SIMULATOR FOR COOPERATION AND COMPETITION OF MOBILE ROBOTS

Claudiu Chiculita, Dan Dascalescu, Laurentiu Frangu

University of Galati, Dept. of Automatic Control, Industrial Informatics and Electronics
Claudiu.Chiculita@ugal.ro

Abstract: The paper deals with the structure and properties of a software simulator, intended for mobile robot and autonomous agent experiments. The simulator accepts mechanical, sensory and actuator attributes, specified by the user. It also accepts a user written control module that contains the control law for each animate. Its main application is the simulation of specific environments and of corresponding control laws that allow strategy validation. The simulation applies for physical implemented agents inclusive. The models and the software implementation of the simulator are described.

1. INTRODUCTION

The objective of this paper is to build a simulation tool, allowing the development of experiments of cooperation and competition in a collectivity of mobile robots. The problems raised by the competition between mobile robots concerns the fulfilment of optimum criteria, in an environment in which each team of robots modifies the conditions in which the other teams perform. The notion of team, in this context, accepts different numbers of robots; a variable number of teams may be present in the environment. The goal of each team is to maximize its optimum criterion, before the other teams. The simplest case of robotic competition is that in which there are only two teams, each composed of a single robot. As an example, the classical problem of the prey and the predator, in which the predator hunts after the prey, while the prey's objective is to avoid the capture. Obviously, the two actors have different properties (features), so they will use different strategies to achieve their goals. The automatic control issues involved by this simple competition case consist of:

- individual technique (the lower level of control hierarchy), by which the robot interprets the environment information, performs the positioning operations and the interaction with the other objects or with the competing robot;
- tactical decisions, by which the robot chooses the appropriate response, on narrow time intervals, to the changes in the environment;
- strategic decisions (the higher level of control hierarchy), by which the robot anticipates the range of possible responses to the environment and makes changes that maximizes its optimum criterion;

When a team has more than one member usually it is necessary to develop cooperation between the members. The purpose of the cooperation is to combine the available resources in order to obtain a superior value of the optimum criterion, compared to the value of all individuals taken apart. For example, the football game implies the competition between two teams, but also the cooperation between their members, in order to achieve victory. This is a good example where aspects of cooperation and competition coexist. The solving of the cooperation problem can be done either centralized or distributed to the team members; in the latter situation the robots receive their objective and information about their team-mates' resources. During the competition, they negotiate the possible solutions. Thus, the robots gain an autonomous character.
Another distinct class of experiments is that in which there are many robots, each completely autonomous, resulting in a statistical character of the global behaviour.

In order to validate the general solutions of the cooperation and competition problems, and also to study the behavior of large communities (for which there are not enough physical resources), it is useful to help the analysis by simulation. As a general case, the simulators are implemented by the manufacturers of robots, each for his robot. Some of them present the advantage that the controlling software (validated on the simulator), can be transferred on the physical robots. The downside is that they do not simulate other robots, with other dimensions, another set of sensors or actuators, etc.

2. THE PRESENTED SIMULATOR

The simulator has the goal to implement the relations between the concerned objects (mobile robots and passive objects). That implies the mechanical interactions and the sensory information capture. The commands addressed to the robots, including those based on tactical and strategic decisions, are issued by a separate module (control module), written by

the user and accepted by the simulator. This one is designed to have an open structure, in the sense that the user is allowed to add own modules, regarding sensors, actuators and properties of the robots. However, these modules have to be compiled along with the rest of the simulator, opposed to the control module, which may be external.
An important feature of the simulator is the existence of an unlimited number of sensors and actuators, of different types. The sensors are very important, provided that the simulator is intended to be used in mobile robot and autonomous agent experiments. In these experiments, the mechanical and sensory lack of precision is compensated by the rich information (as in the case of all animals). The sensory information, used for the improvement of the robot behaviour, depends on the positions and is part of the loop, so it is compulsory for it to be simulated online, simultaneously with the mechanical movement.

The simulator is used in a more complex context: it is the starting point for building particular simulation environments, for specific problems. For instance, the validation of the command laws, in a mobile robot community, proceeds as it follows:

- the properties (shape, dimensions, sensors, actuators) of the robots and passive objects are defined;
- the control module, containing the global goal, the behaviour of each robot, the sensory signal processing, the command laws, the strategic decisions, etc., is written by the user and added to the simulator;
- the initial positions and the stop criterion are set;
- the simulation is started and the relevant information is recorded.

The researcher repeats the last two steps, with different initial positions and evaluates the properties of system endowed with that control module. If necessary, the control module is re-written and the last two steps are repeated.
The simulator is different with respect to the usual game simulators, in the sense that the control module may include the computing of the global goal criterion, used to evaluate the effectiveness of the control law, instead of a visual evaluation, performed by the user.

There are two main classes of simulations that can be built with this software:
1. Simulation of mobile robots, in order to develop strategies that will actually be implemented on a physical robot. In this case all the properties of the simulated robot should be set the same as the real ones (dimensions, weight, friction, etc.). The sensors and the actuators should mimic those mounted on the real robot. The sensors should return low level information that will be interpreted by the control algorithm of each robot and transformed into usable information; the actuators should receive and execute the low-level commands, necessary to completely define the response to the environment (for instance, the speed or the torque of the motors, driving the wheels).

2. Simulation of agents that do not have a correspondent physical robot or simulation of communities that are difficult to implement in the laboratory. Examples of such simulations are the agents that behave like human beings (football players) or simulation of large crowds (of people, of ants or other). In these cases the sensors may return preprocessed information and the actuators may work with higher level information, which relieves the researcher (who writes the agent brain) from having to handle robot problems and allow him to focus on higher level concepts; for example, a sensor could return all visible obstacles along with their properties, an actuator could have as input values only the speed and orientation angle etc..

3. SOFTWARE IMPLEMENTATION

The presented simulator engine is intended to provide the building blocks to easily construct a wide range of custom simulators. Its applicability is mainly in the field of mobile robots and the study of intelligent agents. The simulator has an open structure, in the sense that user-written modules can be added, regarding the control algorithms, sensors and actuators. The language chosen for implementation was Delphi.
All objects are considered to have right edges (spheres are an exception to this rule), and to have approximately the same height. The simulation is performed in 2D, so only the planar projection of the objects it is considered. In simulation spheres and cylindrical objects look the same (because have the same projection) but they act differently because of their properties (moment of inertia, friction). The main items that can be used to build simulations are: bodies, robots, sensors and actuators, propagation mediums and markers.

Bodies are passive solid objects that have a circular or polygonal shape. The properties that are taken into account when simulating their movement are: shape, mass, moment of inertia, friction coefficients, elasticity, resultant force, linear and angular velocities. The *robots* inherit all properties from bodies, add some new properties (as health and energy) and support for sensors, actuators and a local or remote controller which will act as a brain.

The *sensors and the actuators* have similar implementation, communicating with the robot controller by means of messages. The messages are received and maintained by each robot in a queue and are fed to the sensors and actuators at each time step.
The sensors and the actuators have the following common properties:

- Have an offset and orientation angle relative to the robot they are mounted on;

- They influence the energy drain of the robot by a specified amount (per second or per activation);
- Add weight to the robot;
- Can be limited to a maximum number of activations per second in order not to be abused and consume the cpu resources;
- Their precision is altered by an error coefficient, which can change during execution;

The *sensors* activate in one of the situations: if they are set to auto-activate at fixed time intervals, if the controller requested them to, or if they receive information from a propagation medium they are linked to. At activation they analyse the environment and return the results by means of messages.

The *actuators* activate on request and can change the properties of the robot they are mounted on (like speed, direction) or of other objects on the environment (kick, pick-up, etc). For example the driving wheels actuator (for a robot with two wheels, differentially steered) is commanded only by two parameters representing the speed of each wheel. Based on the power and characteristic of the motors, and the other properties of the robot, the two speed parameters are transformed in linear and angular velocity and are applied to the robot.

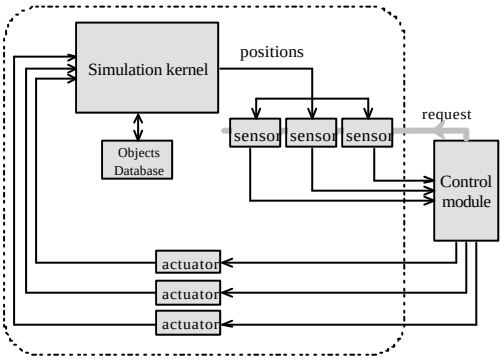

Figure 1.

Propagation mediums represent ways of exchanging information in the simulated environment. Examples of such mediums are: radio, sound, wind, smell, bright light, etc. In order to receive information from a specific medium, a sensor must register himself to that medium. When an actuator emits information, the medium decides (based on emitter and receiver position, propagation properties and environment configuration) which of the receivers (sensors) will receive that information, and how attenuated this will be.

Markers are areas on the ‚ground' which can be used as guides for robots or which can be tested to see if a certain object is overlapping them.

The *controller* of the robot it is designed to be a separate program and to communicate with the simulator in a client-server manner (the simulator plays the server role while the controller acts as client). The client asks information from the sensors, or receives it from the server periodically, then computes and sends the output to the actuators.

The advantages of this architecture are:

- there are no restrictions on how clients are built (they can be written in any programming language) the only requirement being that the language used for development to support communication through TCP or UDP protocols.
- The clients can run on different machines, performing heavy computations without interfering with the simulation engine or with the other clients;

This structure was mainly choose to support competitions, where the members of each robot team run on different computers than the simulator. Besides the independence of resources this prevents cheating tricks that could be directed towards the simulator or to the other team's controller. In order to prevent abuse of the simulator by clients, other different mechanisms were implemented.

The disadvantage of this approach is that each client must implement a separate layer (in the programming language he chose to write his controller) which deals with the communication with the server and the extraction and interpreting the received data.

For the very simple robots or when simulating a large collectivity of reactive agents (that do not perform a lot of thinking) the brain can be placed inside the simulator, his execution being in this case synchronous with the simulator. Figure 2 present different ways of placing the robot controller.

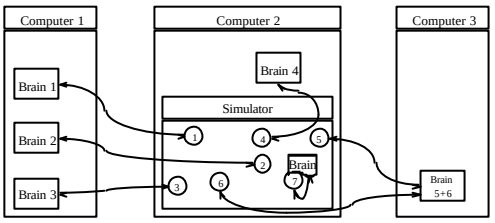

Figure 2. Different ways of placing the robot brain.

The current *physics* model support dynamic collisions between objects (having either circular or polygonal shape) having linear and angular speed. It also has a limited support for static interaction forces between objects.

The motion of bodies is simulated in two dimensions with these assumptions: bodies are rigid and do not deform by contact or collision; the shape of an object is polygonal or circular; collision is modelled by the coefficient of restitution. The method used was that of impulse-based dynamic simulation.

As long as no collisions occur the objects evolve on their ballistic trajectories and the dynamic equations of motion can be and calculated independently of each other; if contact or collision occurs constraint impulses that impose correct dynamics behaviour and prohibit inter-penetrations of objects are applied. If objects remain in contact after the collision is resolved, contact forces are calculated to model the contacts.

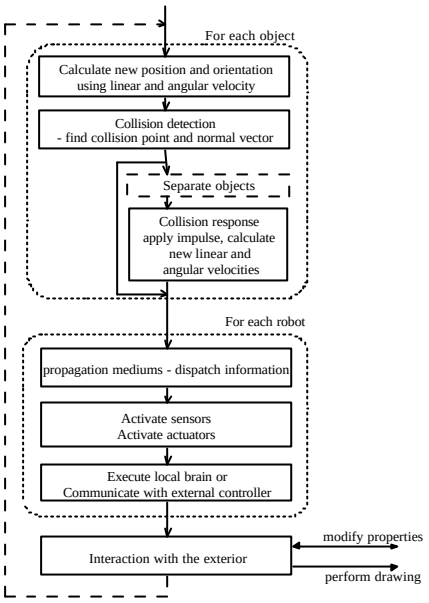

Figure 3. Main simulation loop.

Time management. The server is working with discrete time intervals. At each time interval the dynamics of the objects are computed and the messages from and to the sensors and actuators are processed (Figure 3).
The Simulation steps can be chosen constants (an interval of 10 ms being enough to perform all calculations) or the simulation can run continuous (the cycles succeeds one after another with no delay, the time used for integration in one cycle being the time spent on the previous cycle).

Graphics. Because the task of displaying the animation of the robots is much more expensive than the simulation (two or three orders of magnitude), the drawing is not performed at every simulation step. The graphic routines are also designed to have an open structure and to be easily extended to any graphic library. Drawing routines using the GDI, GDIPlus and G32 libraries were implemented.

4. SETTING UP AN EXPERIMENT

As an example of how simulations are built an experiment where a number of robots have the task of clearing an area where there are scattered objects will be set up. The robots' objective is to move all these scattered objects in a certain area. They do not have any device to pick up objects so their only choice is to push them to the desired location. For the physical experiment two small robots with two differential wheels will be used which will receive command from a computer by radio. Reaction will be provided by a video camera placed above the scenery. Before setting up the experiment it is desired to first study the behaviour of the control algorithms in simulation.

First the properties of the simulated robots have to be set up. For the pusher robots, their dimensions, mass, friction coefficients will be set to the values close to the real robots. The following sensors and actuators will be added to each of the robots:

- Actuator for propulsion with two motors, each connected to a wheel.
- global video camera which returns the approximate position of all objects in the scene
- radio receiver to communicate with the other robots
- radio emitter

The video camera will be set up to consume no power (because is not mounted on the robot), the propulsion and radio emitter will consume energy at each activation, while the radio receiver will drain energy continuously. The control algorithm of the robots will follow these steps:

1. find the closest object that is outside the destination area;
2. move around to the selected object's back;
3. push the object towards the destination area, steering it to maintain the correct course;
4. if the object slides outside the steering zone go to step 2;
5. if the object is inside the destination area, go to step 1;

In order to make things harder for the pusher robots, a perturbing robot is introduced; this will be equipped only with propulsion and a proximity sensor. His control algorithm will be slowly wander around, and when he detects an obstacle in front of him, will strongly bump into it.
Because the robots control algorithms are very simple, they will be added to the main program.
The environment will be populated with the following entities:

- a radio channel that enables communication between the two robots;
- two pusher robots;
- one disturbing robot;
- one marker representing the destination area;
- a number of randomly placed objects;

5. CONCLUSIONS

The important feature of the simulator are tha includes sensory information, that evolves simultaneously with the position and other mechanical data and the possibility to build various derived simulators, tailored to the specific properties of different environments.

REFERENCES

Brian Mirtich, John Canny, Impulse-based simulation of rigid bodies, Proceedings of Symposium on Interactive 3D Graphics, 1995