SIMULATING LOOSELY AND TIGHTLY COUPLED MULTI-ROBOT COOPERATION

Luiz Chaimowicz, Mario Campos

VERLab – DCC – Universidade Federal de Minas Gerais,
Belo Horizonte, MG, Brasil, 31270-901
{chaimo, mario}@dcc.ufmg.br

Vijay Kumar

GRASP Lab. – Univ. of Pennsylvania, Philadelphia, PA, USA, 19104
kumar@grasp.cis.upenn.edu

Abstract— This paper presents MuRoS, a simulator for multi-robot cooperation. MuRoS allows the simulation of different robots in several types of cooperative tasks, ranging from tightly coupled to loosely coupled tasks. We show the implementation of two of these tasks in the simulator: Forage and Cooperative Manipulation, and discuss important results related to cooperative robotics such as scalability, communication and hybrid control.

Key Words— Cooperative Robotics, Multi-robot simulation

1 Introduction

The use of multi-robot teams has received much attention in the last few years. The basic idea is to have groups of robots working cooperatively to execute different types of tasks. The use of multiple robots increase the overall reliability of the system, while decreasing the task complexity and execution time. Groups of simpler and less expensive robots can be used instead of expensive specialized robots in the execution of several applications.

Also, there are certain kinds of tasks, named **Tightly Coupled Tasks**, that cannot be executed by a single robot and require the use of multiple robots working cooperatively. To execute tightly coupled tasks, the robots must act in a highly coordinated fashion and this normally requires some knowledge about the states and actions of the teammates, either through implicit (sensory perception) or explicit communication. Further, in many cases, each robot is critical to the task and an individual failure could cause failure of the task as a whole. These are different from the **Loosely Coupled Tasks**, that can be executed by a single robot alone, but have better performance when a team of robots is used. When executing loosely coupled tasks, the robots can act independently from each other and strict coordination is not required. Surveys of cooperative robotics applications can be found in (Cao et al., 1997) and (Parker, 2000).

Good simulation tools are very important in the study of cooperative robotics. Normally, it is difficult and expensive to get and maintain multi-robot teams, mainly when a large number of robots (more than 10 for example) are required. Multi-robot simulators allow researchers to obtain results rapidly and to implement and test different approaches before starting real implementations.

This paper presents MuRoS[a], a Multi-Robot Simulator that can be used for simulating various types of tasks, ranging from loosely coupled to tightly coupled cooperative tasks. Developed using object orientation in the MS Windows environment, MuRoS has a very friendly user interface and can be easily extended with the development of new classes and the implementation of new robots, controllers and sensors. Used alone or in conjunction with implementations in real platforms, the simulator has allowed the study of different aspects of cooperative robotics in several application domains, such as: cooperative manipulation (Chaimowicz et al., 2001), robot formations (Fierro et al., 2001) and robot control (Das et al., 2001).

There are some general simulators for cooperative robotics in the literature. For example, Teambots (Balch and Hybinette, 2000) is implemented in Java but use only the behavior based paradigm to control the robots. Other simulator is Stage, that simulates mobile robots moving in and sensing a two-dimensional bitmapped environment, controlled through a device server called Player (Gerkey et al., 2001). An extension that groups a previous version of Stage with a wireless network simulator is described in (Ye et al., 2001). Another interesting approach is CHARON, a tool for modelling and analyzing hybrid systems that can be used to simulate multi-robot coordination and control (Alur et al., 2000). General software tools such as Matlab and Simulink (Mathworks, 2001) have also been used to simulate some specific cooperative robotic applications. The main advantage of MuRoS is that it allows the implementation of various types of robots with different characteristics, thus the user is not restricted to

[a] http:\\www.verlab.dcc.ufmg.br\muros

use a specific kind of robot or controller given by the simulator. Also, its friendly user interface and good performance makes MuRoS a powerful tool for simulating cooperative robotics.

This work also presents two test bed applications developed and tested on the simulator. The first one is a Forage task, in which a group of robots must search an environment looking for items and collect these items taking them to a specific goal. It is a classical example of loosely coupled cooperation. The second is a cooperative manipulation task, where the robots must collaborate to transport a large object between two different locations in the environment. It is an example of a tightly coupled task, that cannot be executed by a single robot. These applications allow us to study several important aspects of cooperative robotics such as scalability, communication and hybrid control.

This paper is organized as follows: next section gives a general overview of the simulator, including the class hierarchy, controllers and communication. Section 3 presents the forage application while Section 4 shows the cooperative manipulation task. Finally, section 5 brings the conclusion and possibilities for future work.

2 MuRoS: a Multi-Robot Simulator

2.1 General Description

MuRoS is an object oriented simulator developed in Visual C++ for the MS Windows environment. It allows the simulation of several multi-robot applications such as cooperative manipulation, formation control, foraging, etc. Both loosely coupled and tightly coupled tasks can be simulated. It has a friendly user interface that allows the instantiation of different types of robots, the creation of obstacles (circles, rectangles and polygons) and the observation of the simulation in real time. Also, result data can be exported to other tools such as Matlab for future analysis. New types of robots can be implemented with different controllers, driving mechanisms and sensors, inheriting characteristics from other robots. Explicit communication can be simulated, allowing the robots to exchange information during the task execution. New applications can also be developed making the simulator a very powerful tool. Figure 1 shows a snapshot of the simulator with 4 nonholonomic robots carrying an object, 10 robots moving towards the goal and 3 obstacles.

The simulator has a very good performance because it uses two different threads: one for the integration of dynamic equations and other for the real time display. As expected, the performance decreases as the number of robots increase, because more equations must be updated in each simulation step. An important feature is that several simulation parameters, such as the integration step and controller constants, can be changed dynamically during the execution. This adds flexibility to the system and makes the simulations more efficient.

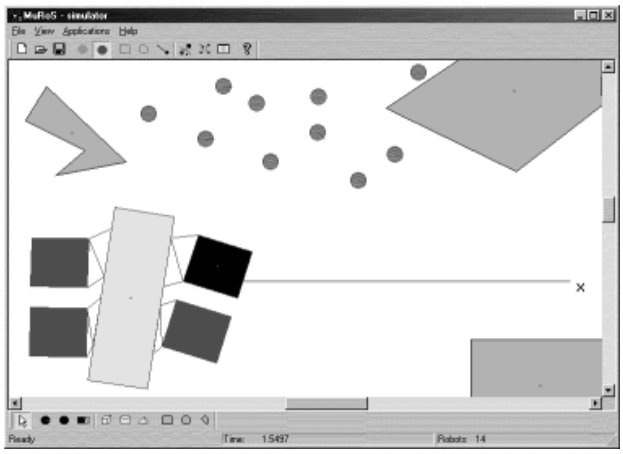

Figure 1. Snapshot of MuRoS interface

2.2 Class Hierarchy

One of the fundamental concepts in objected oriented programming is inheritance. The inheritance mechanism allows the developer to create base classes with general characteristics of an entity and specialize them in new subclasses, with the addition of members, methods and the redefinition of virtual functions. It is a very important concept, mainly when code reuse and extensibility are necessary.

In MuRoS, we have implemented a class hierarchy that allows the creation of new types of robots inheriting characteristics from classes that had already been developed. The main class is CRobot that contains the basic members and methods of a generic robot. In the first level of inheritance, we have the subclasses CRobotHolonomic and CRobotNonHolonomic, each one with its proper characteristics. From them, we can derive more specific classes, for example: CRobotForage with robots for the forage task or CRobotLabmate representing a TRC Labmate platform equipped with a compliant arm (Sugar and Kumar, 1998). Basically, to create a new type of robot, the user will have to inherit from one of the CRobot* classes, redefine some of its functions and add methods and data members for new capabilities such as new sensors, controllers and planners.

The class hierarchy also include different types of obstacles, objects and a class for mapping and planning, that is used by some of the robots. Figure 2 shows the class hierarchy diagram. Some internal classes of the simulator (for example the class that controls the user interface) are not shown in this figure.

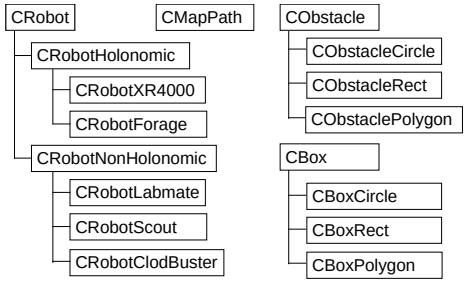

Figure 2. Class hierarchy of the simulator

2.3 Controllers

Different types of controllers can be implemented in the simulator. A distributed hybrid approach is used, where each robot can switch among different discrete states (modes), having different continuous controllers in each mode. Thus, the robots can dynamically change their behavior during the task execution, adapting better to changes and unexpected events in the task and on the environment.

Basically, the user can define the dynamic equations that control the robots in each mode, and the simulator integrate these equations displaying the robots' trajectories in real time. Switches between different controllers are triggered based on the robots' states, perception of the environment and explicit communication, as we show in the examples of next sections. In the applications presented in this paper, we use potential field controllers, where the robots are attracted by the goal and repulsed by each other. Other controllers for holonomic and nonholonomic robots have also been implemented in the simulator, such as path following, leader-follower and open loop approaches.

2.4 Communication

The robots can exchange information (messages) using explicit communication. All messages are broadcast and received by all robots. There are two types of communication: synchronous and asynchronous. In synchronous communication, the messages are sent and received continuously in a constant rate while in asynchronous communication, an interruption is generated when a message is received. Synchronous messages are important in situations where the robots must receive constant updates about the state of the others, for example, in a leader-follower cooperative manipulation task (Chaimowicz et al., 2001). On the other hand, asynchronous communication is used when, for example, one robot needs to inform the others about unexpected events or discrete state changes such as the presence of obstacles, robot failures, etc.

2.5 Sensors, Localization and Mapping

Each robot is equipped with a sensor that allows it to detect the presence of obstacles, robots and other targets in a certain range. For now, a strong assumption of the simulator is that each robot knows its exact position in the environment, i.e., we are not considering odometry errors. In real robots, odometry errors can be corrected using stochastic localization algorithms, that can estimate the robot position using features detected in the environment and other robots' estimations. These kind of algorithms are currently being implemented in the simulator. Knowing their position and using information acquired by the sensors, the robots can construct maps of the environment and plan trajectories in some of the tasks.

3 Loosely-Coupled Task: Forage

As an example of a loosely coupled cooperative task simulation, we present here some experiments of a forage task. As mentioned, in the forage task the robots must search the environment for items, retrieve, and transport these items to a goal location. It is a classical example of a loosely coupled cooperative task and has been used by several researchers as a test bed for simulations and real implementations of cooperative architectures, for example (Arkin et al., 1993) and (Drogoul and Ferber, 1992).

In our simulation, the robots use potential field controllers and discrete mode switching to guide themselves during the execution of the task. The control is completely distributed. There are three basic discrete modes: **Wander**, **Retrieve** and **Transport** (Figure 3). In the Wander mode each robot searches the environment for items to be retrieved. When it detects an item, the robot changes its state to Retrieve and moves to get the item. After getting it, the robot switches to the Transport mode and carry the item to the goal.

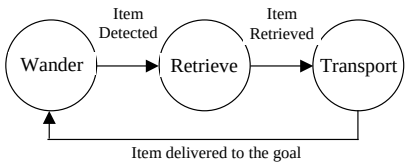

Figure 3. Discrete state diagram for the forage task

Figure 4 shows a snapshot of the simulator during the forage task: the items are the small dots in the screen, while the robots are the circles. The sensor range of each robot is shown as a dashed circle around it. There are 4 robots in the Transport mode carrying items (black), 4 in the Wander mode (light gray) and 2 in the Retrieve mode (dark gray, at the bottom-left of the screen). The goal is marked with an **x**

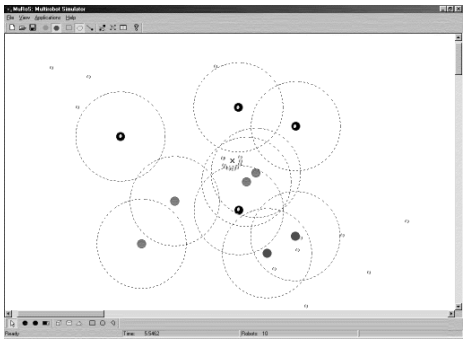

Figure 4. Snapshot of MuRoS during a forage task

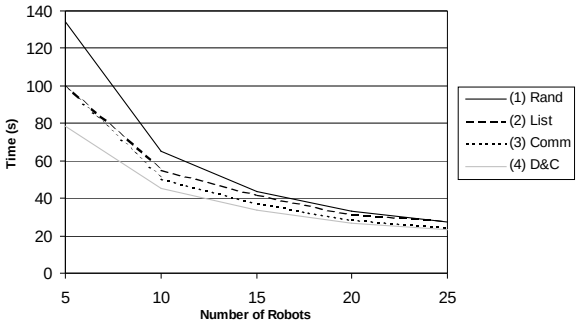

Figure 5. Execution Time × Number of Robots

Four different algorithms have been implemented to coordinate the robots during the forage task:

1. Random: in this algorithm, the robots move randomly around the area and, when they find a item, they retrieve and transport it to the goal. There is no communication or multi-robot strategy.

2. List: this algorithm is very similar to the previous one, with the difference that each robot keeps a list with the position of the items that it has already detected but has not been able to retrieve because it can only transport one item at time.

3. Communication: in this algorithm, the robots also move randomly and maintain a list with the detected items, but they also exchange their lists using explicit communication. In this way, robots that are idle in a certain moment, can go and retrieve items detected by other robots.

4. Divide and Conquer: in the Divide and Conquer algorithm, the robots do not communicate and, instead of moving randomly, they divide the search space among them and perform an exhaustive search in their area. When a robot finishes searching its area, it can start searching other areas.

Experiments were executed varying the number of robots (from 5 to 25) and the coordination algorithm. Each experiment was repeated 100 times, and the average time to complete the mission was computed. In the experiments we used holonomic robots, 50 items and a search area of 10x10 meters with the goal placed in the middle. Each robot cannot carry more than one item each time. Also, as mentioned, we consider that there is no uncertainty in the localization: the robots know their exact position and the position of the goal. The results are shown in Figure 5.

When the number of robots is small, the Divide and Conquer algorithm has the best performance, showing that strategy is more important than keeping a list or communicating at this point. But the other two algorithms also have good performances when compared to the completely random approach. When the number of robots is increased, the difference in the execution time of all the methods decreases, showing that a larger number of robots brings benefits independently of the algorithm that is being used. There are two important results to point out when 25 robots are used: the performance of the Random algorithm is equal to the the List algorithm, showing that with a large number of robots, keeping a list of the previously detected items does not contribute much to the task . The Communication and Divide and Conquer algorithms also have a very similar performance, which demonstrates that the use communication is as important as strategy when a large number of robots is used.

A metric that can be used for this application is the Speedup, a common metric used in the parallel programming community to analyze the benefits of using multiple processors or machines to perform a task. In our application domain, Speedup can be defined as:

$$Speedup = \frac{Execution\ time\ using\ 1\ robot}{Execution\ time\ using\ N\ robots}.$$

Table 1 shows the speedup values computed for the four algorithms varying the number of robots. All the algorithms have large speedups, mainly the random and communication algorithms. The speedup shows how much the execution time is reduced when more than one robot is used. Speedup values close to n ($n = number\ of\ robots$) are called linear speedups and indicate that the task is benefiting totally from the use of multiple robots. This happens with the Communication Algorithm for $n \leq 10$ and the Random for $n \geq 10$. The communication helps orienting the idle robots to the items that have been found by others, bringing significant improvements over the single robot approach. In the Random Algorithm, linear speedups were expected because the robots act completely independent, thus, the addition of more robots causes a proportional reduction in the execution time. The Divide and Conquer is the algorithm that has the smaller speedup results be-

	5	10	15	20	25
Rand	4.72	9.69	14.62	19.16	23.13
List	4.99	9.09	12.03	16.13	17.98
Comm	5.00	9.89	13.38	17.65	20.55
D&C	4.50	7.81	10.41	13.13	15.16

Table 1. Speedup results for different foraging algorithms varying the number of the robots

cause the use of strategy is effective even when only one robot is in use. When more robots are in use, their search areas overlap and this redundancy reduces the speedup values.

All these results showed that loosely coupled tasks benefit significatively from the use of multiple robots. Although the use of communication and coordination brought good results, they are not necessary for task completion. The use of strategy or even random algorithms with a large number of robots are sufficient for the execution of loosely coupled tasks.

4 Tightly-Coupled Task: Manipulation

Cooperative manipulation is a typical example of a tightly coupled task. To transport an object in cooperation, the robots must coordinate themselves in order to pick up the object and carry (or push) it from two different locations. Examples of approaches for cooperative manipulation can be found in (Chaimowicz et al., 2001), (Brown and Jennings, 1995) and (Donald et al., 2000).

In this paper, our approach to the cooperative manipulation uses potential field controllers to guide the robots, and discrete mode switching and communication for coordination. This is similar to the approach used in the loosely coupled cooperation, with the main difference that communication and strict coordination are completely necessary. The robots are attracted by the object and the goal, and repulsed by each other. The contacts with the object, obstacles and other robots are computed using rigid body dynamic models (Song et al., 2001). Figure 6 shows the discrete modes and the transitions during the execution of the task. The transitions marked with a * are triggered by messages received from the other robots.

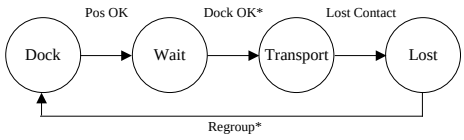

Figure 6. Discrete state diagram for the cooperative manipulation task

Initially, the robots are in the **Dock** mode and are attracted by the object. At the same time, they are repelled by each other, being able to distribute themselves along the object and prepare for the transportation. When one robot senses that it is close enough to the object, it goes to the **Wait** mode, and broadcasts a message communicating that it is ready. When all robots are ready they change to the **Transport** mode, and there is a controller switch so that each robot becomes attracted by the goal. If for some reason one robot losses contact with the object, it will change to the **Lost** mode, broadcast a message and stop moving. If all robots lose contact, they regroup and start docking again.

Figure 7 shows some snapshots of the simulator during the manipulation of a round object by ten holonomic robots in an environment with three obstacles. The goal position is marked with an **x**. Snapshot (a) shows the robots in the Dock mode, starting to move in the direction of the object (light gray circle). Snapshot (b) shows nine robots in the Wait mode while the last one is still finishing the dock phase. In (c) eight of the robots have lost contact with the object because of a collision with an obstacle. It is important to note that the robots lose contact when the object is outside their sensor range, so two robots (showed in black) are still in contact. As these two robots move towards the goal, they will also lose contact with the object. When this happens, they regroup, grab the object again and resume the transport finishing the task (snapshot (d)).

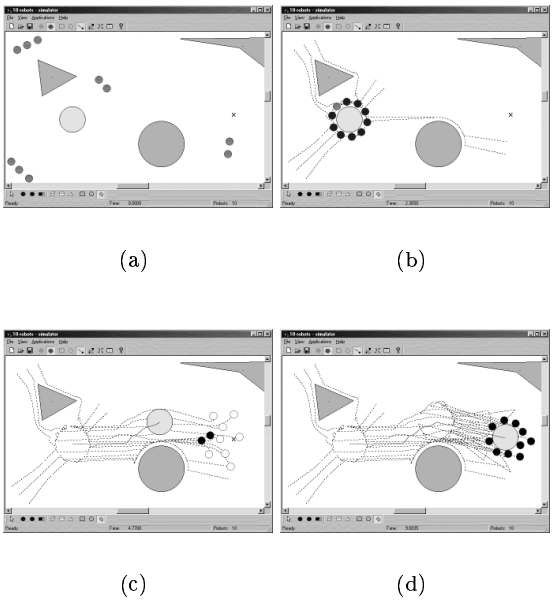

Figure 7. Snapshots of the manipulation task: (a) robots (small circles) in the dock mode. (b) robots preparing to transport the object. (c) robots lose contact with the object after colliding with an obstacles. (d) transportation is finished after a robot regrouping

Differently from loosely coupled tasks, communication and coordination are necessary in this task. Without strict coordination and communi-

cation the robots would not be able to transport the object and recover from failures. Also, depending on the size, shape and mass of the object, a single robot alone or a small team cannot be able to complete the task. Some experiments were performed varying the mass and size of the object and in all cases at least 3 robots were necessary to transport the object. In the case of Figure 7, eight robots were necessary to complete the task adequately. All these results showed that cooperative approaches that rely on strict coordination and explicit or implicit communication mechanisms are required for the execution of tightly coupled tasks.

5 Conclusion [a]

In this work, we presented MuRoS, a object oriented simulator for multi-robot cooperation. We described its basic features and how it can be used for simulating both loosely coupled and tightly coupled tasks. We implemented two test bed applications: forage and cooperative manipulation and studied important aspects of cooperative robotics such as scalability and hybrid control. The results showed that cooperative approaches increase the performance of the tasks and allow the execution of certain tasks that could not be performed by a single robot. We also showed that tightly coupled tasks require strict coordination and communication, while in loosely coupled tasks, robots can run independently from each other.

Our future work is directed toward studying important aspects of cooperative robotics such as wireless communication and dynamic role assignment. For this, we are planning to implement more cooperative applications on the simulator and develop new features, such as a probabilistic localization mechanism. We also want to implement some of the approaches in real platforms to validate the results obtained with the simulator.

References

Alur, R., Grosu, R., Hur, Y., Kumar, V. and Lee, I. (2000). Modular specification of hybrid systems in charon, *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control*.

Arkin, R., Balch, T. and Nitz, E. (1993). Communication of behavioral state in multiagent retrieval tasks, *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*.

Balch, T. and Hybinette, M. (2000). Social potentials for scalable multirobot formations, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pp. 73–80.

Brown, R. G. and Jennings, J. S. (1995). A pusher/steerer model for strongly cooperative mobile robot manipulation, *Proceedings of the 1995 IEEE/RJS International Conference on Intelligent Robots and Systems*, Vol. 3, pp. 562–568.

Cao, Y. U., Fukunaga, A. S. and Kahng, A. B. (1997). Cooperative mobile robotics: Antecedents and directions, *Autonomous Robots* **4**: 1–23.

Chaimowicz, L., Sugar, T., Kumar, V. and Campos, M. (2001). An architecture for tightly coupled multi-robot cooperation, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pp. 2292–2297.

Das, A., Fierro, R., Kumar, V., Southall, B., Spletzer, J. and Taylor, C. (2001). Real-time vision-based control of a nonholonomic mobile robot, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pp. 1714–1719.

Donald, B. R., Gariepy, L. and Rus, D. (2000). Distributed manipulation of multiple objects using ropes, *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pp. 450–456.

Drogoul, A. and Ferber, J. (1992). From tom thumb to the dockers: Some experiments with foraging robots, *Proceedings of the 2nd Int. Conference on Simulation of Adaptive Behavior*, pp. 451–459.

Fierro, R., Das, A., Kumar, V. and Ostrowski, J. (2001). Hybrid control of formations of robots, *Proceedings of the 2001 IEEE Int. Conference on Robotics and Automation*, pp. 3672–3677.

Gerkey, B., Vaughan, R., Stoy, K., Howard, A., Sukhatme, G. and Mataric, M. (2001). Most valuable player: A robot device server for distributed control, *Proceedings of the Second International Workshop on MAS at Autonomous Agents 2001*.

Mathworks (2001). http://www.mathworks.com.

Parker, L. E. (2000). Current state of the art in distributed robot systems, *Distributed Autonomous Robotic Systems 4*, Springer Verlag, pp. 3–12.

Song, P., Kraus, P., Kumar, V. and Dupont, P. (2001). Analysis of rigid body dynamic models for simulation of systems with frictional contacts, *ASME Journal of Applied Mechanics* **68**(1): 118–128.

Sugar, T. and Kumar, V. (1998). Design and control of a compliant parallel manipulator for a mobile platform, *Proceedings of the 1998 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*.

Ye, W., Vaughan, R., Sukhatme, G., Heidemann, J., Estrin, D. and Mataric, M. (2001). Evaluating control strategies for wireless-networked robots using an integrated robot and network simulator, *Proceedings of the 2001 IEEE Int. Conference on Robotics and Automation*, pp. 2941–2947.

[a] Acknowledgments: Luiz Chaimowicz is supported by CAPES Foundation and CNPq and Mario Campos by CNPq.