Indoor Automation with Many Mobile Robots

Philippe Caloud, Wonyun Choi,
Jean-Claude Latombe, Claude Le Pape and Mark Yim

CIFE Working Paper
Number 12

January, 1992

Stanford University

IEEE International Workshop on Intelligent Robots and Systems
IROS '90

INDOOR AUTOMATION WITH MANY MOBILE ROBOTS

Philippe Caloud, Wonyun Choi,
Jean-Claude Latombe, Claude Le Pape and Mark Yim
Robotics Laboratory
Stanford University
Stanford CA 94305 USA

Abstract: The goal of the GOFER project is to control the operations of many mobile robots (several dozens) in an indoor environment. This project raises many research issues: implementation of non-conflicting sensor systems, man-robot and robot-robot communication systems and protocols, contingency-tolerant motion control, multi-robot motion planning, multi-robot task planning and scheduling. The aim of this paper is to provide an overview of the project and present our current solutions to these problems.

Acknowledgements: This research has been funded by the Center for Integrated Systems (CIS), the Center for Integrated Facility Engineering (CIFE) and Digital Equipment Corporation (DEC). Claude Le Pape was supported by ELF-Aquitaine.

1 The GOFER Project

The goal of the GOFER project is to control the operations of many mobile robots (several dozens) in an indoor environment (an office environment, a shop-floor, an airport, an hotel) in order to automate a variety of tasks. Typical tasks include transportation of objects (beverages, books, mail), operation of machines (copiers, vending machines), cleaning, maintenance and hazard detection. Different tasks may require different physical capabilities. However, many tasks essentially involve mobility and transportation of relatively small objects.

The GOFER project raises many research issues: design of non-conflicting sensor systems, man-robot and robot-robot communication systems and protocols, contingency-tolerant motion control, multi-robot motion planning, multi-robot task planning and scheduling.

- In section 2, we discuss the development of robot hardware. Although most of the hardware work consists of integrating commercially available products (mobile base, sensors, computing equipment), we have had to develop some hardware components. For instance, most marketed sensors are "active". If several robots use the same active sensors without precaution, they will strongly interfere. The development of passive sensors is consequently an important issue. Another important issue is communication: we want any two robots to be able to directly exchange information in order to cooperate and coordinate their actions appropriately.
- In section 3, we present our current system for planning and execution. This system integrates a collection of software components: a **task planning system** to derive plans made of "high-level" actions (such as "go to position P" or "get object O") from a description of the tasks to be performed; a **task allocation system** to order and allocate tasks or actions to robots; a **motion planning system** to convert high-level actions into motion commands; and an **execution system** to monitor execution and react to unexpected events. These components are implemented in COMMON-LISP on a DEC-3100 workstation. The overall system is tested with the help of a simulator specially designed to simulate actions of autonomous agents [12].
- In section 4, we discuss our current status and future work concerning both the problems considered so far (development of passive sensors, planning, contingency-tolerant execution) and other research issues to consider in the future (development of communication models, definition of an architecture to coordinate all the physical and cognitive activities of a robot).

2 Hardware Development

Since the GOFER project is a relatively large project and is expected to take many years to complete, we decided to improve the robot hardware gradually by creating several generations of robots. This will allow the hardware to be more up-to-date with what technology exists at that time. Therefore, we tried to keep the design philosophy as follows: (1) use as many off-the-shelf products as possible; (2) modularize the parts as much as possible; (3) focus on advancing one technology at a time instead of trying to improve everything at the same time. By following the above philosophy, we were able to construct each generation of the robot in a relatively short amount of time for demonstrating research software while improving the hardware without interruption. Currently, we are developing the third generation of GOFER. Most of the following will apply to this generation.

The robots we currently use are semi-autonomous mobile platforms receiving high-level commands from workstations (Sun-3 and Macintosh II). Each robot is instrumented with odometric, touch, and infra-red proximity sensors. In addition, we are currently implementing a laser-camera-based range sensor. We are also investigating various devices for communication among robots and with a computer network (radio modem/ethernet and infra-red communications).

2.1 Mobile Platform

The hardware of GOFER consists of a 12 inch diameter mobile base (B12 by Real World Interface[1]) and interface modules (see figure 1). The 3-wheeled 2 DOF mobile base is equipped with two DC motors, four 6V gel-cell batteries and an 8-bit microcomputer for low-level control. The mobile

[1] See the "Real World Interface Mobile Robot Base B12 Guide to Operations".

base has a belt-driven synchro-drive mechanism which allows the base to translate and rotate independently. When the base rotates, only the top plate and the wheels rotate. Two optical shaft encoders are directly attached to the motors to provide linear position and angular position of the base. Motor control and inquiries about encoders, battery status and motor status are achieved through a serial port (RS232).

The current interface modules consist of a touch sensor module, an infra-red module, a laser-ranging module and a computer module. Each module, except the touch sensor module, is built on its own modular plate and is placed one on top of the other with the bottom one rigidly attached to the top plate of the base. Interface modules are designed such that they are completely independent of each other. Also, each module communicates on a common robot bus which is passed through each module. Therefore, each module can be detached for test or repair without affecting the use of other modules. This design allows easy expandability. It facilitates debugging and experimentation with the robot hardware.

The computer module has a 5 slot VMEbus card cage. A totally CMOS 20MHz MC68030 based computer board and a custom-designed IO board are installed in it. The 68030 board has battery backed 256 Kbyte SRAM, maximum 1 Mbyte and four RS232 serial ports. It draws about 1 Amp. The IO board has 64 general IO lines and draws less than 100mA.

The touch sensor module is placed around a steel bumper which encases the robot base. The bumper serves to protect the base as well as house six additional 6V gel-cell batteries. It expands the diameter of the base to 18 inches. The touch sensor has two levels of 12 segment tape switches. Each tape switch detects a pressure of 9 oz and provides on/off information. The batteries provide about 10 hours continuous running time powering all the interface modules. By expanding the diameter of the base and putting the heavy batteries at the bottom of the robot, the stability of the robot is greatly enhanced.

On the infra-red sensor module, 16 emitter/receiver pairs are evenly distributed around the perimeter of the modular plate and a control circuit board is located at the center of the plate. Each infra-red emitter/receiver can detect an object within 16 inches in 250 micro-seconds with an average resolution of 0.25 inches. This is an intensity based device, so it is inherently subject to color and specularity problems.

On the laser-ranging module, a laser diode and a CCD camera are placed in a reconfigurable fixture. The fixture allows the laser unit and the CCD camera to have different offsets and relative orientations. More details about the laser-ranging sensor are provided in the following section.

2.2 Laser-Ranging Sensor

Ranging is achieved by emitting a plane of light and watching from some offset distance for intersections with this plane. Then, triangulation is done to find the actual distance. An infra-red laser diode and a cylindrical lens create the plane of light. By turning a camera 90 degrees on its side, so the scan lines run vertically, we can measure the distance to an object by monitoring the composite video output. By measuring the time from the beginning of any scan line to the sudden increase in intensity that would correspond to an object reflecting the laser light, we get a direct relation to the object's distance. An interference filter that allows only the laser light frequency to pass is put over the camera to reduce noise.

This system provides 15360 data points per second, 512 ranging values over the 30 degree field of view of the camera 30 times per second. The timing information along with the intensity of the reflection is then directly dumped into memory by DMA over the VMEbus. The resolution varies with the distance: closer objects are seen with better resolution. The resolution is also directly related to the offset of the camera with the laser plane. So a 6 inch offset produces a theoretical resolution of 0.03 inches at one foot and a resolution of 2.3 inches at ten feet.

2.3 Communication System

We have implemented a 2400 effective baud radio modem with the robots, but are now considering radio ethernet as the technology is becoming available. This would give a higher data rate as well as the well-accepted standard for local area networking. Another system which we are considering is an infra-red ethernet, where the robots dock next to an infra-red communication port and communicate by being in close proximity.

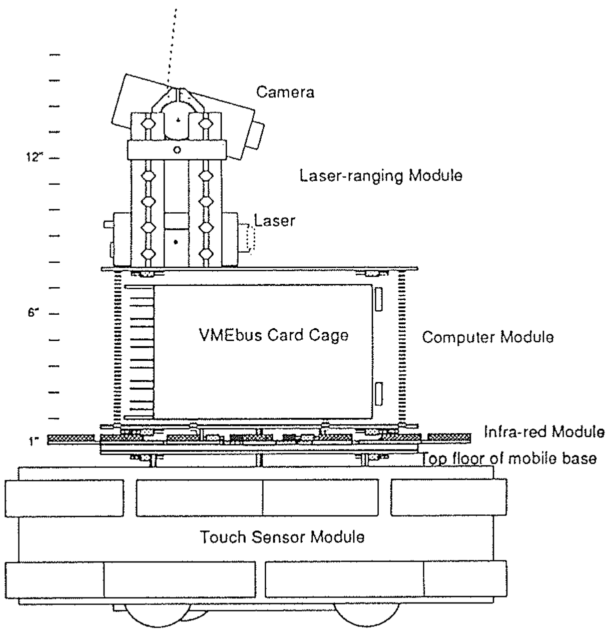

Figure 1: A GOFER

3 Planning and Execution

The overall multi-robot system should be "taskable": it should accept task descriptions which specify *what* the users want to be done rather than *how* to do it. Thus, the robotic system must decide which actions to execute and when. The global goal is to ensure the best service. This includes performing tasks in a timely fashion, minimizing the cost of doing so and bothering humans only when necessary. The reasoning activities which concern planning and scheduling are currently organized as follows:

- Task Planning: determine plans of "high-level" actions (such as "go to position P" or "get object O") from a description of the goals to be achieved.

base has a belt-driven synchro-drive mechanism which allows the base to translate and rotate independently. When the base rotates, only the top plate and the wheels rotate. Two optical shaft encoders are directly attached to the motors to provide linear position and angular position of the base. Motor control and inquiries about encoders, battery status and motor status are achieved through a serial port (RS232).

The current interface modules consist of a touch sensor module, an infra-red module, a laser-ranging module and a computer module. Each module, except the touch sensor module, is built on its own modular plate and is placed one on top of the other with the bottom one rigidly attached to the top plate of the base. Interface modules are designed such that they are completely independent of each other. Also, each module communicates on a common robot bus which is passed through each module. Therefore, each module can be detached for test or repair without affecting the use of other modules. This design allows easy expandability. It facilitates debugging and experimentation with the robot hardware.

The computer module has a 5 slot VMEbus card cage. A totally CMOS 20MHz MC68030 based computer board and a custom-designed IO board are installed in it. The 68030 board has battery backed 256 Kbyte SRAM, maximum 1 Mbyte and four RS232 serial ports. It draws about 1 Amp. The IO board has 64 general IO lines and draws less than 100mA.

The touch sensor module is placed around a steel bumper which encases the robot base. The bumper serves to protect the base as well as house six additional 6V gel-cell batteries. It expands the diameter of the base to 18 inches. The touch sensor has two levels of 12 segment tape switches. Each tape switch detects a pressure of 9 oz and provides on/off information. The batteries provide about 10 hours continuous running time powering all the interface modules. By expanding the diameter of the base and putting the heavy batteries at the bottom of the robot, the stability of the robot is greatly enhanced.

On the infra-red sensor module, 16 emitter/receiver pairs are evenly distributed around the perimeter of the modular plate and a control circuit board is located at the center of the plate. Each infra-red emitter/receiver can detect an object within 16 inches in 250 micro-seconds with an average resolution of 0.25 inches. This is an intensity based device, so it is inherently subject to color and specularity problems.

On the laser-ranging module, a laser diode and a CCD camera are placed in a reconfigurable fixture. The fixture allows the laser unit and the CCD camera to have different offsets and relative orientations. More details about the laser-ranging sensor are provided in the following section.

2.2 Laser-Ranging Sensor

Ranging is achieved by emitting a plane of light and watching from some offset distance for intersections with this plane. Then, triangulation is done to find the actual distance. An infra-red laser diode and a cylindrical lens create the plane of light. By turning a camera 90 degrees on its side, so the scan lines run vertically, we can measure the distance to an object by monitoring the composite video output. By measuring the time from the beginning of any scan line to the sudden increase in intensity that would correspond to an object reflecting the laser light, we get a direct relation to the object's distance. An interference filter that allows only the laser light frequency to pass is put over the camera to reduce noise.

This system provides 15360 data points per second, 512 ranging values over the 30 degree field of view of the camera 30 times per second. The timing information along with the intensity of the reflection is then directly dumped into memory by DMA over the VMEbus. The resolution varies with the distance: closer objects are seen with better resolution. The resolution is also directly related to the offset of the camera with the laser plane. So a 6 inch offset produces a theoretical resolution of 0.03 inches at one foot and a resolution of 2.3 inches at ten feet.

2.3 Communication System

We have implemented a 2400 effective baud radio modem with the robots, but are now considering radio ethernet as the technology is becoming available. This would give a higher data rate as well as the well-accepted standard for local area networking. Another system which we are considering is an infra-red ethernet, where the robots dock next to an infra-red communication port and communicate by being in close proximity.

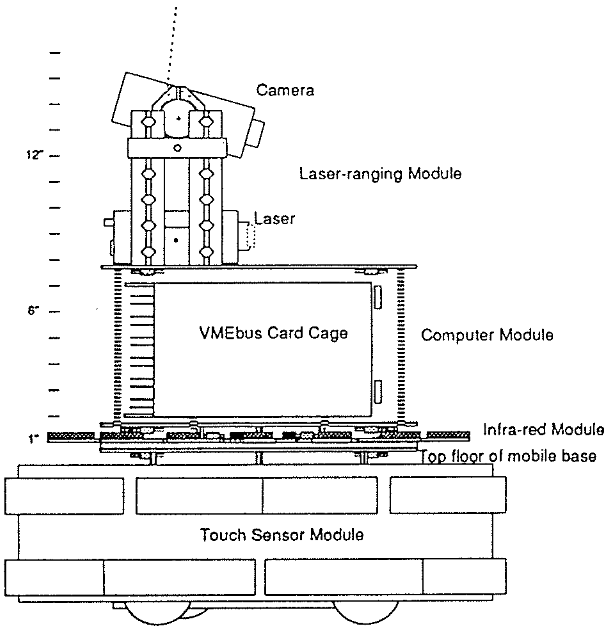

Figure 1: A GOFER

3 Planning and Execution

The overall multi-robot system should be "taskable": it should accept task descriptions which specify *what* the users want to be done rather than *how* to do it. Thus, the robotic system must decide which actions to execute and when. The global goal is to ensure the best service. This includes performing tasks in a timely fashion, minimizing the cost of doing so and bothering humans only when necessary. The reasoning activities which concern planning and scheduling are currently organized as follows:

- **Task Planning**: determine plans of "high-level" actions (such as "go to position P" or "get object O") from a description of the goals to be achieved.

- Task Allocation: allocate tasks to robots with respect to task characteristics and robot availability.
- Motion Planning: convert high-level actions into motion commands.
- Decision Making at Execution Time: monitor plan execution and react to unexpected events.

Distributed problem-solving is advocated for most of these activities. We believe that distributed problem-solving is more efficient in all cases in which (a) communication with a central system is too costly and (b) there is too much uncertainty in the environment to make reliable predictions in advance.

3.1 Task Planning

A robot acquires goals in two ways. The first is to offer or to be asked to achieve goals of other agents (human clients or other robots). We deal with this situation in section 3.2. The second is to autonomously generate goals with respect to the current situation. For example, idle robots should contact other agents through the communication network to determine what they could do for them. A robot generates a goal ((on-line myself) = true) as soon as it is free of other goals.

Task planning consists of determining programs of actions that can be carried out to achieve goals. We consider plan structures, which contain variables, and plan instances, which are instantiated plan structures. A plan structure is defined as an action hierarchy and a set of constraints.

- Action Hierarchy. The top-level action represents the entire process to carry out. Each action can be refined into either a sequence of more detailed actions, a set of un-sequenced actions (to be performed either in parallel or in any order), or a set of exclusive alternatives. The actions at the bottom of the hierarchy are atomic formulas containing variables. For example, (connect ?R ?P) corresponds to the connection of any robot ?R with any communication network port ?P.
- Constraints provide a specification of admissible assignments of values to variables. Type specifications such as ((type ?R) = (or mobile-robot workstation)) describe domains over which variables can vary. Variable relations such as ((location ?R) = ?P) define a subset of the cartesian product of these domains.

A plan instance is an action hierarchy in which variables have been replaced by values satisfying the constraints. For example, [((type ?P) = port) (sequence (move myself ?P) (connect myself ?P))] is a plan structure enabling the achievement of ((on-line myself) = true). If port-22 is a particular port, then (sequence (move myself port-22) (connect myself port-22)) is an instance of this plan structure.

Given a new goal, we consider three planning steps. In the first step, the robot determines plan structures to achieve the goal. Plan structures are either retrieved from a library (as in [6] and [7]) or constructed with the help of a constraint-based (currently linear) planner. In the second step, the robot instantiates plan structures. It uses a general constraint satisfaction algorithm which allows to determine *one*, *all*, *the best* or *the n best* substitutions of variables, with respect to some user-provided evaluation function. For example, it allows to determine the substitution which minimizes an estimate of the total duration of the plan. In the third step, the robot makes final decisions. Depending on its planning policies, the robot may have developed several goals, several plan structures for the same goal and several plan instances for the same plan structure. The overall problem is complex since plans developed to achieve individual goals are likely to interact [4]. One method consists of combining available plans, detecting interactions and solving them. Currently, we apply another method which consists of selecting one plan to execute prior to the others.

Both the construction and instantiation of plan structures are NP-complete problems. However, the use of powerful constraint propagation techniques allows us to considerably reduce search. Furthermore, plan instantiation usually involves a small set of variables (one to ten) and variable relations permitting low-cost propagation steps (consistent with the locality criterion [17] [10]). In the most complex case encountered so far, the planner needs only 4.7 seconds to construct 5 plan structures (with on average 8 actions per plan) and 2.5 seconds to instantiate them (to find the optimal plan instance).

3.2 Task Allocation

In this section, we consider the case in which robots are ready to contribute to achieve goals of other agents. In this case, we use a partially centralized method to determine an appropriate task allocation. Each available robot communicates with a central task planning and scheduling system (CTPS). The CTPS cannot communicate with un-connected robots. However, most of these robots picked tasks in accordance with the CTPS. Hence, the CTPS approximately knows what these robots are doing. Generally speaking, the CTPS has a global view of (a) the tasks to be performed in the environment and (b) the availability of robots to perform these tasks.

The task allocation method is reminiscent of the contract net problem-solving strategy [16] [15]. When the CTPS receives orders (goals) from clients, it generates plan structures enabling the achievement of these goals and provides available robots with a description of pending goals and plan structures.[2] Robots determine which roles they can play in the execution of the plan structures. This means each robot substitutes itself to variables the type of which is consistent with its own type and propagates the substitutions to check the satisfiability of variable relations. Then each robot uses the constraint satisfaction algorithm mentioned in section 3.1 to extend the substitutions and proposes complete substitutions to the CTPS. Each proposal includes (a) a reference to the considered goal, (b) a reference to the considered plan structure, (c) a description of the substitution and (d) values returned by evaluation functions considered during constraint satisfaction (e.g. an estimate of the total execution time).

In general, the CTPS does not receive all the proposals from all the robots at the same time. For example, some robots connect after others made proposals. In some cases, the overall system will perform much better if the CTPS waits

[2]The CTPS can do this because plan structures do not depend on the particular agents contributing to the satisfaction of goals. Conversely, the CTPS cannot instantiate plan structures since it does not know which agent is going to be available to do what. More generally, "agent-independent" and "agent-dependent" planning steps can be distinguished. Any agent aware of a goal can construct agent-independent plan structures with no information about the current situation of agents in the environment. Conversely, deciding which particular agent will execute which plan requires an assessment of the current physical capabilities and situation of the agents.

a few minutes before making a decision. The CTPS cannot precisely predict that the best solution is to wait. However, it can determine whether waiting is *likely* to be better than making an immediate decision. Following other researchers applying concepts from decision theory to planning problems (see [8]), we provide the CTPS with a decision-theoretic definition of the allocation problem. We use an incremental algorithm which makes use of statistical information to maximize the expected value of a given utility function u. Given a set of proposals and a set of goals, the CTPS must decide for each goal either to wait or to allocate the goal to a robot. Decisions can be revised each time a new proposal arrives or when an abnormal situation such as the unexpected absence of a robot is detected. However, when a robot is already executing a plan, the corresponding decision cannot be revised.

Our current implementation relies on a decomposition assumption. The value of u for a set of goals $\{$*goal-1 ... goal-n*$\}$ is a linear combination of (a) the sum over goals of the waiting times preceding the allocation of the goal to a robot and (b) the sum over goals of the utility of having the eventually chosen robot achieving the goal, independently of when it starts doing it. This means we want to maximize the expected value of the expression $\alpha\Sigma$*waiting-time(goal-i)* + $\beta\Sigma$*utility(goal-i achieved as it is achieved)* in cases in which the utility of achieving a goal as it is achieved does not vary with the waiting time. For example, any function of the execution duration and energy consumption satisfies this requirement. In particular, the overall expression to maximize can be set to $-\Sigma$*waiting-time(goal-i)* $-\Sigma$*execution-time(goal-i)* which corresponds to minimizing the average response time between the generation and the satisfaction of a goal.

Given the decomposition assumption, the CTPS can easily compute the expected value of both terms given a set of decisions $\{$*decision-1 ... decision-n*$\}$ for $\{$*goal-1 ... goal-n*$\}$. For each goal and each robot, the CTPS knows the plan instance the robot proposes to execute. With this plan instance and appropriate statistical information, the CTPS computes the expected utility of having the goal achieved as the robot proposes. Therefore, when the CTPS hypothesizes the assignment of a robot *robot-j* to a goal *goal-i*, it knows the utility of this assignment. Furthermore, it can estimate the waiting time for *goal-i* under this hypothesis since *robot-j* will start executing the plan instance as soon as the CTPS tells it to do so. The evaluation is slightly more complex when the CTPS hypothesizes that it will not assign *goal-i* to any of the connected robots. In the absence of any other information, the expected value of *utility(goal-i achieved as it will be achieved in the future)* equals the mean value the function *utility* took for similar goals achieved in the past. Similarly, the CTPS relies on statistical information to determine the sum of the waiting times of m un-assigned goals when n robots are busy and p robots expected soon.

An incremental algorithm is used to review proposals. When a new proposal arrives, the CTPS determines whether the current allocation remains the best. If it does not, then it is replaced by the new optimal allocation. If the CTPS receives other proposals in the meantime, they are reviewed one after the other. When there is no more proposal to review, the CTPS asks robots to execute plans with respect to the optimal allocation. Robots to which tasks are assigned respond with an acknowledgement and plan and execute the corresponding motions in a distributed fashion.

3.3 Motion Planning

The motion planner developed within the GOFER project rests upon three assumptions:

- The geometry of the environment is mainly defined by walls, machines and heavy furniture which do not (or rarely) move in time. Therefore, robots can be provided with an approximate map of the environment and refer to this map to plan motions "in advance".
- In most cases, two obstacles are either very close to each other (a machine may be against a wall) or sufficiently distant to make it easy for the mobile robots to move in the environment. This is not a very restrictive assumption because shop-floors, office environments, hotels, airports, are designed so that it is easy to move inside. For example, the width of a corridor is such that two mobile objects of an average shape and size can easily pass each other. An important consequence of this assumption is that we can compute an artificial road network on which robots will move whenever it is possible.
- In general, there is too much uncertainty about the places where two robots will meet to solve every possible interaction in advance. During execution, robots must apply procedures which ensure that no collision occurs with other robots or, more generally, with other unexpected obstacles.

In this context, we advocate a distributed approach where each robot is responsible for its own motion planning. The planning technique is based on the pre-computation of a road network. Each robot is provided with this network and uses an A* algorithm [13] to compute a path which follows this network. The pre-computation of the network and the motion planning algorithm are presented below. Motion coordination and collision avoidance are ensured at execution time with the help of "behavior rules" and "potential fields" discussed in section 3.4.

To design a road network, we first compute two trapezoidal decompositions of the workspace along two orthogonal directions.[3] Each cell is a trapezoid whose upper and lower edges are edges (or parts of edges) of obstacles (the surrounding rectangular frame is considered as a special obstacle). When a cell has at most one neighbor cell at each of its left and right sides, a road is designed through the cell to connect its left and right edges. Let us call these cells *C-cells*. Any sequence of contiguous C-cells is called a *channel*. The roads in the contiguous cells of a channel can be connected. This results in a road which connects the left side of the leftmost cell to the right side of the rightmost cell of the channel. We call *T-cell* a cell which has more than one neighbor cell at its left or at its right. Let us suppose that a given T-cell T has two neighbor cells at its right. In this case, two roads separated by an obstacle are connected to the right edge of T. A vertical road must be designed to connect these two roads. The key idea is that the decomposition into cells along the orthogonal direction provides us with a channel which overlaps the T-cell T. The road designed in this channel can be used to connect the two roads at the right of T. Therefore, the computation of the road network can be summarized as

[3] We restrict ourselves to a two-dimensional problem: the mobile robots are disks moving among polygonal obstacles without holes inside a rectangular frame. We describe the decomposition along the x axis. The same algorithm applies along the y axis.

a few minutes before making a decision. The CTPS cannot precisely predict that the best solution is to wait. However, it can determine whether waiting is *likely* to be better than making an immediate decision. Following other researchers applying concepts from decision theory to planning problems (see [8]), we provide the CTPS with a decision-theoretic definition of the allocation problem. We use an incremental algorithm which makes use of statistical information to maximize the expected value of a given utility function u. Given a set of proposals and a set of goals, the CTPS must decide for each goal either to wait or to allocate the goal to a robot. Decisions can be revised each time a new proposal arrives or when an abnormal situation such as the unexpected absence of a robot is detected. However, when a robot is already executing a plan, the corresponding decision cannot be revised.

Our current implementation relies on a decomposition assumption. The value of u for a set of goals {*goal-1* ... *goal-n*} is a linear combination of (a) the sum over goals of the waiting times preceding the allocation of the goal to a robot and (b) the sum over goals of the utility of having the eventually chosen robot achieving the goal, independently of when it starts doing it. This means we want to maximize the expected value of the expression $\alpha\Sigma$*waiting-time(goal-i)* + $\beta\Sigma$*utility(goal-i achieved as it is achieved)* in cases in which the utility of achieving a goal as it is achieved does not vary with the waiting time. For example, any function of the execution duration and energy consumption satisfies this requirement. In particular, the overall expression to maximize can be set to $-\Sigma$*waiting-time(goal-i)* $-\Sigma$*execution-time(goal-i)* which corresponds to minimizing the average response time between the generation and the satisfaction of a goal.

Given the decomposition assumption, the CTPS can easily compute the expected value of both terms given a set of decisions {*decision-1* ... *decision-n*} for {*goal-1* ... *goal-n*}. For each goal and each robot, the CTPS knows the plan instance the robot proposes to execute. With this plan instance and appropriate statistical information, the CTPS computes the expected utility of having the goal achieved as the robot proposes. Therefore, when the CTPS hypothesizes the assignment of a robot *robot-j* to a goal *goal-i*, it knows the utility of this assignment. Furthermore, it can estimate the waiting time for *goal-i* under this hypothesis since *robot-j* will start executing the plan instance as soon as the CTPS tells it to do so. The evaluation is slightly more complex when the CTPS hypothesizes that it will not assign *goal-i* to any of the connected robots. In the absence of any other information, the expected value of *utility(goal-i achieved as it will be achieved in the future)* equals the mean value the function *utility* took for similar goals achieved in the past. Similarly, the CTPS relies on statistical information to determine the sum of the waiting times of m un-assigned goals when n robots are busy and p robots expected soon.

An incremental algorithm is used to review proposals. When a new proposal arrives, the CTPS determines whether the current allocation remains the best. If it does not, then it is replaced by the new optimal allocation. If the CTPS receives other proposals in the meantime, they are reviewed one after the other. When there is no more proposal to review, the CTPS asks robots to execute plans with respect to the optimal allocation. Robots to which tasks are assigned respond with an acknowledgement and plan and execute the corresponding motions in a distributed fashion.

3.3 Motion Planning

The motion planner developed within the GOFER project rests upon three assumptions:

- The geometry of the environment is mainly defined by walls, machines and heavy furniture which do not (or rarely) move in time. Therefore, robots can be provided with an approximate map of the environment and refer to this map to plan motions "in advance".
- In most cases, two obstacles are either very close to each other (a machine may be against a wall) or sufficiently distant to make it easy for the mobile robots to move in the environment. This is not a very restrictive assumption because shop-floors, office environments, hotels, airports, are designed so that it is easy to move inside. For example, the width of a corridor is such that two mobile objects of an average shape and size can easily pass each other. An important consequence of this assumption is that we can compute an artificial road network on which robots will move whenever it is possible.
- In general, there is too much uncertainty about the places where two robots will meet to solve every possible interaction in advance. During execution, robots must apply procedures which ensure that no collision occurs with other robots or, more generally, with other unexpected obstacles.

In this context, we advocate a distributed approach where each robot is responsible for its own motion planning. The planning technique is based on the pre-computation of a road network. Each robot is provided with this network and uses an A* algorithm [13] to compute a path which follows this network. The pre-computation of the network and the motion planning algorithm are presented below. Motion coordination and collision avoidance are ensured at execution time with the help of "behavior rules" and "potential fields" discussed in section 3.4.

To design a road network, we first compute two trapezoidal decompositions of the workspace along two orthogonal directions.[3] Each cell is a trapezoid whose upper and lower edges are edges (or parts of edges) of obstacles (the surrounding rectangular frame is considered as a special obstacle). When a cell has at most one neighbor cell at each of its left and right sides, a road is designed through the cell to connect its left and right edges. Let us call these cells *C-cells*. Any sequence of contiguous C-cells is called a *channel*. The roads in the contiguous cells of a channel can be connected. This results in a road which connects the left side of the leftmost cell to the right side of the rightmost cell of the channel. We call *T-cell* a cell which has more than one neighbor cell at its left or at its right. Let us suppose that a given T-cell T has two neighbor cells at its right. In this case, two roads separated by an obstacle are connected to the right edge of T. A vertical road must be designed to connect these two roads. The key idea is that the decomposition into cells along the orthogonal direction provides us with a channel which overlaps the T-cell T. The road designed in this channel can be used to connect the two roads at the right of T. Therefore, the computation of the road network can be summarized as

[3] We restrict ourselves to a two-dimensional problem: the mobile robots are disks moving among polygonal obstacles without holes inside a rectangular frame. We describe the decomposition along the x axis. The same algorithm applies along the y axis.

the computation of roads inside channels along the x and y directions and the design of suitable intersections. Figure 2 provides an example of a road network generated in our laboratory. Each road is composed of two lanes. The lines between the obstacles represent the spines of these lanes.

To compute a path from an initial location to a goal location (as specified by a task-level plan), a robot first tries to find a path from the initial location to any road of the network and from any road of the network to the goal location. This results in the creation of "connection lanes" allowing the robot to reach some lanes belonging to the network from the initial location and to reach the goal location from some other lanes of the network. The road network is temporarily updated by including these connection lanes. Then the robot uses an A* algorithm to determine a path from the initial to the goal location by searching the graph whose vertices are the intersections between the lanes and whose edges are the lanes of the road network. The solution provided by this algorithm is a sequence $(L_{init} X_{init} L_1 X_1 ... X_{n-1} L_n X_{goal} L_{goal})$ of lanes L_i and crossroads X_i, such that L_{init} and L_{goal} are connection lanes and L_1 ... L_n are lanes of the road network.

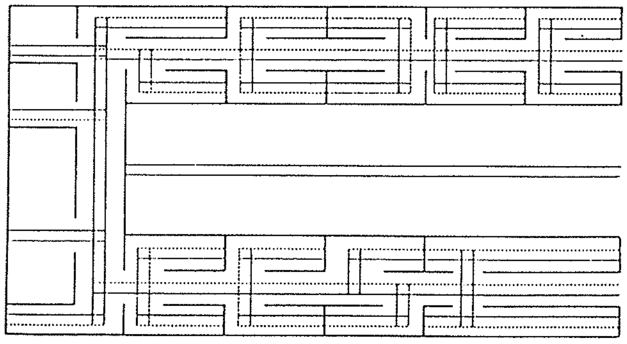
Figure 2: A Road Network

3.4 Execution

In the previous sections, we have been considering robot planning activities. We now present an execution system used to monitor execution and react to unexpected events. Hierarchical Petri nets (as originally developed with the SONIA scheduling system to simulate the execution of job-shop schedules [10]) are used to interpret the plan decomposition. Given a chosen plan instance, a robot generates a net $(S\ T)$ composed of states, action transitions and hierarchy transitions. Action transitions correspond to the actual performance of individual actions such as (connect myself port-22). Hierarchy transitions correspond to the interpretation of the plan decomposition. Figure 3 shows parts of the nets generated for an individual action, a sequence of actions, a set of un-sequenced actions and a set of exclusive alternatives.

Let us provide some background on Petri nets prior to explain how they are used to guide execution. Each state s in S can be provided with tokens. A state is active when it possesses at least a token. Each transition t in T is a couple $(in(t)\ out(t))$ of subsets of S. A transition t is active when every state s in $in(t)$ is active. An active transition can be selected for execution. States in $in(t)$ give a token when execution begins. States in $out(t)$ receive a token when execution ends. Resource states corresponding to individual or compound resources can be added to the net. A resource state s is in $in(t)$ if t consumes or require the use of a resource. A resource state s is in $out(t)$ if t produces a resource or gives it back at the end of the execution.

When the net is used to perform simulations (as in [10]), it is "timed" (as in [3]). Hierarchy transitions are instantaneous and durations of action transitions are computed with respect to statistical rules reflecting the stochastic nature of the action process. When the net is used to guide execution monitoring, its transitions are interpreted by the monitoring system. Interpreting a hierarchy transition t consists of (1) decrementing the number of tokens of each state s in $in(t)$ and (2) immediately incrementing the number of tokens of each state s in $out(t)$. Interpreting an action transition t consists of (1) decrementing the number of tokens of each state s in $in(t)$ and (2) initiating the control of the corresponding effectors. At some point in time (uncompletely predictable, depending on the stochastic nature of the action process), the robot is able to determine whether the action succeeds or fails. When the action succeeds, the robot increments the number of tokens of each state s in $out(t)$ and the process of interpreting transitions continues. When the action fails, the robot discards the net and determines other ways to achieve its goal.

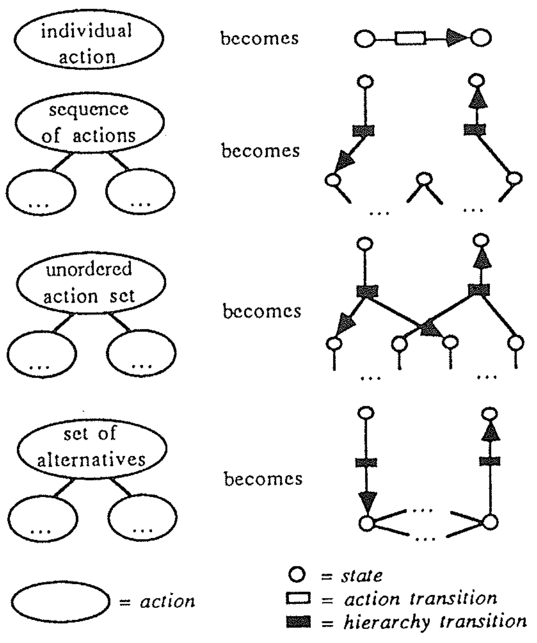

Figure 3: Derivation of Petri Nets

The control of the execution of an individual action may be a complex activity requiring the use of dedicated techniques. The most interesting example in the current system is robot motion. As mentioned in section 3.3, the method employed to coordinate the motions of several robots is to make them use behavior rules which implement space allocation policies:

- A robot must not enter a crossroad if there is another robot in the crossroad.
- A robot moving in a connection lane (L_{init} or L_{goal}) must stop when it detects another robot moving in a lane that the connection lane intersects with.
- A robot moving in a lane must keep a minimum distance between itself and any other robot preceding it in the same lane.

- A robot must never enter a crossroad X_i if there is not enough space in the next lane L_{i+1} to completely leave the crossroad X_i.
- A robot arriving at a crossroad must take note of all the robots already waiting on other lanes and wait for these robots to pass (this rule implements a first-come first-served crossroad allocation policy).
- A robot uses its motion planner again as soon as the total amount of time it spent waiting since the last crossroad becomes greater than a given constant. In this case, the robot tries to find a new path (from the current location to the goal location) in the road network in which the next crossroad of its current path is removed (this rule implements a solution to unlikely deadlocks [2]).

In addition, potential field techniques are used to avoid collisions with unexpected obstacles. When a robot moves inside a lane, we want it to stay as close as possible to its nominal path and to reach the next crossroad in its plan. These simple behaviors are implemented with attractive potential fields (similar to those described in [9]). But we also want the robot to stay in its lane and avoid collisions with known and unknown obstacles. These behaviors are implemented with repulsive potential fields (see [2] for details).[4]

4 Current Status - Future Work

We currently have three robots equipped with odometric, touch and infra-red proximity sensors. They are able to accomplish simple tasks such as pushing a box, tracking walls in a corridor and following each other. One of these robots will soon have a laser-ranging sensor. The current version of the planning and execution system is written in COMMON-LISP. Experiments with this system are performed on a DEC-3100 workstation with the help of a simulator designed to simulate actions of autonomous agents [12]. We are conducting experiments to check the efficiency of the task allocation strategy and determine in which cases a centralized analysis of task interactions allows to improve the behavior of the system.

Planning and scheduling algorithms will soon be experimented with the three robots. We will use the "action network" system [14] to control the use of potential fields during robot motion. Each robot will be provided with an action network to dynamically modify potential fields as the situation of the robot changes.

Other research issues include the development of models to trigger communication acts and the definition of an architecture to coordinate all the physical and cognitive activities of a robot.

- There are many cases in which appropriate reactions to unexpected events include communication. Hence a need to design methods allowing robots to determine (in a given situation) which pieces of information are worth getting (or spreading) and to plan communication acts accordingly. This includes the design and the representation of various communication operators and, most importantly, the design of mechanisms allowing error propagation to stop.[5]
- Another issue is the design (or the choice) of an architecture to coordinate the physical and cognitive activities of a robot. Currently, a blackboard-like system is used to coordinate planning and scheduling activities (see [11]), a Petri net interpreter deals with plan execution (cf. section 3.4), and action networks are considered to control the use of potential fields. There are a lot of similarities between the three systems. In the long term, we would like to define a unique control system to replace them, without sacrificing the advantages of each approach: explicit definition and implementation of control reasoning, efficiency and concurrency.

References

[1] G. W. Allport and L. Postman. *The Psychology of Rumor.* Russell and Russell, 1965.

[2] P. Caloud. *Distributed Motion Planning and Motion Coordination for Multiple Robots.* Working Paper, Stanford University, 1990.

[3] J. Carlier, P. Chrétienne and C. Girault. *Modeling Scheduling Problems with Timed Petri Nets.* Fourth European Workshop on Theory and Applications of Petri Nets, 1983.

[4] D. Chapman. *Planning for Conjunctive Goals.* Artificial Intelligence, 32(3):333-377, 1987.

[5] W. Choi, D. Zhu and J.-C. Latombe. *Contingency-Tolerant Robot Motion Planning and Control.* IEEE/RSJ International Workshop on Intelligent Robots and Systems, 1989.

[6] M. P. Georgeff and A. L. Lansky. *Reactive Reasoning and Planning.* Sixth National Conference on Artificial Intelligence, 1987.

[7] M. P. Georgeff and F. F. Ingrand. *Decision-Making in an Embedded Reasoning System.* Eleventh International Joint Conference on Artificial Intelligence, 1989.

[8] E. J. Horvitz, J. S. Breese and M. Henrion. *Decision Theory in Expert Systems and Artificial Intelligence.* International Journal of Approximate Reasoning, 2(3):247-302, 1988.

[9] O. Khatib. *Real-Time Obstacle Avoidance for Robot Manipulators and Mobile Robots.* International Journal of Robotics Research, 5(1):90-98, 1986.

[10] C. Le Pape. *Des systèmes d'ordonnancement flexibles et opportunistes.* Thèse d'Université, Université Paris XI, 1988.

[11] C. Le Pape. *A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling.* IEEE International Conference on Robotics and Automation, 1990.

[12] C. Le Pape. *Simulating Actions of Autonomous Agents.* Working Paper, Stanford University, 1990.

[13] N. J. Nilsson. *Principles of Artificial Intelligence.* Springer-Verlag, 1982.

[14] N. J. Nilsson. *Action Networks.* Rochester Planning Workshop: "From Formal Systems to Practical Systems", 1988.

[15] H. Van Dyke Parunak. *Manufacturing Experience With the Contract Net.* Fifth Workshop on Distributed Artificial Intelligence, 1985.

[16] R. G. Smith. *The Contract Net: A Formalism for the Control of Distributed Problem Solving.* Fifth International Joint Conference on Artificial Intelligence, 1977.

[17] G. L. Steele. *The Definition and Implementation of a Computer Programming Language Based on Constraints.* PhD Thesis, Massachussets Institute of Technology, 1980.

[4] Another approach to motion execution monitoring, which we investigate in parallel, is described in [5].

[5] Unless each robot carefully verifies critical pieces of information prior to propagate them, we can expect errors concerning important issues to propagate much quicker than less important errors (as for human rumors [1]). Error propagation is consequently an important problem in domains in which important errors can be made.

- A robot must never enter a crossroad X_i if there is not enough space in the next lane L_{i+1} to completely leave the crossroad X_i.
- A robot arriving at a crossroad must take note of all the robots already waiting on other lanes and wait for these robots to pass (this rule implements a first-come first-served crossroad allocation policy).
- A robot uses its motion planner again as soon as the total amount of time it spent waiting since the last crossroad becomes greater than a given constant. In this case, the robot tries to find a new path (from the current location to the goal location) in the road network in which the next crossroad of its current path is removed (this rule implements a solution to unlikely deadlocks [2]).

In addition, potential field techniques are used to avoid collisions with unexpected obstacles. When a robot moves inside a lane, we want it to stay as close as possible to its nominal path and to reach the next crossroad in its plan. These simple behaviors are implemented with attractive potential fields (similar to those described in [9]). But we also want the robot to stay in its lane and avoid collisions with known and unknown obstacles. These behaviors are implemented with repulsive potential fields (see [2] for details).[4]

4 Current Status - Future Work

We currently have three robots equipped with odometric, touch and infra-red proximity sensors. They are able to accomplish simple tasks such as pushing a box, tracking walls in a corridor and following each other. One of these robots will soon have a laser-ranging sensor. The current version of the planning and execution system is written in COMMON-LISP. Experiments with this system are performed on a DEC-3100 workstation with the help of a simulator designed to simulate actions of autonomous agents [12]. We are conducting experiments to check the efficiency of the task allocation strategy and determine in which cases a centralized analysis of task interactions allows to improve the behavior of the system.

Planning and scheduling algorithms will soon be experimented with the three robots. We will use the "action network" system [14] to control the use of potential fields during robot motion. Each robot will be provided with an action network to dynamically modify potential fields as the situation of the robot changes.

Other research issues include the development of models to trigger communication acts and the definition of an architecture to coordinate all the physical and cognitive activities of a robot.

- There are many cases in which appropriate reactions to unexpected events include communication. Hence a need to design methods allowing robots to determine (in a given situation) which pieces of information are worth getting (or spreading) and to plan communication acts accordingly. This includes the design and the representation of various communication operators and, most importantly, the design of mechanisms allowing error propagation to stop.[5]
- Another issue is the design (or the choice) of an architecture to coordinate the physical and cognitive activities of a robot. Currently, a blackboard-like system is used to coordinate planning and scheduling activities (see [11]), a Petri net interpreter deals with plan execution (cf. section 3.4), and action networks are considered to control the use of potential fields. There are a lot of similarities between the three systems. In the long term, we would like to define a unique control system to replace them, without sacrificing the advantages of each approach: explicit definition and implementation of control reasoning, efficiency and concurrency.

References

[1] G. W. Allport and L. Postman. *The Psychology of Rumor*. Russell and Russell, 1965.

[2] P. Caloud. *Distributed Motion Planning and Motion Coordination for Multiple Robots*. Working Paper, Stanford University, 1990.

[3] J. Carlier, P. Chrétienne and C. Girault. *Modeling Scheduling Problems with Timed Petri Nets*. Fourth European Workshop on Theory and Applications of Petri Nets, 1983.

[4] D. Chapman. *Planning for Conjunctive Goals*. Artificial Intelligence, 32(3):333-377, 1987.

[5] W. Choi, D. Zhu and J.-C. Latombe. *Contingency-Tolerant Robot Motion Planning and Control*. IEEE/RSJ International Workshop on Intelligent Robots and Systems, 1989.

[6] M. P. Georgeff and A. L. Lansky. *Reactive Reasoning and Planning*. Sixth National Conference on Artificial Intelligence, 1987.

[7] M. P. Georgeff and F. F. Ingrand. *Decision-Making in an Embedded Reasoning System*. Eleventh International Joint Conference on Artificial Intelligence, 1989.

[8] E. J. Horvitz, J. S. Breese and M. Henrion. *Decision Theory in Expert Systems and Artificial Intelligence*. International Journal of Approximate Reasoning, 2(3):247-302, 1988.

[9] O. Khatib. *Real-Time Obstacle Avoidance for Robot Manipulators and Mobile Robots*. International Journal of Robotics Research, 5(1):90-98, 1986.

[10] C. Le Pape. *Des systèmes d'ordonnancement flexibles et opportunistes*. Thèse d'Université, Université Paris XI, 1988.

[11] C. Le Pape. *A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling*. IEEE International Conference on Robotics and Automation, 1990.

[12] C. Le Pape. *Simulating Actions of Autonomous Agents*. Working Paper, Stanford University, 1990.

[13] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag, 1982.

[14] N. J. Nilsson. *Action Networks*. Rochester Planning Workshop: "From Formal Systems to Practical Systems", 1988.

[15] H. Van Dyke Parunak. *Manufacturing Experience With the Contract Net*. Fifth Workshop on Distributed Artificial Intelligence, 1985.

[16] R. G. Smith. *The Contract Net: A Formalism for the Control of Distributed Problem Solving*. Fifth International Joint Conference on Artificial Intelligence, 1977.

[17] G. L. Steele. *The Definition and Implementation of a Computer Programming Language Based on Constraints*. PhD Thesis, Massachussets Institute of Technology, 1980.

[4] Another approach to motion execution monitoring, which we investigate in parallel, is described in [5].

[5] Unless each robot carefully verifies critical pieces of information prior to propagate them, we can expect errors concerning important issues to propagate much quicker than less important errors (as for human rumors [1]). Error propagation is consequently an important problem in domains in which important errors can be made.