2$	i.R_2 navigates to M_3 ii.R_2 takes measurement and verifies M_3 charge iii.R_2 navigates from M_3 to M_2 iv.R_2 charges M_2
Task 4	UGS_2^2	i.M_2 takes measurement
Task 5	$R_2vS_2cS_4^2$	i.R_2 navigates to M_2 ii.R_2 takes measurement and verifies M_2 charge iii.R_2 navigates from M_2 to M_4 iv.R_2 charges M_4
Task 6	UGS_4^2	i.M_4 takes measurement
Task 7	$R_1vS_4^2$	i.R_1 navigates to M_4 ii.R_1 takes measurement and verifies M_4 charge
Output	y^2	i.Charging of a predefined set of UGSs

Table 4: *Mission 2* – Rule-base

Mission 2 – Operation Sequence	
Rule 1 x_1^2	If u^2 *occurs* and R_1 *available* then *start* $R_1cS_3^2$
Rule 2 x_2^2	If $R_1cS_3^2$ *completed* and M_3 *available* then *release* R_1 and *start* UGS_3^2
Rule 3 x_3^2	If UGS_3^2 *completed* and R_2 *available* then

	release M_3 and *start* $R_2vS_3cS_2^2$
Rule 4 x_4^2	If $R_2vS_3cS_2^2$ *completed* and M_2 *available* then *release* R_2 and *start* UGS_2^2
Rule 5 x_5^2	If UGS_2^2 *completed* and R_2 *available* then *release* M_2 and *start* $R_2vS_2cS_4^2$
Rule 6 x_6^2	If $R_2vS_2cS_4^2$ *completed* and M_4 *available* then *release* R_2 and *start* UGS_4^2
Rule 7 x_7^2	If UGS_4^2 *completed* and R_1 *available* then *release* M_4 and *start* $R_1vS_4^2$
Rule 8 x_8^2	If $R_1vS_4^2$ *completed* then *release* R_1 and *terminate* mission-2 by producing output y^2

Table 5: ***Mission 3*** **- Task Sequence**

Mission-1	**Notation**	**Description**
Input	u^3	Forty minutes have elapsed
Task 1	UGS_{1c}^3	M_1 takes measurement
Task 2	R_1dA^3	R_1 picks up dangerous chemical and drops off at temporary storage location A
Task 3	UGS_{5c}^3 UGS_{6c}^3	M_5 and M_6 take measurements
Task 4	R_2pA	R_2 picks up chemical from A and transports to location B
Output	y^3	*Dangerous chemical transported*

Table 6: ***Mission 3*** **– Rule-base**

Mission 3 – Operation Sequence	
Rule 1 x_1^3	If u^3 *occurs* and M_1 *available* then *start* UGS_{1c}^3
Rule 2 x_2^3	If UGS_{1c}^3 *completed* and R_1 *available* then *release* M_1 and *start* R_1dA^3
Rule 3 x_3^3	If R_1dA^3 *completed* and M_5 and M_6 *available* then *release* R_1 and *start* UGS_{5c}^3 and UGS_{6c}^3
Rule 4 x_4^3	If UGS_{5c}^3 and UGS_{6c}^3 *completed* and R_2 *available* then *release* M_5 and M_6 and *start* R_2pA
Rule 5 x_5^3	If R_2pA then *release* R_2 and *terminate* mission-3 by producing output y^3

$$F_v^1 = \begin{array}{c} \begin{matrix} R_1P_a & R_1P_b \end{matrix} \\ \begin{matrix} UGS_1 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \text{a)} \quad F_r^1 = \begin{array}{c} \begin{matrix} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & R_1 & R_2 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \text{b)}$$

Figure 4- ***Mission 1*****-Job sequencing matrix** F_v^1 **and Resource requirement matrix** F_r^1

$$C_{out} = \begin{array}{c} \begin{matrix} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & R_1 & R_2 \end{matrix} \\ \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ & & & \vdots & & & & \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix}_{[30x8]} \end{array} \quad \text{a)}$$

$$J_0 = \begin{array}{c} \begin{matrix} R_1P_a & & R_1P_b & & & & R_1dA & UGS_{6c} \\ UGS_1 & & R_1cS_3 & & & & UGS_{5c} & R_2pA \end{matrix} \\ \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ & & & & \vdots & & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \cdots & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \cdots & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{bmatrix}_{[30x15]} \end{array} \quad \text{b)}$$

Figure 5- Circular wait matrix C_{out} and Critical Subsystem matrix J_o.

5.1 Experiment 1- Deadlocks caused by multiple activation of the same mission

Suppose that two different intrusions threats are detected and that, in the meantime, two sensors launch a low-battery alert. In this case both *mission 1* and *mission 2* are triggered two times and deadlock conditions occur (figure 6). In particular we get two empty circular waits: $R_1 \rightarrow M_1$ and $R_2 \rightarrow M_2$. As an example let us analyze the first circular wait. The first row of the circular wait matrix (figure 5a) shows that resources R_1 and M_1 form Circular wait 1, whereas the first row of the critical subsystem matrix (figure 5b) shows that tasks R_1P_a and UGS_1 form Critical subsystem 1. When both these tasks are simultaneously in progress, Circular wait 1 becomes empty and a deadlock occurs.

In figure 6a the Matlab simulation illustrates the triggering of *mission 1* at time instant 14 and time instant 19 for two different intrusion alerts. When *mission 1* is triggered the first time, task R_1P_a starts and runs to completion. Then task UGS_1^1 starts. While UGS_1^1 is still in progress, *mission 1* is triggered again and another instance of task R_1P_a starts. Now resources R_1 and M_1 are consumed. When task UGS_1^1 completes, R_1P_b has to be fired and this requires resource R_1 which is already consumed. Also, when R_1P_a completes, resource M_1 is needed to fire task UGS_1^1. Since M_1 is being used, we have a cyclic wait of resources which leads to a deadlock situation. As shown in figure 6b (path sequences $p_1 \ldots p_5$ and $q_1 \ldots q_4$ describe the motions of R_1 and R_2 robot 2 respectively), R_1 and R_2 keep on wandering in the warehouse without accomplishing *mission 1*. Same considerations hold for *mission 2* with the circular wait $R_2 \rightarrow M_2$.

Figure 7 illustrates the same case when the MAXWIP algorithm is applied. As can be seen, both missions successfully run to completion two times. In particular, for *mission 1*, in order to avoid deadlocks, the dispatching policy inhibits rule 1 (x_1^1) when task UGS_1^1 is in progress by conveniently updating the conflict resolution vector u_d. Similar considerations hold for mission 2. The controller inhibits R_2 from performing task $R_2vS_3cS_2^2$ as long as task UGS_2^2 is in progress. In this way *mission 1* and *mission 2* are executed two consecutive times and the network successfully reacts to multiple intrusions and low battery alarms. A comparison of figure 7a and 7b shows that the task sequences in both simulation and experiment cases show a satisfactory correlation (the different durations of the tasks in the two cases is not related to the behavior of the DEC but to real-word navigation of the robots and is not therefore relevant to our discussion).

5.2 Experiment 2- Deadlock caused by simultaneous activation of multiple missions

As shown in the previous experiment, deadlock can occur due to multiple triggering of a certain mission. Deadlocks can also arise when two or more missions run in parallel and share a circular wait. This scenario

is illustrated in figure 8 wherein *mission 1* and *mission 3* execute in parallel and share the circular wait $R_1 \rightarrow M_1$. As shown in figure 8a, when task R_1P_a from *mission 1* and task UGS_{1c} from *mission 3* execute in parallel, the circular wait $R_1 \rightarrow M_1$ gets empty and a deadlock occurs. With no deadlock resolution, robot R_1 performs task $R_1P_a^1$ first and navigates to sensor M_2 and sensor *M1,* before getting stuck in a deadlock (figure 8b). With the deadlock resolution algorithm applied (figure 9a and 9b), task R_1P_a is inhibited until UGS_{1c} is completed. R_1 performs task R_1dA^3 first and navigates to location "A" before successfully completing the assigned missions (figure 9c).

Interested readers can watch videos of the proposed missions represented in 3D Studio Max at http://arri.uta.edu/acs/WSN/multimedia.html

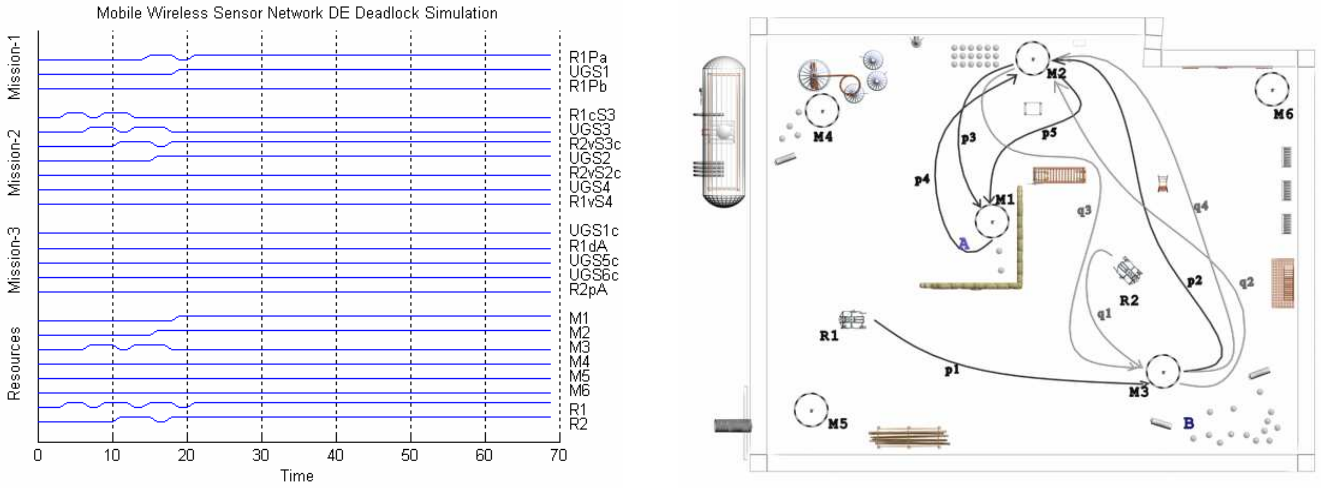
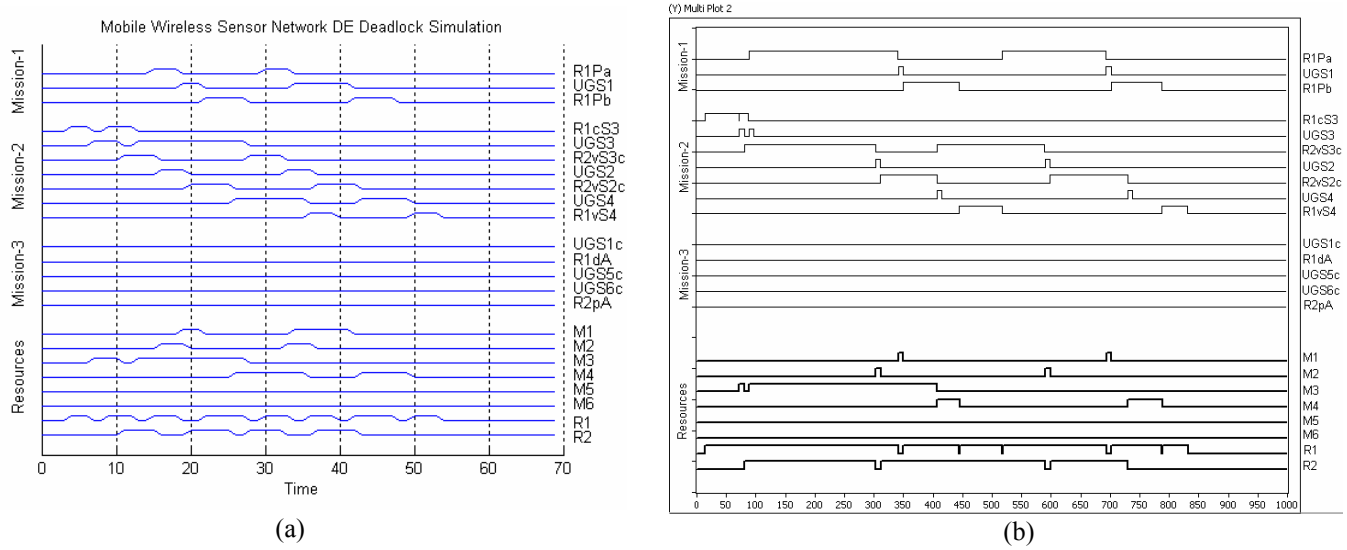
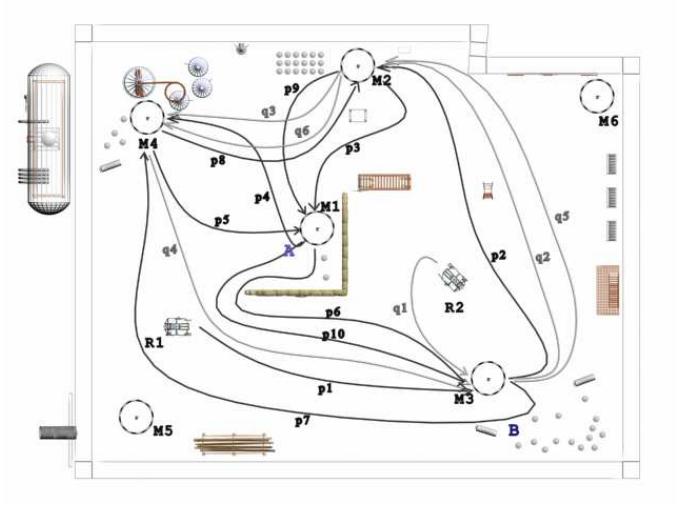


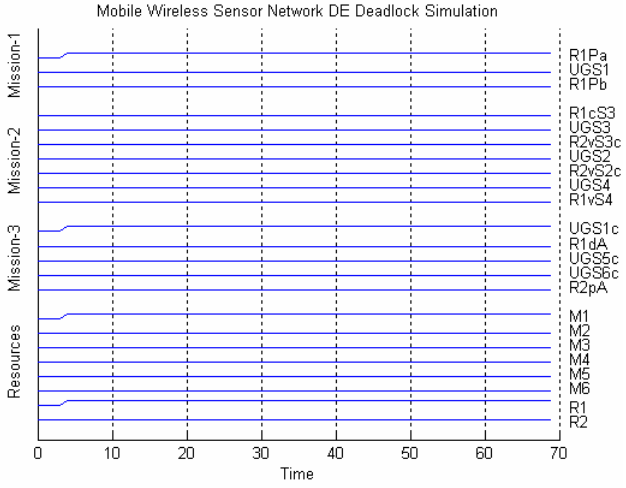
Figure 6. Missions 1, 2 deadlock: (a) Event Time Traces, (b) Top view Robot Paths (Darker paths are R_1 paths, lighter paths are R_2 paths)



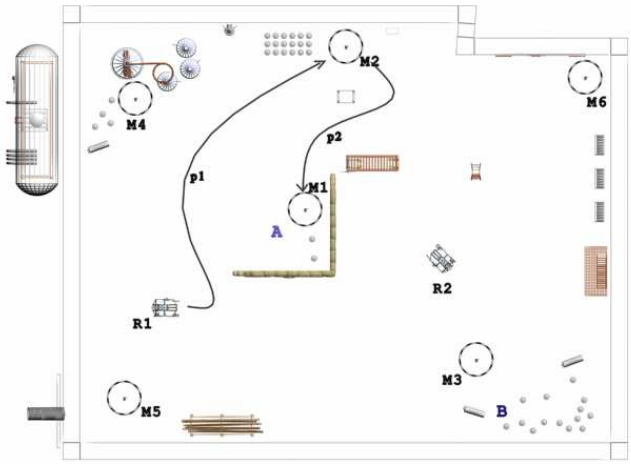


(c)

Figure 7. Missions 1,2 Deadlock Avoidance: (a-b) Event Time Traces-Matlab simulation and LabView implementation, (c) Top view Robot Paths (Darker paths are R_1 paths, lighter paths are R_2 paths)

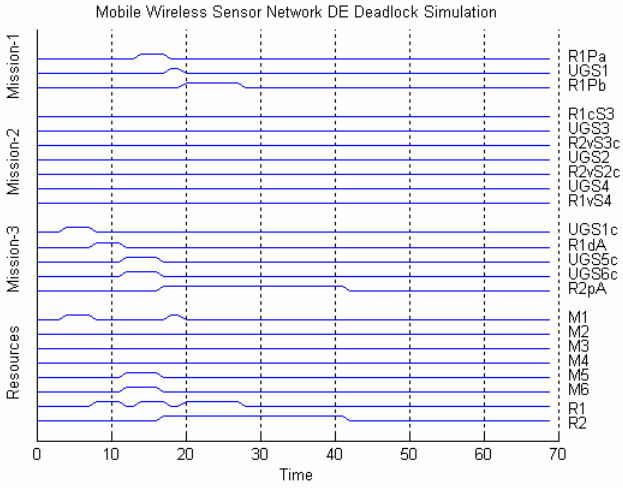


(a)

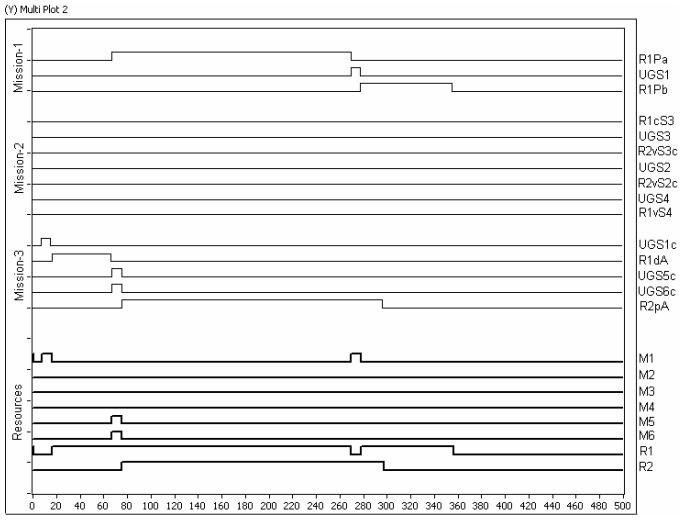


(b)

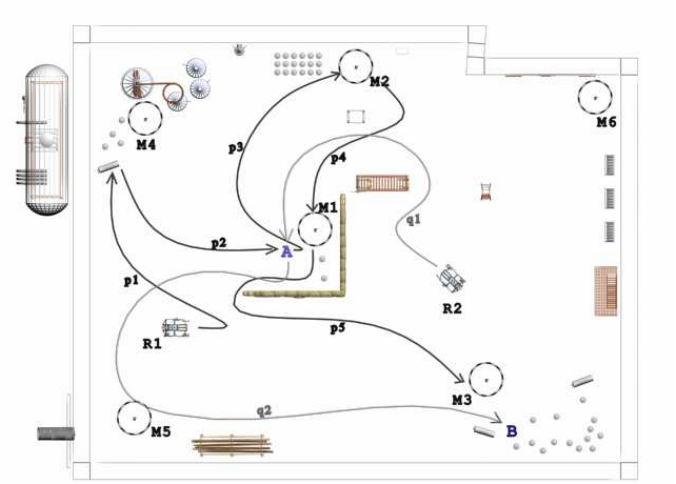
Figure 8. Missions 1,3 Deadlock: (a) Event Time Traces-Matlab simulation, (b) Top view Robots'Paths



(a)



(b)



(c) Robot path. (Darker paths are R_1 paths, lighter paths are R_2 paths.)

Figure 9. Missions 1,3 Deadlock Avoidance: (a-b) Event Time Traces-Matlab simulation and LabView implementation, (c) Top view Robots' Paths (Darker paths are R_1 paths, lighter paths are R_2 paths)

6 CONCLUSIONS

This paper has presented the experimental implementation of a mobile wireless sensor network composed of heterogeneous resources in charge of performing complex interconnected tasks. A discrete event controller has been used to define the sequence of operations each resource has to follow to accomplish multiple concurrent missions triggered by external events. Since the presence of resources shared by multiple missions may lead to deadlocks, a deadlock avoidance policy is applied to on-line to on-line adapt the coordination strategy of the MSN. Both simulation and experimental results have been provided using the MSN test-bed at the Automation and Robotics Research Institute, University of Texas at Arlington. The proposed coordination control strategy proves to be effective to solve deadlocks caused by multiple activation of the same mission or by simultaneous activation of concurrent missions in real-world applications. Current research is devoted to the integration of deadlock avoidance policies together with distributed dynamic resource assignment algorithms.

REFERENCES

[1] Akyldiz I., Su W., Sankarasubramaniam Y, Cayirci E., "A survey on sensor networks", IEEE Communications Magazine, August 2002

[2] Balch T., Hybinette M., "Social potentials for scalable multi-robot formations", Proceedings of the IEEE International Conference on Robotics and Automation, April 2000

[3] Butler Z., Rus D., "Event-based motion control for mobile-sensor network", IEEE Transactions on Pervasive Computing, vol.2 issue 4, October-December 2003

[4] Chong C., Kumar S., "Sensor Networks: Evolution, Opportunities and Challenges", Proceedings of the IEEE, col. 91, no.8, August 2003

[5] Cortes J., Martinez S., Karatas T., Bullo F., "Coverage control for mobile sensing network", IEEE Transactions on Robotics and Automation, vol.20, no.2, April 2004

[6] Gerkey B., Mataric M., "Sold! Auction methods for multirobot coordination", IEEE Transactions on Robotics and Autoation, vol. 18, no. 5, October 2002

[7] Giordano V., Lewis F., Mireles J., Turchiano B., "Coordination control policy for mobile sensor networks with shared heterogeneous resources", Proceedings of the IEEE International Conference on Control and Automation, Budapest, June 2005

[8] Giordano V., Ballal P., Lewis F., Turchiano B., Zhang J. B., "Supervisory control of mobile sensor networks: Matrix formulation, simulation and implementation", to appear in IEEE Transactions on System, Man and Cybernetics part B
[9] Kusiak A. "Intelligent scheduling of automated machining systems", In Intelligent design and Manufacturing. A. Kusiak (ed.) Wiley, New York (1992)
[10] Lewis F., "Wireless sensor networks", Smart environments: Technologies, Protocols, and Applications, ed. D. J. Cook and S. K. Das, John Wiley, New York, 2004.2004
[11] Lewis F., Gurel A., Bogdan S., Docanalp A. Pastravanu O., "Analysis of deadlock and circular waits using a matrix model for flexible manufacturing systems", Automatica, vol.34, no. 9, September 1998
[12] Makarenko A., Kaupp T., Durrant-Whyte H., "Scalable Human-Robot Interactions in Active Sensor Networks", IEEE Pervasive Computing Magazine 2003, vol 2, no 4, Oct-Nov 2003 pp 63--71.
[13] Mireles J., Lewis F., Gurel A., "Implementation of a deadlock avoidance policy for multipart reentrant flow lines using a matrix-based discrete event controller", Proceedings of the International symposium on advances in robot dynamics and control, New Orleans, November 2002
[14] Murata, T. "Petri Nets: Properties, Analysis and Applications." Proceedings of the IEEE, vol.77, no.4, April 1989, pp.541-80
[15] Sinopoli B., Sharp C., Schenato L., Schaffert S., Sastry S., "Distributed Control Applications Within Sensor Networks", Proceedings of the IEEE, vol. 91, no.8, August 2003
[16] Steward D. V., "The design structure system: a method for managing the design of complex systems", IEEE Trans. Engineering Management, pp. 45-54, August 1981
[17] Tacconi D., Lewis F., "A new matrix model for discrete event systems: application to simulation", IEEE Control System Magazine, vol.17 October 1997
[18] Wysk, R.A.; Yang, N.S.; Joshi, S.; "Detection of deadlocks in flexible manufacturing cells", IEEE Transactions on Robotics and Automation, vol.:7 , Issue: 6 , December 1991
[19] Zhao F., Shin J., Reich J., "Information-Driven Dynamic Sensor Collaboration", IEEE Signal Processing Magazine, March 2002