Multi-robot cooperation in the MARTHA project

Rachid Alami, Sarah Fleury, Matthieu Herrb, Félix Ingrand, Frederic Robert

Multi Robot Cooperation in the Martha Project

R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert *
LAAS/CNRS
7, Avenue du Colonel Roche, 31077 Toulouse CEDEX 04
E-mail: {rachid,sara,matthieu,felix,fr}@laas.fr

Abstract

The Martha[1] project objectives are the control and the management of a fleet of autonomous mobile robots for transshipment tasks in harbours, airports and marshaling yards. Our presentation focuses on one of the most challenging and key problems of the Martha project: the multi robot cooperation. Indeed, high level missions are produced by a Central Station and sent to robots. It is then up to the robots to refine their missions, to plan their actions and trajectories in the environment and to coordinate these actions and trajectories with the other robots. In particular, these coordinations occur in crossings, in lanes when unexpected obstacles require the robot to move in the opposite lane, and in open areas where robots need to synchronise their trajectories.

We present a general concept for the control of a large fleet of autonomous mobile robots which has been developed, implemented and validated in the framework of Martha.

Numerous researches have been conducted in the autonomous mobile robot field, nevertheless, the Martha project is the first one to add the multi robot cooperation capabilities to such a large fleet of robots.

The Martha robots demonstrate advanced autonomous features including non-holonomic motion planning, environment modelling, sensor-based obstacle avoidance, and decentralised cooperation schemes at mission and trajectory levels.

1 Introduction

The Martha Project Objectives are the operation and control of a large fleet of autonomous mobile robots (10-100) for containers transshipment tasks in harbours, airports and marshaling yards.

There are already places in the world where transshipment tasks are performed by Automated Guided Vehicles, such as the Delta Terminal of the Rotterdam Harbour. However, these sites have been designed to suit the AGV needs and capabilities: they rely on a centralised control, and on a grid based navigation.

*Authors list in alphabetical order.

[1] MARTHA: European ESPRIT III Project No 6668. "Multiple Autonomous Robots for Transport and Handling Applications"

One of the goals of Martha is to study the use of mobile robots with as little as possible centralised control[2], evolving in already existing open sites (i.e. sites not designed for this particular application and that can be traversed by other vehicles).

The key solution to this decentralisation problem is to give more *autonomy* to the vehicles to allow them to cope with unexpected events and obstacles, inaccurate environment model, other vehicles, and so on. This high level of autonomy is achieved using advanced sensor-based capabilities (for localisation, obstacle detection and modelling) as well as planning and deliberation between the robots through local communication and coordination.

In such context, the dynamics of the environment, the impossibility to correctly estimate the duration of actions (the robots may be slowed down due to obstacle avoidance or re-localisation actions, and delays in load and unload operations, and so on) prevent a central system from elaborating long or medium term efficient and reliable detailed robot plans.

A global plan based on an estimation of the durations of robot actions will very quickly be inapplicable or inefficient. Indeed, it is difficult to determine precisely when a robot will need a given resource and how much time it will use it. More than that, even the set of needed resources will depend on the current execution context. This is particularly clear when one considers non-holonomic robots in constrained environments.

A more flexible way would be to allow the robots to determine incrementally the resources they need taking into account the execution context. This is the basic idea which influenced the design of the complete system.

We first give a general presentation of the overall system architecture (§2). We then present the generic multi robot cooperation scheme that we propose and its application to a fleet of mobile robots (§3). Sections 4 and 5 provide a description of the robot architecture as well as the software tools that we have used to build the system and validate the concept. We then describe the testbeds and provide data obtained through illustrative runs (Section 6).

2 Overall Architecture

A Martha system is composed of a Central Station (CS) and a set of autonomous mobile robots able to communicate with each other and with the Central Station.

The CS as well as the robots make use of the same description of the environment for several purposes dealing with mission specification, robot navigation or multi robot conflict resolution.

Environment Model This model has been designed to enable the implementation of the multi robot cooperation scheme presented in this paper, and as such is very much linked to this approach. The model is a topological and geometrical representation of the environment.

An environment is a topological graph (Figures 2) of *areas*, *routes* and *crossings*. The areas contains docking/undocking *stations*. The routes are composed of *lanes*; crossing and lanes are then composed of *cells* which have a nominal (but not exclusive) direction. Cells, areas and stations have a geometrical description (polygonal regions) (Figure 1).

[2] Indeed, a completely centralised control requires robust and permanent communication capabilities between all the robots and the central station, while a decentralised approach only require local communication between the robots and a low bandwidth intermittent communication with the central station.

Besides, one can have a geometrical description of known obstacles as well as complementary data for localisation or docking purposes. The real environment may also contain unknown obstacles/objects which have to be taken into account on-line by the robots (detection and avoidance if possible).

Martha's Robots Missions Although one of the goal of the Martha project is to alleviate the burden on a Central Station (CS), one remains present. However, its role is mainly to plan the transshipment operations (which robot loads/unloads which container)[3] and the routes the robot *should* use (See Figure 3 for a mission example). The CS uses the topological model to plan these routes. The CS does not intervene in the robot plans coordination (such as in crossing or area), nor does it plan the precise trajectories which are executed by the robots. As a consequence, the communication bandwidth required between the robots and the CS is very low. Moreover, the computational power devoted by the CS to control the robot is far less important than the one used in a completely centralised application.

The Robot System and the Multi robot cooperation in Martha The robots receive their missions from the Central Station. From then on, the Robot Control System (RCS) is on its own to perform the mission. It has to refine the mission, to plan its routes and then its trajectories, to coordinate the resulting plans and trajectories with other robots and to execute all these actions, monitoring critical situations (such as unknown obstacles) and reporting unrecoverable action failure to the CS (mostly those requiring an operator assistance).

Martha is a large Esprit project with many partners[4]. The integration of the complete RCS workpackage, which is composed of several modules and software components (corresponding to all the necessary robot functionalities), is done at LAAS. Two of these modules, the Robot Supervisor and the Motion Planner, are also developed by LAAS. Nevertheless, to be able to test the entire RCS on real robots and also to emulate large fleets of robots, all the necessary software modules and components have been developed at LAAS (as replacement plug-ins of the one developed by our partners) to create a complete system.

3 Multi Robot Cooperation and Plan Merging Paradigm

The most challenging part of Martha is the autonomous coordination of the robots. Indeed, the central station only sends high level missions as the one shown on figure 3. It is then up to the robots to refine, plan and coordinate route sections and crossings use, as well as trajectories in open areas.

3.1 Basic Ideas

The basic idea of the approach we have chosen and implemented is that whenever a robot produces a plan which makes use of some kind of shared resources (cells, trajectories in areas), it advertises

[3] The transshipment operations planning problem, which remains under the responsibility of the CS is more or less a temporal allocation problem and is not presented in this paper.

[4] ECT, Framatome, Frankfurt Airport, FZI, Ikerlan, Mannesmann Demag Gottwald, PROMIP (LAAS/Midi Robots), Rol, SNCF.

it, and collects from other robots the plans which specify how they plan to use these resources, as well as the right (i.e. the necessary exclusive tokens) to perform its plan coordination. It then produces a coordinated plan and informs the other robots of events occurrence (cells exits or particular point traversal on a trajectory) it wants to be informed of.

More generally, planning and plan coordination can be classified along different strategies or choices.

Global versus local. When one plans actions for a fleet of robots, one can consider the whole fleet or limit the scope of planning to the sets of robots with conflicting actions. However, this global versus local tradeoff is only possible when dealing with a properly sized environment. If the number of critical exclusive resources is more or less equal to the number of robots, conflict resolution may, by propagation, involve the whole fleet. On the other hand, if the environment is properly sized, conflicts remain local, and the solutions are negotiated locally without disturbing the unconcerned robots.

Complete versus incremental. Similarly, one can limit the scope of the planning and plan coordination in time. When a mission is sent to a robot, it can plan (or try to plan) and coordinate the whole mission. But considering the execution hazards, it seems to be inefficient (not to say a waste of time and resource) to plan too far ahead. The plan coordination should be done continuously, to guarantee a fluid navigation, and slightly ahead, to avoid to over constrain the other robot plans and to break the coordinated plans too often.

Centralised versus distributed. This last aspect of the planning and plan coordination problem is where should it take place: on a centralised computer or on board the robots? This does not change the computing complexity of the treatment itself. However, in a centralised approach, all the data (which are mostly local) need to be sent to the central station, and therefore require a more reliable communication link with a higher bandwidth between the robot and the central station. Moreover, the proposed protocol can be implemented with a local communication between the robots.

Our approach and our contribution to the planning and plan coordination problem can be classified as *local*, *incremental* and *distributed*. Missions are sent to each robot which plans and coordinates on-board *(distributed processing)*, up to a small resources horizon *(incremental)*, using a distributed coordination approach involving only the robots which are concerned by the required resources *(local)*.

3.2 The Plan Merging Paradigm

We have devised a domain independent multi robot cooperation scheme called "The *Plan-Merging* Paradigm". It applies to systems that involve the simultaneous operation of several autonomous agents, each one seeking to achieve its own task or goal.

Let us assume a set of autonomous robots, which have been given, through a centralised system, a set of tasks or goals. These tasks or goals have a sufficient range and are sufficiently independent to allow the robot to carry them on. However, each robot, while seeking to achieve its task will have to compete for resources, and to comply with other robots activities.

The *Plan-Merging* Paradigm favours a fluid execution but also guarantees a coherent behaviour of the agents in all situations (including the avalanche of situations which may occur after an

execution failure) and a reliable detection of situations which call for a new task distribution process.

Whenever a robot elaborates a new plan, it has to validate it in the current multi robot context. This is done by collecting all the other robot plans and by "merging" its own plan with them. This operation is "protected" by a distributed mutual exclusion mechanism [16], and is performed *without modifying* the other robots plans, to allow them to continue their current execution. However, the robot which does the plan-merging may ask the other robots to inform it of the occurrence of particular events, so that it can synchronise itself on them. We call this operation, a *Plan-Merging Operation* (PMO).

Note that such an operation only concerns future (near term) robot actions. It can run in parallel (and with a slight anticipation) with the execution of the current robot plan.

3.2.1 The Plan Merging Protocol

The robots are equipped with a reliable inter-robot communication device which allows them to broadcast a message to all robots in the vicinity or to send a message to a given robot. Each robot processes sequentially the goals it receives, taking as initial state the final state of its current plan. Doing so, it incrementally appends new sequences of actions to its current plan.

At any moment:

- a robot has its own *coordinated plan* under execution. Such a *coordinated plan* consists of a set of actions and synchronisations (i.e. events to be signalled to other robots as well as events which are planned to be signalled by other robots).
- the "global plan" is the union of all current robot coordinated plans.
- an individual robot coordinated plan (as well as the global plan) is such that all possible resources conflicts between robots actions are solved by temporal constraints (precedence) between events (actions begin or end) and such that the resulting graph is a *directed acyclic graph* (*dag*) (Figure 4) .

When a robot has to plan, it uses the following protocol which we call the *Plan Merging Protocol* (Figures 4 and 5):

1. It asks for the right to perform a PMO and waits until it obtains it together with the involved coordinated plans (i.e. with common resources to share) of all the other robots (in our example on Figure 4, Robot 2 will receive Robot 1, 3, and 4 coordinated plans).
2. It then builds the *dag* corresponding to the union of all coordinated plans (including its own coordinated plan) (the union of the light grey plan on Figure 4)
3. It then tries to produce a new plan (dark grey on the figure) which can be inserted in the *dag*, *after* its current coordinated plan. The new plan insertion may only add temporal constraints which impose some of its actions to be executed *after* some time-points of other robots coordinated plans. Besides, the *dag* property of the obtained global plan is maintained.
4. If it succeeds in producing the desired plan, the robot appends it to its current coordinated plan, and informs the other robots of the synchronisation it will need (Robots 1, 3 and 4 will inform Robot 2 of the occurrence of the hatched pattern events on the figure). We call these

synchronisation events between the robots: *execution events.* (because they occur during plan execution).

5. And finally, it releases the right to perform a PMO. Note that, even when a robot fails in its PMO, it leaves the global plan in a correct state (it is still a *dag* and the execution of the already coordinated plans can continue).

3.3 Situations where PMO is deferred or where a "deadlock" is detected

When a robot R_i tries to perform a PMO, it may fail to produce a plan which can be inserted in the set of collected plans. This means that the final state of at least another robot R_j (as it is specified in its current plan) forbids R_i to insert its own plan.

In such situation, R_i can simply abandon its PMO and decide to wait until the robots, that it has identified, have performed a new PMO which may possibly make them change the states preventing it to insert its plan.

Hence, besides *execution events* — i.e. events which allow robots to synchronise the execution of several robot plans —, we introduce *planning events* — i.e. events which occur whenever a robot performs a new PMO. These events can also be awaited for. They establish a temporal order relation between robots plan-merging activities, noted $Wait\_for\_PMO(R_i, R_j)$.

When R_i concludes that it has to wait for R_j to perform a successful PMO, it informs R_j. Each Robot maintains and propagates a graph (called *Planning Dependency Graph* of robots waiting for it (directly or by transitivity) to perform a PMO.

When a robot R_i succeeds in performing a PMO, it informs its immediate successors (if any) and discards its graph. But if it fails, it has to determine the set of robots from which it has to wait *planning event*. A "deadlock" occur if one of these robots is already waiting (directly or by transitivity) for R_i to perform a PMO.

When a deadlock occurs, it is necessary to take explicitly into account, in a *unique planning operation*, the conjunction of goals of all the robots involved in the cycle. One can decide to allow the robot which detected the deadlock, to plan for all the concerned robots. The Plan-Merging paradigm remains then applicable: the inserted plan will then concern several robots at a time. Again a PMO for several robots at a time, instead of one, may fail leading, in very intricate situations, to more and more aggregation until one reaches a completely centralised system (see [18] for a detailed discussion). At this point, if the last planning robot cannot find a solution, it means that the problem is infeasible.

Here we must recall that we do not claim that the Plan-Merging paradigm can solve or help to solve multi-robot planning problems.

The main point here is that the Plan-Merging paradigm is *safe* and guarantees a coherent behaviour. It favours fluid distributed planning and execution activities and it limits the use of more intricate and more time consuming planning operations to situations where they are strictly necessary.

The paradigm and the protocol presented so far is generic. We believe that it can be used in numerous applications. Several instances of the general paradigm can be derived, based on different planners: action planners in the stream of STRIPS, as well as more specific task planners

or motion planners. One class of applications which is particularly well suited is the control of a large number of autonomous mobile robots. We present in the sequel how this can be done.

3.3.1 Application to a fleet of mobile robots

For the case of a number of mobile robots in a route network environment, we have devised a specific *Plan-Merging Protocol* based on spatial resource allocation (see [1]). It is an instance of the general protocol described above, but in this context, *Plan-Merging Operation* is done for a limited list of required spatial resources: a set of cells. The robot broadcasts the set of required cells and receives back only the set of coordinated plans from other robots which have already planned to use some of the mentioned cells.

One of the most interesting property of this protocol is that it allows several PMOs to be performed simultaneously if they involve disjunctive resource sets. This is particularly useful when there are several local conflicts at the same time as it is often the case in a route network like environment.

Plan-Merging for cell occupation: In most situations, robot navigation and the associated Plan-merging procedure are performed by trying to maintain each cell of the environment occupied by at most one robot. This allows the robots to plan their trajectories independently, to compute the set of cells they will cross and to perform Plan-Merging at cell allocation level.

In order to optimise cells resource allocation and to minimise crossing obstruction, the allocation strategy is to allocate one cell ahead when the robot moves along lanes, while it allocates all the cells necessary to traverse and leave the crossing.

When reasoning about cells is not sufficient: While, most of the time, the robots may restrict their cooperation to cells allocation, there are situations where this is not enough. This happens when they have to cross large (non-structured) areas or when an unexpected obstacle, encountered in a lane or in a crossing, forces a set of robots to manoeuvre simultaneously in a set of cells. In such situations, a more detailed cooperation (using the same protocol but a different planner: the motion planner) takes place allowing robots to coordinate their actions at trajectory level. Thus, we have a hierarchy of PMOs: first, at the cell level. Then, depending on the context, at trajectory level: motion planning in a set of common cells determined by the first level. This hierarchy authorises a "light" cooperation, when possible, and a more detailed one, when needed.

3.3.2 Examples of Plan Merging Operation

We shall now present two examples to illustrate the use of the Plan-merging paradigm. The first one is a sequence of PMOs at Cell Level.

Example 1: Coordination at cell level (Figure 6).

- **Step 1:** This snapshot shows the involved cells of the environment.
  The robot destinations are the followings:
  - $r0$ and $r1$ on the right go to cell $C8$ above the crossing using cell $C4$.

- $r2$ and $r3$ at the bottom right traverse the crossing to reach the left cell $C0$ using cells $C5, C4$ and $C2$.
- $r6$ goes from left to the right cell $C7$ using cells $C3$ and $C5$.
- $r4$ goes from up to the lower cell $C10$ using cells $C2$ and $C3$.

The PMOs have occurred in the following order: $r0$ then $r2$ and then $r6$ in parallel with $r1$ (because $r6$ and $r1$ have disjunctive lists of resources) and finally $r4$.

- **Step 2:** The following synchronisations have been planned: $r2$ on $r0$ (which frees $C4$), $r6$ on $r2$ (which frees $C5$) and $r1$ on $r2$ (which frees $C4$), $r4$ on $r2$ (which frees $C2$) and $r6$ (which frees $C3$)

- **Step 3:** One should note that at this stage, $r3$ PMO fails because $r2$ has not yet planned an action to free the cell $C0$.

Example 2: Trajectory level (Figure 7) The second example illustrates PMO at trajectories level in a large open area with two obstacles in the middle, and 10 docking/undocking stations. In such an environment, there are no cell allocations (the robots are all in the same cell), all synchronisations are made at trajectory level.

Figure 7 shows a situation where all the robots have planned and coordinated a complete trajectory. The trajectories displayed on the figure are the one which have been sent by the robots for execution display.
• The robot destinations are: $r0$ goes to station 9, $r4$ to station 5, $r1$ to station 1, $r28$ to station 7, $r27$ to station 0, $r20$ to station 4.
• PMOs were done in the following order: $r1$, $r4$, $r27$, $r20$, $r28$ and $r0$.
• One can see that synchronisation hold for the following robots: $r4$ on $r1$, $r27$ on $r1$ and $r4$, $r20$ on $r1$ $r27$ and $r4$, $r28$ on $r27$ $r20$ and $r1$, $r0$ on $r1$ $r4$ $r28$ and $r27$.

4 Robot Control System

The RCS includes all the functional modules as well as the robot supervisor which, among other things, runs the Plan-Merging protocol and algorithms presented above.

4.1 General Architecture

As shown on Figure 8, the RCS is layered out on two levels: the Decisional Level also called the Robot Supervisor (RS), and the Functional Level composed of a set of functional modules.

Each module of the Functional level is a *server*: it provides services, through request/reply messages exchanges and with shared data structures (called *posters*), related to its function. On the other side, the RS is a *client* of most of these servers to which it sends requests when needed or access posters when necessary.

4.2 Decisional Level: The Robot Supervisor

The Robot Supervisor (RS) consists of three layers, corresponding to a hierarchical decomposition of planning and supervision activities (Figure 8): the Mission layer, the Coordination layer and the Execution layer.

The first two layers are themselves composed of a planning and a a supervision process. All tasks run in parallel and satisfy different response time constraints.

1. The Mission layer: The Mission layer deals with mission refinement and control. Mission refinement is performed through the context dependent instantiation of a library of "predefined plans" or "plans skeletons". A mission is first refined as if the robot was alone. The resulting plan is a sequence of actions (including planned trajectories) annotated with cell entry and exit monitoring operations which will be used to maintain the robot execution state and to synchronise its actions with other robots. The Mission Supervisor is in charge of controlling the execution of such plans. If a plan fails to achieve a particular goal, alternative plans are refined and attempted.

2. The Coordination layer: produces and controls "coordinated plans". It performs Plan-merging operations (presented in Section 3) and manages the interactions with the other robots (exchanging coordinated plans and events). The result is a "coordinated plan" which specifies all trajectories and actions to be executed, together with all events to be monitored and sent to other robots or to be awaited from other robots.

3. The Execution layer is in charge of the interpretation and the execution of coordinated plans. As a result, it is responsible of most interactions with the functional level. Besides all actions and monitors included in the plan, it also monitors and reacts to a number of critical events, such as unexpected obstacles in its path, or its own status (battery or fuel level), failure reports from the different modules, as well messages sent by the other robots (information about synchronisation events or plan failures).

4.3 Functional Level: The Modules

The actions planned by the decisional level are managed by the functional level. The organisation of the functional level and of its integration in the overall control architecture results from LAAS studies [4]. Indeed, the capabilities required by autonomous mobile robots cover a large variety of functionalities including strong real time constraints. In this project for instance each robot should provide both autonomous and coordination capabilities, which require the following functionalities:

- motion execution and feedback control,
- obstacle detection and avoidance,
- obstacles modelling,
- landmarks identification and localisation (numerical and logical),
- trajectory planning (in mono and multi robot contexts),
- communication with the central station and the other robots.

Moreover, all these functions should be integrated homogeneously in the system to be observable and controllable by the decisional level.

All these modules have been built using a tool developed at LAAS (GenoM, see §5.1) which allows us to describe and specify the modules with a high level formal language.

The functionalities listed above have been gathered into eight modules (Figure 8). The main ones are described below.

4.3.1 Motion Execution Modules

The motion execution modules are in charge of the close loop motion control of the robot. Two modules have been set.

The first one ensures the position feedback control along given trajectories. This module estimates the instantaneous robot position, to be controlled according to the reference position, by integration of the elementary displacements measured from proprioceptive sensors (odometer, gyroscope, inertial central, ...). This position is also exported in the "current position" poster at the others system components disposal.

The second module allows dynamical avoidance of unpredicted obstacles. The obstacles are detected with range sensors, such as ultra sonars or laser range finder. The avoidance consists in doing a deformation of the given trajectory under the influence of a potential field [11].

It is important to note that the trajectory should remain in a bounded area specified by the supervisor (the allocated cells or a corridor along the initial trajectory) in order not to interfere with the other robots. The failures, due to bulky obstacles, must be managed by the supervisor. The recovery generally consists in a modelling of the obstacle followed by a new multi robot trajectory planning taking into account the added obstacles.

The sequences of trajectories dynamically computed by this second module are sent to the first one which ensures the position feedback control.

The Hilare robots use odometry to compute their position with a probabilistic uncertainty. One of them is also equipped with a gyroscope. The range data for obstacle avoidance are acquired with a belt of ultrasonic sensors.

4.3.2 Perception Module

The perception module has two main functions: (1) localisation of the robot in its environment, and (2) the modelling of new obstacles.

The robot position exported by the motion module, computed on the basis of the elementary displacements, accumulates errors. Thus this position should be updated from time to time relatively to the environment. The difficulty is to design a localisation procedure that can be used in various structured environments without specific instrumentation like transponders or beacons. This procedure uses pre-existing landmarks present in the environment (buildings, furniture, ...). The geometry and the position of these landmarks are given to the robot. The localisation is based on a matching of segments extracted from acquired images with segments from the landmark models [3].

The obstacles modelling is directly the result of the segmentation of the acquired image. These obstacles are exported in the "perceived obstacles" poster to be used by the other modules. In particular, the motion planner module will integrate these obstacles in the environment model to

compute the next trajectories. This procedure is required by the supervisor when an unavoidable obstacle has been detected by the sensor based motion execution module.

For real experimentations, the images are obtained by an horizontal scan with a laser range finder. However, one of the three robots is not equipped yet with a laser and its localisation uses external cameras on the ceiling.

4.3.3 The Motion Planning Module

Each robot is equipped with an independent Motion Planner composed of a Topological Planner, a Geometrical Planner and Multi Robot Scheduler. It is used in order to compute not only feasible trajectories but also synchronisation events between different robot trajectories.

The Topological Planner: performs a search in the graph of cells in order to determine the set of cells to be used for a given motion task. The selection of cells may be done in two modes: a *Local Obstacle Avoidance* mode that selects only the cells which correspond to the nominal way of traversing lanes or crossings (i.e. according to the oriented topological graph), and an *Extended Obstacle Avoidance* mode which takes into account the obstacles exported in the "perceived obstacles" poster. A major obstacle can force the robot to leave its current lane and to use cells belonging to a "parallel" lane.

The Geometrical Planner: computes a geometrical path between an initial position and a goal position with respect to the allocated cells and the non-holonomic constraints of the robot[5]. When other robots share common resources, the computed path avoids the last positions of these robots. The perceived obstacles are of course taken into account. This planner produces Reeds and Shepp like paths -segments and arcs of circle- and it is based on techniques similar to those described in [12].

The Multi Robot Scheduler: determines, along a trajectory, the synchronisation points with the trajectories of the other robots: the positions where a robot should update its set of occupied resources, the positions where it should signal a trajectory synchronisation *execution event* to another robot, and the positions where it should stop and wait for synchronisation. The paths of the others robots are obtained by the communication modules and passed on through the "multi robot plans" poster (Figure 8).

Figure 7 shows geometrical paths with synchronisation points computed by this module.

5 Tools and Software environment

A number of generic tools were used to setup the RCS. All modules in the functional level were developed with the help of the GenoM development tool. The supervisor is written in C-PRS, a procedural reasoning system.

5.1 The Generator of Modules GenoM

GenoM is a generic tool which allows a programmer to specify and write functional modules to be integrated in a real-time control environment (such as the ones described in section 4.3) at a high

[5] The dynamic of the path - velocities and accelerations- are computed by the motion execution modules.

level of abstraction [6]. From the analysis of the structure, the behaviour and the interfaces of the modules, a *generic module* has been defined.

A description language allows to declare all the services managed by the module, their input and output parameters, the control graph of the associated computations and the algorithms linked to each step of the processing. The algorithms are usually developed by the expert of the module functionality. From this description, the module is generated by instantiating the *generic module* canvas. This generation produces:

- a module that can run on a Unix workstation (for emulation purpose) or on an embedded system under the *VxWorks* real time operating system.
- a library of interface functions for invoking the services and accessing to the data exported by the module; such a library will be linked with the clients of the module, for instance the supervisor or other modules.
- an interactive program to run preliminary tests.

An intersecting aspect of this approach, especially in projects involving many functionalities and many developers, is that all the modules have the same logical behaviour and homogeneous interfaces, avoiding fastidious preliminary tests. Moreover, each module declaration file constitutes a complete document of the interface and of the logic the module at the disposal of the developers which can then elaborate their modules independently.

At last, modules can service the robot supervisor but also other modules. This possibility allows us to design a dynamic hierarchy of modules and elaborated composed functions (reflex actions —i.e. actions associated to monitors—, multi levels feedback control, and so on).

5.2 PRS

PRS [9, 8] is composed of a set of tools and methods to represent and execute plans and procedures.

A PRS kernel is composed of three main elements:
- a **database** which contains facts representing the system view of the world and which is constantly and automatically updated as new events appear.
- a **library of plans (or procedures, or scripts),** each describing a particular sequence of actions and tests that may be performed to achieve given goals or to react to certain situations.
- a **tasks graph** which is a dynamic set of tasks currently executing.

Tasks are dynamic structures which execute the "intended plans", they keep track of the state of execution of these intended procedure, and of the state of their posted subgoals. For example, in a mobile robot application, the task graph could contain the tasks corresponding to various activities the robot is performing (one activity to refine its current mission, another to monitor incoming messages from a central station giving orders, another one managing the communication layer with low level functional modules, etc).

An interpreter manipulates these components. It receives new events (from outside and from asserted facts) and internal goals, checks sleeping and maintained conditions, selects appropriate plans (procedures) based on these new events, goals, and system beliefs, places the selected procedures on the tasks graph, chooses a current task among the roots of the graph and finally executes

one step of the active procedure in the selected task. This can result into a primitive action, or the establishment of a new goal.

In PRS, *goals* are descriptions of a desired state associated to a behaviour to reach/test this state. The possible goals are the goals to **achieve** a condition, to **test** a condition, to **wait** for a condition to become true, to passively **preserve** and to actively **maintain** a condition while doing something else.

PRS is used to implement the Decisional Level of the RCS (Figure 8). An interface has been developed to use the modules of the Functional Level. It allows us to access posters, to send requests to modules (with arguments encoding) and to receive asynchronous replies which results can be decoded and interpreted.

The complete Decisional Level of the RCS runs about 16 tasks in parallel and uses about 350 PRS procedures.

5.3 Data collection and display tools

The display tool (Figures 6, 7 (2D version) and 12) is basically a display server on which each robot is connected and updates its position at a high rate. This server has a model of the environment and shows in real time the evolution of the fleet. Runs can be recorded and replayed at a later time for a finer analysis (to check for example that trajectories are indeed collision free, or to analyse the synchronisation done).

Other tools record the various communications taking place between the robots and allows us to make statistical analysis on the number/size of messages exchanged and to evaluate the minimum required communication bandwidth.

6 Implementation and Testbeds

All software components were developed and run under Unix *and* VxWorks. The demonstration on a large number of emulated robots under Unix was necessary to validate our approach, while using our three mobile robots demonstrated the effectiveness of this approach on real robots and the possibility to install all the necessary functions on board the robots.

6.1 Emulation Testbed

To validate the proposed RCS architecture and the PM paradigm, we developed an emulation testbed (Figure 9) which includes an emulation of the Central Station, and the display tool to visualise the evolution of the emulated robots.

To thoroughly test our approach, we have build approximatively fifteen different environments with a great variety of features: indoor versus outdoor; small (10-100 meters) versus large environment (100-500 meters); small (1-2 meters) robots versus large (17 meters) container carriers; large open areas versus complex graphs of areas/lanes/crossing; environments with and without unknown obstacles, and so on. We also modelled the real environments used for the demonstrations of the project (held at LAAS (multi robot), Trappes-SNCF (heavy load), Frankfort Airport (medium load)), so we could test/tune and validate them (by changing the lanes/crossings/cells breakdown and position) before running the real robots on them.

Overall, the emulation testbed is very useful as it allows us to:
- make thousands of hours of realistic experimentation,
- run a much larger fleet of robots than the one we currently have...
- test and debug the RCS, the modules and the PMO paradigm,
- test and tune the environment description choices (size of areas, number of cells in crossing, and so on),
- validate trajectory execution (some environment models can be too constrained to allow a robot to manoeuvre),
- validate the size of a fleet for a given environment.

Some numerical results Running a fleet of ten robots during thirty minutes on a large outdoor environment slightly more complex than the one presented on Figure 1 generates about 4024 messages between the robots which can be classified into 416 PMO requests, resulting into 3744 responses. 128 coordinated plans are exchanged resulting into 48 synchronisations between executable plans. 32 wait for planning are generated. 15 K-bytes of compressed data are exchanged over the robot network ($<$ 10 bytes per second).

Another example also involving ten robots for the same duration, in a more constrained indoor environment resulted in 10168 messages exchanged (note the increase of the number of messages), including 928 PMO requests which led to 8352 responses. Due to the small number of cells, and large areas, 220 PMO request conflicts arose. 936 coordinated plans were exchanged, 176 cell synchronisation and 104 trajectories synchronisation were done. 219 plan dependencies were managed and led to 439 plan update messages. 100 K-bytes of compressed data were exchanged over the robot network ($<$ 60 bytes per second).

6.2 Real Robots Testbed

Emulation testbeds are of great utility to validate the approach on a large fleet. However, whenever perception is involved, nothing can replace experiments with real robots and real sensors. This is why we choose to also conduct some experiments with three indoor robots of the Hilare family (Figure 11).

The RCS of each robot is composed of the same decisional level as used in the emulation (the same C-PRS procedures are loaded in both cases), and a functional level integrating the specific modules developed in the project (§ 4.3) with the existing functional modules. All softwares are executed on-board.

Our experiment room (which is about 10×7 meters large) has been structured into two areas including six docking stations and two lanes, according to the environment model presented in Section 2. In this environment, we have conducted runs where the robots keep going for more than two hours. In a typical run, during one hour, one robot:

- covers a cumulated distance of 300 meters,
- exchanges 900 messages with the other robots,
- executes 250 coordination operations which yield to 70 synchronisations at the trajectory level and 20 at the resource level.

- the decisional level produces 1500 requests to the functional level.

The high number of coordinations observed here is a consequence of the small size of the environment compared to the size of our robots. But it fully demonstrates the capabilities of our decisional level.

7 Related work

There are numerous contributions dealing with multi robot cooperation. However, the term "co-operation" has been used in several contexts with different meanings.

We will not consider here contributions to cooperation schemes at servo level (e.g. [14]) nor contributions which aim at building an "intelligent group" of simple robots (e.g. [15]). We will limit our analysis to contributions which involve an effective cooperation (at plan or program level) between several robots.

Several approaches have been proposed, such as generation of trajectories without collision (e.g. [5, 19]), traffic rules [7, 10], negotiation for dynamic task allocation [13, 2], and synchronisation by programming [17, 21].

Inter-robot communication allows to exchange various information, positions, current status, future actions, etc [2, 21, 20] and to devise effective cooperation schemes.

Traffic rules have been proposed as a way to allow several robots to avoid collision and to synchronise their motion (with limited or even without communication). However, many aspects should be taken into account in order to build the set of traffic rules: the tasks, the environment, the robot features, and so on. This entails that the generated rules are valid only under the considered assumptions. If some of them are changed, the rules have to be modified or sometimes be regenerated completely. Besides, these systems are generally built heuristically and do not provide any guarantee such as deadlock detection.

Negotiation have been used for dynamic task [2] or resource [13] allocation to several robots on a situation-dependent basis.

Most contributions which make use of synchronisation through communication are based on a pre-defined set of situations or on task dependent properties.

Indeed, most of the methods listed here, deal essentially with collision avoidance or motion coordination and cannot be directly applied to other contexts or tasks.

We claim that our Plan-Merging paradigm is a generic framework which can be applied into different contexts, using different planners (action planners as well as motion planners). It has some clean properties (and clear limitations) which should allow us, depending on the application context, to provide a coherent behaviour of the global system without having to explicitly encode all situations that may occur.

8 Future Developments and Prospective

By the time this paper is published, the Martha project will have ended. Nevertheless, LAAS remains committed to this research field and plans to continue to work in this area. In particular,

we are studying extensions of the PM protocol to minimise the waiting time, particularly in trajectory coordination. Such extensions could be implemented by:

- taking explicitly into account the time, so that when plans are being coordinated, the robot which is doing the PMO does not always insert its action *after* the other robots conflicting actions, but possibly *before*.
- or relying on a bounded response time layer which could then be used to locally perform a first arrive first cross trajectory.

Other improvements of the paradigm are being considered and currently implemented (convoy mode, better robustness, dynamic add/remove of robots in the protocol) which will overall improve the performance of the system.

Besides the investigation of other classes of applications and the work on a more formal description of the proposed approach, our future work will also concentrate on developing new cooperation schemes by embedding a multi robot planning activity.

9 Conclusion

We have presented a general approach for the control of a large fleet of autonomous mobile robots which has been developed, implemented and validated in the framework of the Martha Esprit project. The most challenging and key problems of the Martha project — the multi robot cooperation — has been addressed with a "generic" approach called *Plan-Merging Paradigm*.

The Plan-Merging paradigm has the following properties;

1. It makes it possible for each robot to produce a coordinated plan which is compatible with all the plans executed by other robots.
2. No system is required to maintain the global state and the global plan permanently. Instead, each robot updates it from time to time by executing a PMO.
3. The PMO is safe, because it is robust to plan execution failures and allows us to detect deadlocks.

We believe that it can be applied to a large variety of contexts and with different planners (from action planners to task or motion planners), and at different granularities.

Such a multi robot cooperation scheme "fills the gap" between very high level planning (be it centralised or distributed) and distributed execution by a set of autonomous robots in a dynamic environment.

Indeed, it appears to be particularly well suited to the control of a large number of robots navigating in a route network. The application that we have implemented clearly exhibits its main features. It allowed us to make a large number of autonomous robots behave coherently and efficiently without creating a huge activity at the central system.

We have presented the architecture, the robot supervisor, the functional modules as well as the various tools we use to implement the Robot Control System.

The Martha project "multi robot" demonstration took place at LAAS with the three Hilare robots and with thirty emulated robots. Moreover, the complete RCS and the LAAS modules

were used for the other demonstrations of the project, held on the Commutor robot at Trappes (SNCF) and on the Indumat robot at the Frankfort airport.

References

[1] R. Alami, F. Robert, F. F. Ingrand, and S. Suzuki. Multi-robot cooperation through incremental plan-merging. In *IEEE International Conference on Robotics and Automation, Nagoaya, (Japan)*, 1995.

[2] H. Asama, K. Ozaki, et al. Negotiation between multiple mobile robots and an environment manager. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, pages 533–538, June 1991.

[3] H. Bullata and M. Devy. Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot. In *IEEE International Conference on Robotics and Automation, West Lafayette, USA*, April 1996.

[4] R. Chatila. Deliberation and reactivity in autonmous mobile robots. *Robotics and Autonomous Systems*, 18(2-4), December 1995.

[5] H. Chu and H.A. EiMaraghy. Real-time multi-robot path planner based on a heuristic approach. In *IEEE International Conference on Robotics and Automation, Nice, (France)*, pages 475–480, May 1992.

[6] S. Fleury, M. Herrb, and R. Chatila. Design of a modular architecture for autonomous robot. In *IEEE International Conference on Robotics and Automation, San Diego California, (USA)*, 1994.

[7] D.D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, 4(5):491–497, Oct. 1988.

[8] F. F. Ingrand, R. Chatila, and R. Alami F. Robert. Prs: A high level supervision and control language for autonomous mobile robots. In *IEEE International Conference on Robotics and Automation, St Paul, (USA)*, 1996.

[9] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, December 1992.

[10] S. Kato, S. Nishiyama, and J. Takeno. Coordinating mobile robots by applying traffic rules. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '92), Raleigh (North Carolina, USA)*, pages 1535–1541, July 1992.

[11] M. Khatib and R. Chatila. An extended potentiel field approach for mobile robot sensor-based motions. In *Intelligent Autonomous Systems (IAS'4), Karlsruhe (Germany)*, 1995.

[12] J.-C. Latombe. A fast path planner for a car-like indoor mobile robot. In AAAI Press, editor, *9th Natl. Conf. on Artificial Intelligence*, 1991.

[13] C. Le Pape. A combination of centralized and distributed methods for multi-agent planning and scheduling. In *IEEE International Conference on Robotics and Automation, Cincinnati, (USA)*, pages 488–493, May 1990.

[14] Z.W. Luo, K. Ito, and M Ito. Multiple robot manipulators cooperative compliant manipulation on dynamical environments. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '93), Yokohama, (Japan)*, pages 1927–1934, July 1993.

[15] M. Mataric. Minimizing complexity in controlling a mobile robot population. In *IEEE International Conference on Robotics and Automation, Nice, (France)*, pages 830–835, May 1992.

[16] M. Naimi, Trehel M, and A. Arnold. A log(n) distributed mutual exclusion algorithm based on path reversal. *Journal of Parallel and Distributed Computing*, 34, 1996.

[17] F.R. Noreils. Integrating multi-robot coodination in a mobile-robot control system. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '90), Tsuchiura (Japan)*, pages 43–49, July 1990.

[18] S. Qutub, R. Alami, and F. Ingrand. How to Solve Deadlock Situations within the Plan-Merging Paradigm for Multi-robot Cooperation. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '97), Greboble, (France)*, September 1997.

[19] T.Tsubouchi and S.Arimoto. Behavior of a mobile robot navigated by an "iterated forecast and planning" scheme in the presence of multiple moving obstacles. In *IEEE International Conference on Robotics and Automation, San Diego California, (USA)*, pages 2470–2475, May 1994.

[20] J. Wang. On sign-board based inter-robot communication in distributed robotic systems. In *IEEE International Conference on Robotics and Automation, San Diego California, (USA)*, pages 1045–1050, May 1994.

[21] S. Yuta and S.Premvuti. Coordination autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '92), Raleigh (North Carolina, USA)*, pages 1566–1574, July 1992.

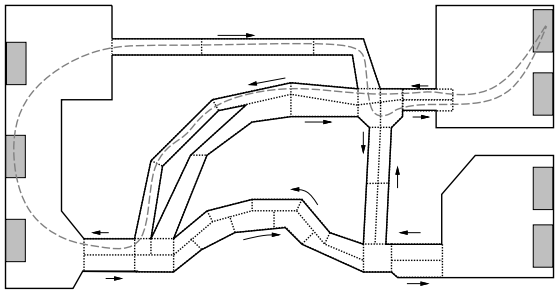


Figure 1: An Environment: Geometrical Representation.

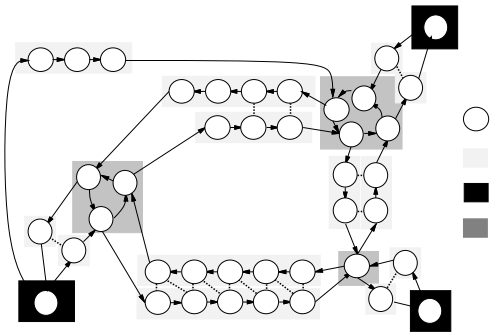
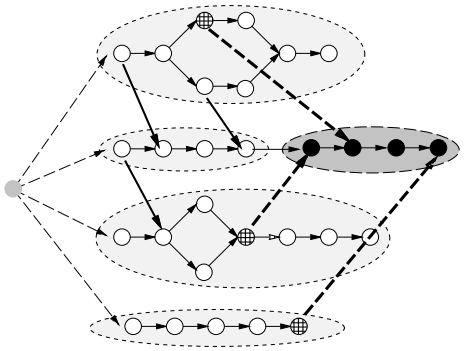


Figure 2: Topological Representation.

```
; Starting from station 1.
(mission (.
   (action 1 (goto (station 3))
             (using (lane 10) (lane 1) (lane 11)))
   (action 2 (dock))
   (action 3 (pick-up (container 5)))
   (action 4 (undock))
   (action 5 (goto (station 1))
             (using (lane 13) (lane 9)))
   (action 6 (dock))
   (action 7 (putdown))
   (action 8 (undock)) .))
```

Figure 3: Example of a mission sent by the Central Station to a robot



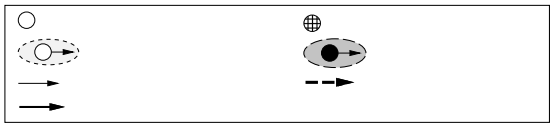


Figure 4: Part of the "global plan" *dag* during the insertion of a new plan by robot-2.

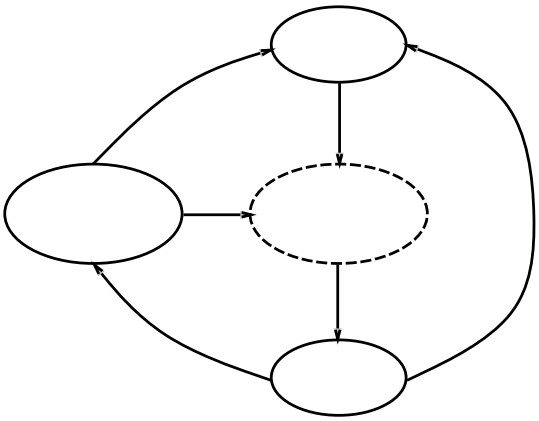


Figure 5: The Plan Merging Protocol

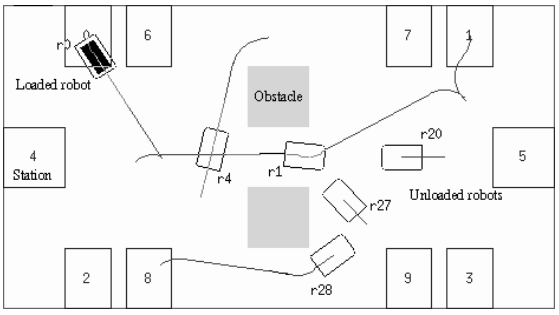


Figure 7: **Plan-merging at trajectory level.**

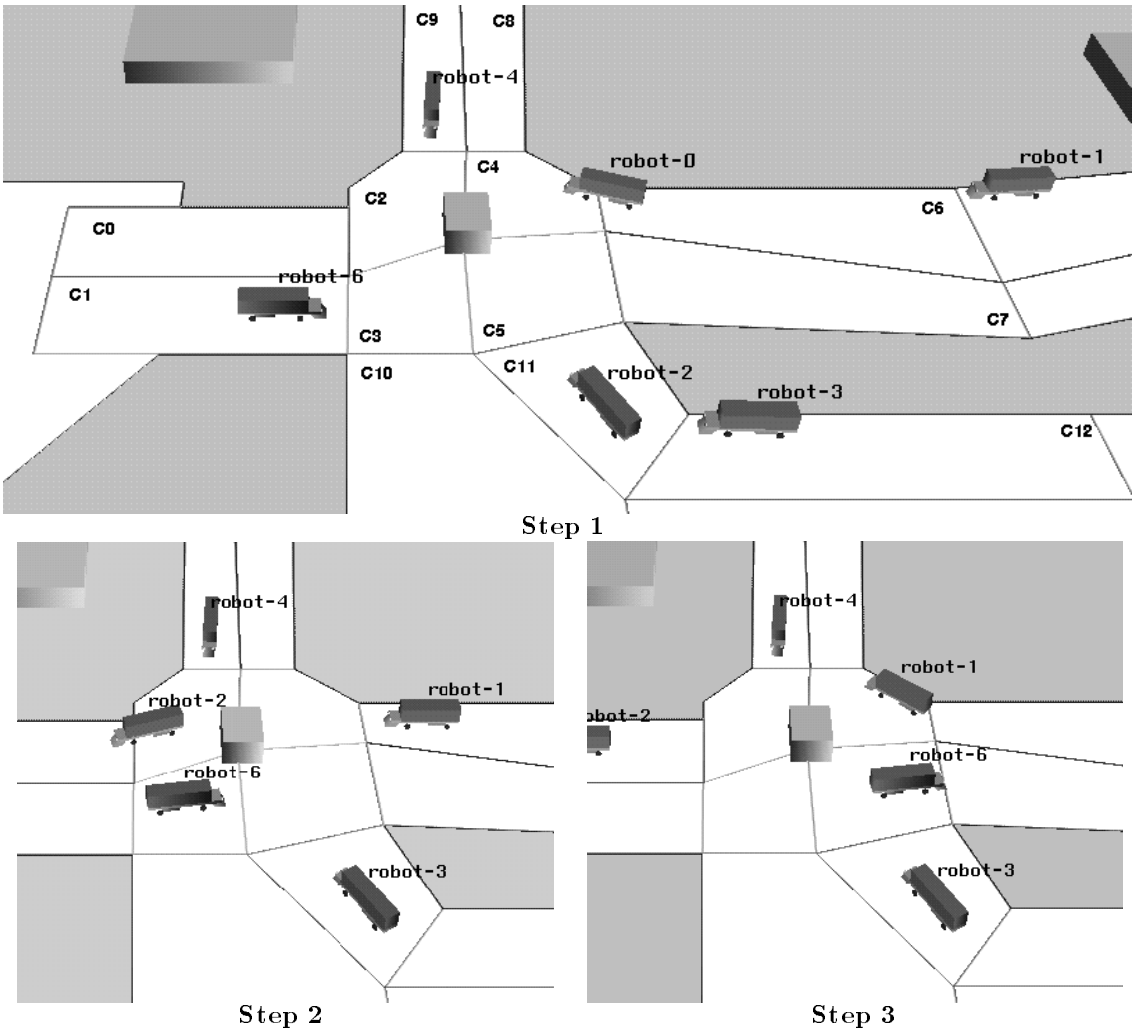


Figure 6: Plan-merging at cell level.

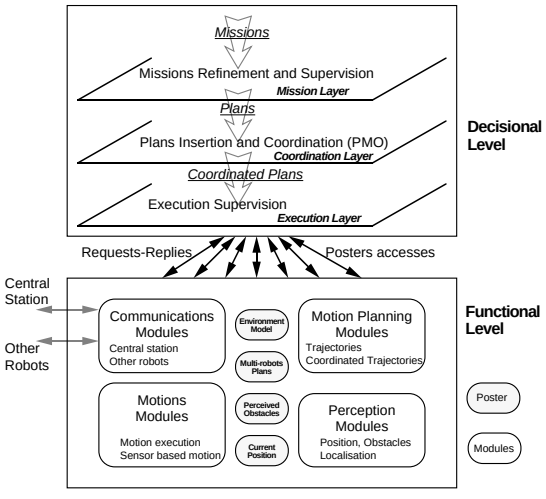


Figure 8: The Robot Control System Architecture

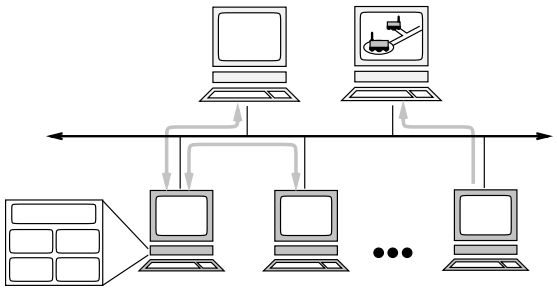


Figure 9: The Emulation Testbed: n Unix workstations implementing n robots

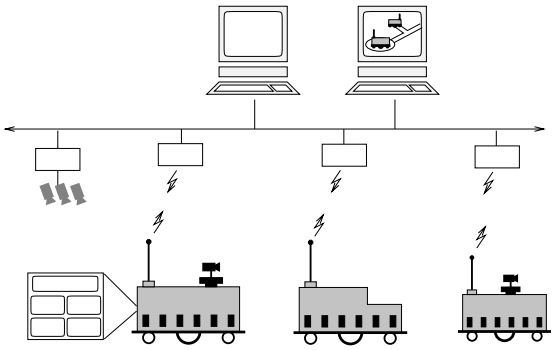


Figure 10: The Experimental Testbed

Figure 11: The Three Hilare Robots in Mission

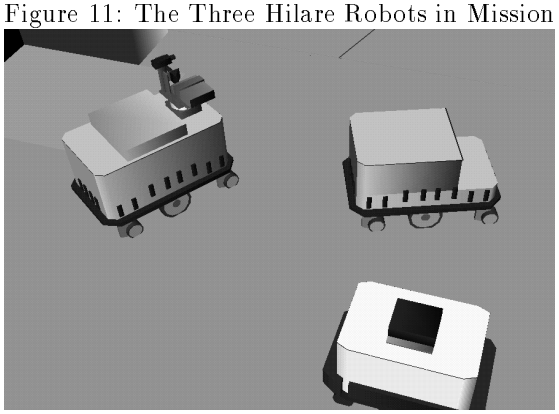


Figure 12: The same situation Viewed with the Display Server