15th Triennial World Congress, Barcelona, Spain

A FLEXIBLE SIMULATION APPROACH FOR MULTIROBOT SYSTEMS

A. Cruz, V. F. Muñoz, A. García-Cerezo

Dpto. De Ingeniería de Sistemas y Automática, Universidad de Málaga.
Plaza El Ejido s/n, 29013 Málaga, Spain.
Phone: (+34) 5 213 14 06 Fax: (+34) 5 213 14 13
E-mail: {anacm,victor,agcerezo}@ctima.uma.es

Abstract: A significant advantage of multirobot simulation lays on the fact that their physical features and environment can be simulated, and hence, difficulties that spring when managing them are avoided. This paper shows a flexible method for simulating multirobot systems. It provides a simple and non proprietary structure of classes, that can be used as the backbone of complex developments, adapted to every multirobot system needings. The approach is based on discrete events systems, so it can simulate the inherent concurrency associated to multirobot systems. The paper presents results obtained when this solution is implemented using MATLAB, a numeric computation tool. *Copyright © 2002 IFAC*

Keywords: autonomous mobile robots, discrete event systems, vehicle simulators, concurrency, trajectory planning.

1. INTRODUCTION

A multirobot system can be defined as an environment inhabited by some robotic elements, such as mobile robots, manipulators, or any kind of robotic mechanism. Elements of a multirobot system can cooperate in doing some task, they can compete for resources in the environment, they also can be independent from the rest of elements, or they can communicate with them... The common point is that, whenever a multirobot system is designed or developed, not only isolated features for every component must be taken into account, but possible interactions among them have a weighty role in the whole system behaviour.

Working with a real multirobot system demands several conditions: an environment with all the required functionality (for example, a conveyor between manipulators, carrying some kind of material), a set of robots ready to do their job, maybe communications between them... To sum up, a set of potentially complicated features, that makes developing a multirobot system not always easy or comfortable. In these cases, simulation seems to be the way to do things, as long as it makes feasible to work with the system without having all its elements physically available, in an simpler and friendly setting.

Multirobot systems have a particularity: they are concurrent systems. This means that maybe a few actions happen at the same time (e.g., a mobile robot moves towards a conveyor while a manipulator picks a piece of wood up from this conveyor). Thus, any simulation model for a multirobot system should reflect this concurrency aspect.

This paper proposes a general approach for simulating multirobot systems; though it is designed for running on a non multitask processor, it keeps the concurrency constraint previously commented. It does not intend to be a closed or complete solution. Several commercial, shareware or freeware tools can be currently found: Webots, Kephera Simulator, JavaBots/TeamBots (Balch and Ram, 1998),

Rossum's Playhouse RP1, Mission Lab, and some others. However, using a standard solution is not always straightforward, since it is very difficult that fits to every single feature of a physical multirobot system. The simulation approach presented in this paper is flexible, because offers the possibility of integrate different elements depending of the multirobot system to be simulated. Furthermore, it is a non proprietary solution, and then it is not bound to a programming language or a software/hardware platform. The method is a general and simple solution, developed using OO techniques, that can adapt to a number of situations; it can be seen as the basis for generating more complex systems, for those cases when adapting to a concrete way of working, such as a standard application, is not as effective as building a particular solution.

The plan of the paper is as follows. Section 2 describes the multirobot system that has given rise to the proposed method. Section 3 analyses some important aspects and features of the solution. Finally, experiments performed with a MATLAB implementation of the method, and final conclusions, are presented on section 4.

2. A MULTIROBOT SYSTEM

The purpose of this section is to present a real work situation that can require a simulation tool as the proposed one. This system is composed of several mobile autonomous vehicles that travel inside a common environment. Thus, navigation is the main point to be considered. The approach by Muñoz, et. al (1999) has been used for solving it, and it is explained along this section.

Muñoz (1999) defines navigation as the methodology of driving the course of a mobile robot while it traverses the environment, with the main goal of guiding the vehicle safely, i.e., without crashing into anything. For this navigation task, the motion control system of the mobile robot uses the position and speed references (trajectory) computed by a global planner, that takes into account the problem specification and a map of the world, so this trajectory should be the one to be secure.

The trajectory calculated by the global planner can be seen as the composition of a spatial plan, or path, and a temporal plan, or speed profile. Both references can be used by the robot's motion control system for tracking the trajectory. Next subsections show how such references are obtained.

2.1 Computing the spatial plan.

The goal of the spatial planner is to work out a free obstacle path, with special geometric properties, from a starting location to an ending one: a spatial plan. In this way, this path must prove the mobile robot non-holonomic restriction and avoid discontinuities in the vehicle's steering. For solving these questions, the planner considers two separate stages: the route planning and the path generation.

A route is defined as a set of subgoals which must be reached sequentially by the robot, in order to achieve the final location. Two consecutive subgoals are always visible (Latombe,1991) and the whole route builds a obstacle-free path which does not contemplate the physical limitations of the vehicle to follow it. However, these limitations have a significant influence for the ability of the robot's path tracking system to follow the route. Many different techniques can be used for planning a route by modelling the robot's environment, such as a configuration space, a Voronoi diagram, or cell decomposition. Afterwards, the planner applies a heuristic search for coming to the goal location by minimising a certain criterion (such as the smallest route length).

A continuous curvature curve is fitted to the computed route and sampled into a stream of robot postures in order to obtain a spatial path with the desired geometric properties for path execution. This process is named path generation and uses a planned route as entry. There are several methods which provide continuous curvature paths but have the disadvantage of lacking a closed-form expression, causing their computational requirements to be unfeasible for real time applications. However, generation methods based on β-Splines have demonstrated their efficiency when a smooth and continuous curvature path is needed, with a small computational cost.

This spatial planner returns, as a path Q, a set of postures $q_i=\{x_i,y_i,\theta_i,\kappa_i,s_i\}$; every posture in the path is defined by a pair of spatial components (x_i,y_i), an orientation component θ_i, a curvature component κ_i, and a distance component s_i.

2.2 Computing the speed profile

Most of path planning methods assume that the robot executes the path at a low and constant speed. This assumption means that the vehicle's dynamic features do not affect the precision of the path tracking algorithm when the vehicle is following the path. However, a speed profile definition along the path is necessary in many applications. There are some methods that generate a speed profile that also eludes mobile obstacles (Kant and Zucker, 1986), that could be used in the multirobot case, but they do not take into account the vehicle's dynamics, so avoidance is not completely guaranteed. Thus, in order to obtain the final speed profile, three stages are applied:

- First, a speed planner process assigns a speed value to certain main points of the path provided by the path planner. This value is obtained from the set of speed restrictions that act on the vehicle (mechanical, cinematic, dynamic and operational constraints).

- The second stage is in charge of verifying the possibility of crashing into another mobile obstacles running in the environment, and if such situation happens, solve it by changing speed values obtained in the previous stage.
- Finally, using information provided for both precedent steps, the speed profile is generated as a continuous function. From this profile, acceleration and temporal components can be calculated almost directly.

So, the final trajectory is built by adding speed values to every path posture acquired from the path planner. In other words, a trajectory Q' is composed by postures $q'_i=(x_i,y_i,\theta_i,\kappa_i,s_i,v_i,,a_i,t_i)$, where v_i, a_i and t_i components are taken from the generated speed profile.

3. MULTIROBOT SIMULATION APPROACH

This section deals with the main features and questions related to the implementation of the proposed solution. It is divided into three subsections: first one is devoted to the concurrency aspects that the simulation method must take into account; second subsection explains why an object-oriented approach has been chosen, and which are the benefits of this election; finally, third one describes the structure of the simulation method.

3.1 Concurrency

As it was stated in section one, any multirobot simulator must cope with the concurrency that is inherent to multirobot systems. The proposed approach solves this problem by means of a discrete events system; this kind of sytems are briefly presented in this subsection (Muñoz, 1998).

A discrete events system (DES) is a simulation method designed to solve systems with non deterministic behaviour due to random components. Its main characteristic lies in the fact that, for every step of the simulation, time increases arbitrarily.

A generic DES is conformed of the following elements:

- *Entities*: system components, like machines, pieces, customers,... They can be classified into *permanent* or *temporaries*.
- *Attributes*: entities features, that distinguish one from another.
- *Sets*: a collection of entities, usually temporary ones. If they are related to a permanent entity, they are called *resources*.
- *Activities*: they model changes on the system, so it could be said that system function depends on a proper modelling of its activities. An activity is defined by a set of instantaneous happenings or events.
- *State*: attributes values, used to choose the next activity to be launched.
- *Scheduler*: it acts as a clock that marks what must be done at the current time; this means that activities are executed depending on scheduler values.

System run is simulated by the scheduler, according to the algorithm pictured in Figure 1. As the algorithm shows, every loop iteration corresponds to an event execution. Therefore, it is the events sequence who decides when changes happen. So, if events are properly ordered, concurrency can be verified.

In this way, the simulation method translates real multirobot functionality into a discrete event system, so concurrency aspects can be correctly simulated.

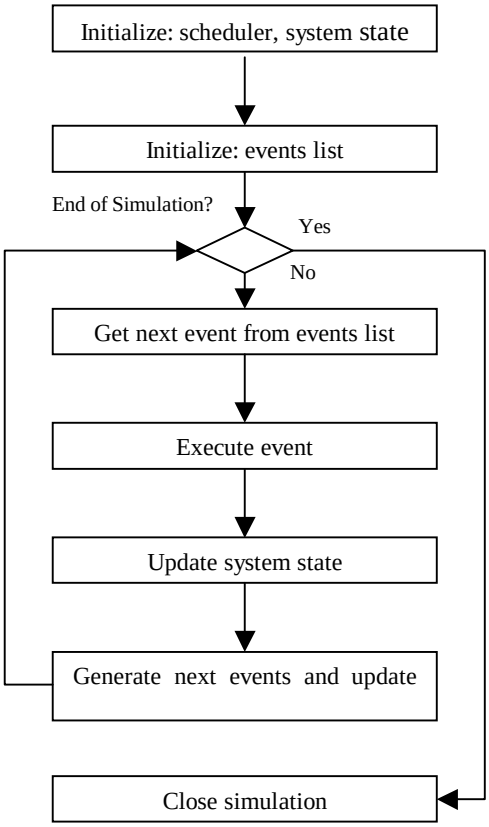

Fig. 1 Discrete Event System algorithm

3.2 Object-oriented programming

For analysis and design stages of this simulation method, an object-oriented approach has been used. Advantages of this methodology relay on the fact that it enables building quality software. Software quality can be measured through five main issues: correctness, robustness, extendibility, reusability and compatibility (Meyer, 1988):

- Correctness and robustness assure that the developed program fits to its specification, and that it also works in situations that requirements did not take into account. Both

concepts are usually joined under the term "reliability".

- Extendibility is reached if software can be easily adapted to changes on its specification.
- Reusability makes possible to reuse code in some other application.
- Compatibility means that the software can work along with other programs.

Object-oriented programming provides a set of techniques for writing code that verifies extendibility, reusability and compatibility. Correctness and robustness can be achieved using analysis and design methodologies that can be combined with the object-oriented approach.

A complementary tool to object-oriented programming is the Unified Modelling Language (UML). UML is a graphical language that helps the analysis and design phases, providing a set of diagrams that capture static and dynamic aspects of the system being developed. In other words, every diagram shows a different perspective of the system, which enhaces the general system overview. The approach presented in this paper has been developed using UML, and some of the generated diagramans will be shown in next subsections.

Furthermore, object-oriented programming features and UML also lead to a better project development (Cantor, 1998), which can be very important in case of building complex systems, or coordinating a team of developers.

3.3 Multirobot simulation structure

This subsection shows the structure of the simulation approach, after the analysis and design phases. It is explained via two UML diagrams: context level diagram, and class diagram. First one, pictured on Figure 2, shows relations among the elements of the multirobot system (a system monitor, and the robots), and some external agents, as the trajectory generator explained in section 2, or a user interface that helps to handle the system (other agents are feasible too, depending on the features of the multirobot to simulate). This means that the multirobot simulator will be composed of a number of robots and a monitor that controls the system; and it will have some kind of relationship with the trajectory generator, and the user interface.

One step beyond context-level diagram, the class diagram arises. The class diagram contains information about all the classes that conforms the whole system, and relationships between them. Figure 3 shows a class diagram for the multirobot simulator, that due to space constraints includes just the main classes of the system, i.e., those who are included into multirobot system package shown in context-level diagram.

A more detailed explanation of this classes is presented in the following:

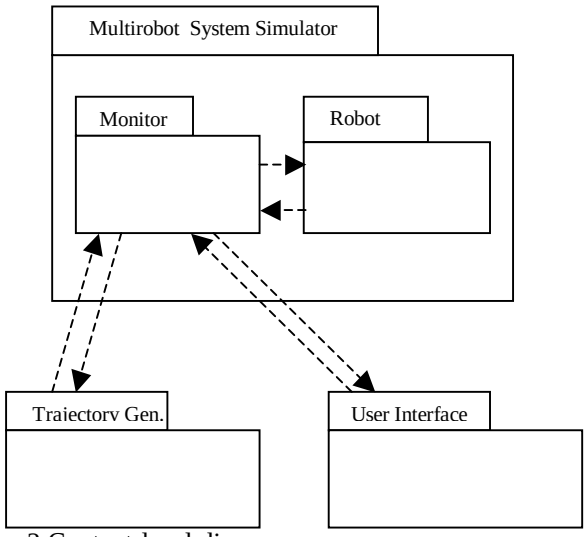

Fig. 2 Context-level diagram

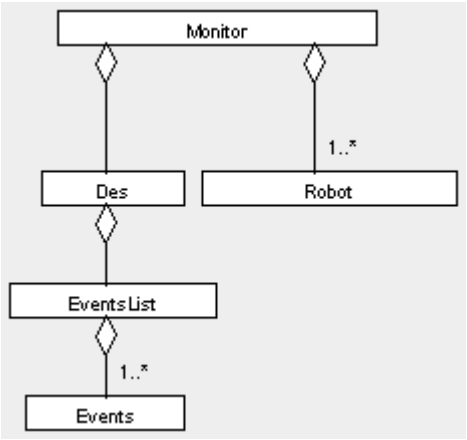

Fig. 3 Class diagram

- DES: discrete events system, that acts as the backbone of the simulation. Its properties are a *time*, a *list_of_events*, and a *current_event*. It is generated by the Monitor, who initializes these properties. This class includes operations that control how the DES moves forward and finishes.
- EventsList: this class represents the ordered list of events included in the DES. Its operations are related to updating and obtaining events from the list.
- Events: as it was mentioned in the previous subsection, events are the translation of the actions that happen during the multirobot system life into the simulator. They have been implemented as a class with three properties: *robot*, that stands for the robot that originates the event; *action*, that points which kind of action represents the event; and *time*, that marks the instant when the event occurs in the system (this attribute is used as a criterion for ordering EventsList). Different kinds of Events can be

implemented, depending on the particularities of the system being developed.

- Robot: the class that represents any robot in the system, including their main features. There should be an instance of the class for every robot in the system. If the multirobot system is composed of several kinds of vehicles, the Robot class should be an abstract one, so differences between robots could be taken into account through class inheritance. As it can be seen, the design of the Robot class depends heavily on the concrete multirobot system to be developed.

To sum up, the function of the whole simulator is controlled by the Monitor, that generates the DES using the information extracted from the robots. In every step of the DES, an event is analysed, and depending on its *action* property, an action is taken by the system.

Trajectory generator, user interface, or any other external agent are out of the scope of this section. They admit different solutions, and this election must be done upon the needs of the multirobot system that is going to be simulated. This means that the design is extensible, since it allows the designer to add different agents to the initial design. Furthermore, some other utilities classes, as file handlers and so on, have been implemented too.

4. EXPERIMENTS AND CONCLUSIONS

Previous section presented the structure of the simulation approach proposed in this paper. This subsection goes further and faces the final step of the whole process: the implementation of the design into a particular programming language.

Codification was done in a PC under Windows2000 using MATLAB 6 R12, the language for numeric computation; this software tool was chosen because it gives results in a fast and easy fashion, including good graphical utilities. However, any other object-oriented programming language (Java or C++), under different platforms (Linux, Windows, Lynx...) could be also used.

MATLAB 6 R12 allows object-oriented programming. A class is a set of M-files grouped under a directory that must have the same name as the class name. Every method in the class is implemented via an M-file. The class needs a constructor method, that initializes the class, and it is also desirable to implement the *set* (updating class properties), *get* (retrieving class properties) and *display* (printing class info and properties) methods. Main OO-programming features, such as overloading and inheritance are also supported.

An example of simulation using the proposed solution is commented. The experiment shows how two mobile robots (RAM-2 and Auriga-α, both developed at System Engineering and Automation Department), navigate simultaneously in the same environment. The task the physical system executes must be studied before the simulator is implemented: it has a relevant influence on the implementation, since it defines concrete key aspects of the simulator that cannot be fixed at the designing phase. In this example, since navigation is the task to be developed, every robot in the system should follow the previously computed trajectory: their motion control systems receive information extracted from the trajectory, and they follow these references. Every reference must be kept a certain time, depending upon the vehicle's features, in order to assure that the motion control system is able to reach it. Whenever any of these situations arises (moving to a new reference, or following a reference), it will have attached a time coordinate.

The simulator will be able to reflect both situations through the Events class. Then, Events belonging to this system will have these properties:

- *robot* will be set to "ram" or "auriga", depending on which robot is related to the event.
- *action* will be set to "follow", if the robot has received a new trajectory reference, or "move", if it is trying to reach a reference sent to the motion control system.
- *time* will be extracted from the trajectory postures, if *action* is "follow", or from the motion control system, if *action* is "move".

Once the requirements of the simulator have been studied, the implementation of the classes begins. Most of the cases need to write the Monitor and Robot classes; furthermore, in this example, due to the nature of the task to simulate, a TrajectoryGenerator class must be implemented. Next paragraphs describe how these classes have been developed.

The TrajectoryGenerator class must provide a trajectory, i.e., a set of references the robot can follow in order to reach its goal position. In this example, trajectories have been calculated using the methodology explained on section 2. Any other way of generating the references for the vehicle is valid; the only condition to verify is that such references must be understood by the robot when the trajectory is tracked.

The Robot class gathers the features of the robots belonging to the system. It can be developed under two different points of view. If robots in the system are similar, the idea would be to build a single class that contains the attributes and methods that characterizes the robots' function. On the other hand, if robots are quite different, an abstract Robot class would be generated, and for every robot in the system, an instantiation would be created. In this example, the first approach has been chosen. The developed Robot class contains a set of attributes (name, size, maximum velocity and acceleration, and control motion system information), and a set of methods

mainly devoted to the motion control system. These methods are able to interpret the references provided by the TrajectoryGenerator.
At last, the Monitor gets the information of each robot and generates, for every robot, two EventsList: one composed of "follow" Events, and another one of "move" Events; after that, it will merge both EventsList in a single one. Then, the Monitor will join all the robots EventsList objects in order to obtain a common EventsList, and it will initialize a DES with it. Then simulation will start, and it will run until the DES finishes.

Figure 4 presents the initial situation, with robots in their starting postures (RAM-2, the light grey octagon; Auriga-α, the dark grey ellipse). The paths they have to follow are drawn in light grey and dark grey, respectively. As it can be seen, both paths elude some static obstacles, pictured as rectangles, but there is a crossing between them in one point. However, speed profiles of the robots will not allow any collision.

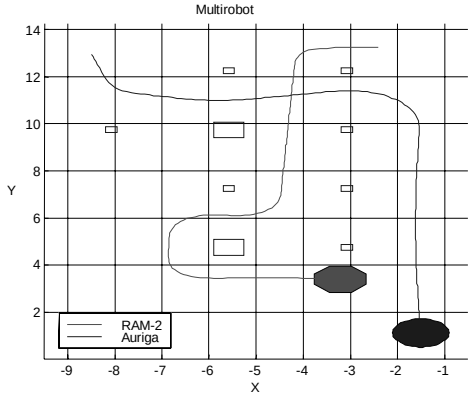

Fig. 4. Initial situation.

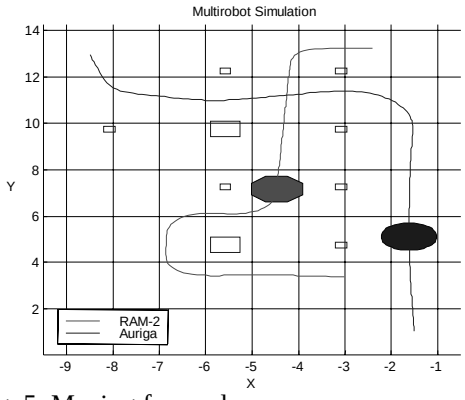

Fig. 5. Moving forward.

How both vehicles move forward along their paths is depicted in Figure 5. Figure 6 illustrates the crash avoidance, as RAM-2 reaches the crossing point before Auriga comes near it. At last, Figure 7 shows both robots in their final positions.

Finally, to set the conclusions of the described work, this paper presents a simulation approach for multirobot systems that offers some useful features: it is a general, flexible, simple and non-propietary method, based on a discrete events system in order to keep concurrency constraints, and designed usign object oriented programming techniques. It is intended to be the basis of more complex simulators that adapts as much as possible to the physical systems being simulated. This solution has been succesfully tested in the simulation of a multirobot system composed of RAM-2 and Auriga-α robots, both designed and built at the Systems and Automation Engineering Department, at Málaga University.

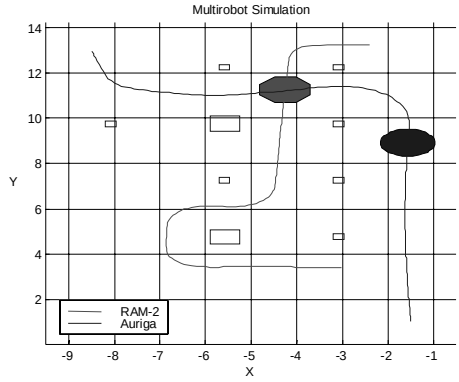

Fig. 6. Avoiding the collision.

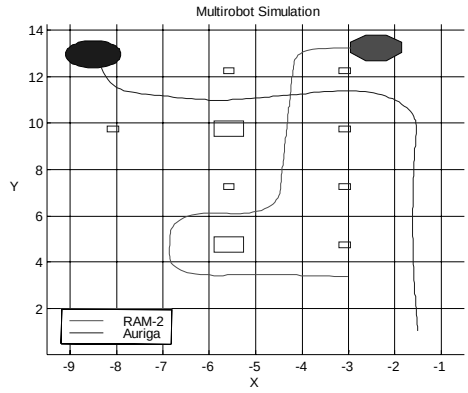

Fig. 7. Final posture reached.

REFERENCES

Balch, T. and A. Ram (1998). Integrating Robotic Technologies with JavaBots. *Working Notes on the AAAI 1998 Spring Symposium, Standford, CA.*

Cantor, M. (2000). *Object-oriented Project Management with UML*. John Wiley & Sons.

Kant, K. and S. Zucker (1986). Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *The International Journal of Robotics Research,* **Vol. 5 No. 3**.

Latombe, J. C. (1991). *Robot Motion Planning*, Kluwer Academic Publishers.

Meyer, B. (1998). *Object-oriented Software Construction*. Prentice Hall.

Muñoz, V. F. (1998). Sistemas de Eventos Discretos. *Class notes for Ph. D. Course "Técnicas de modelado y computacionales, y de análisis en ingeniería"*, System Engineering and Automation Department, Málaga University.

Muñoz, V. F., García-Cerezo A., Cruz A. (1999). Smooth Trajectory Planning Method for Mobile Robots. *Special issue on Intelligent Autonomous Vehicles of the Journal of Integrated Computed-Aided Engineering*, **Vol. 6 No. 4**, IOS Press, Netherlands