USING A REAL-TIME KERNEL TO SIMULATE THE MICRO-RATO ROBOTICS CONTEST

João Capucho[1], Luís Almeida[1], Giorgio Buttazzo[2]
etjcnc@ieeta.pt, lda@det.ua.pt, buttazzo@unipv.it

[1]*DET - IEETA*
Universidade de Aveiro
P-3810-193 Aveiro, Portugal

[2]*DIS - Università di Pavia*
Via Ferrata 1
I-27100 Pavia, Italia

Abstract: After 5 editions of growing number of participants coming from all over Portugal, the Micro-Rato contest has become a well known mobile robotics contest in the country. It promotes a cross fertilization among teams that is strongly educational, appealing and challenging simultaneously for those more or less skilled.
In order to improve the robots performance, the use of a contest simulator during construction is highly desirable. In fact, it allows testing the robot behaviour in parallel with its construction, possibly leading to the early detection of either software or hardware design faults. This paper describes such a simulator that is based on a multi-tasking real-time kernel (HARTIK). The advantages of using such a software infrastructure are also discussed. In particular it allows enforcing the correct temporal behaviour of each simulated robot, leading to a more realistic simulation.

Keywords: Robotics, Simulators, Real-time systems, Robot programming.

1. INTRODUCTION

Started in 1995, the Micro-Rato Contest of the University of Aveiro (Almeida *et al*., 2000) has become a national meeting point for robotics enthusiasts (Fig. 1). Five editions have taken place with growing number of participants coming from all over the country. A particular aspect of the contest is that it uses a set of rules that allow for the participation of teams with very different levels of skills. The result is a cross fertilization among teams that is highly educational, appealing and challenging simultaneously for those more or less skilled.

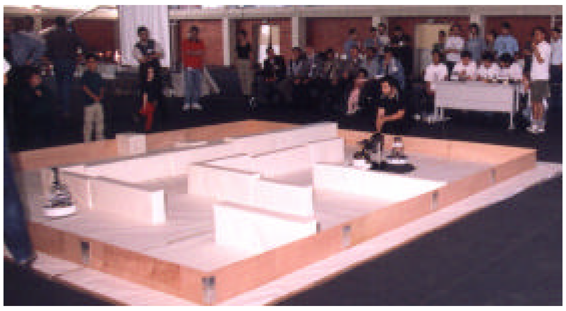
Fig. 1 - A view of the contest in the 2000 edition.

From a competitive point of view, it is obviously important to improve the robots performance. This involves extensive testing, which, typically, can be carried out after the robot is built, only. This means that undesired behavioural characteristics of the robot that originate from the respective programming and become evident after running only, are, sometimes, discovered too late to be fixed. Thus, the possibility of running the program even before the robot is completely built is highly desirable since it allows to test the robot control software in parallel with the robot construction and thus to early detect either software and/or hardware design faults. The availability of an adequate simulator can help achieving such goal.

One of the problems that are inherent to simulation is the accuracy of the models used. If models are not accurate, the output of the simulator is not realistic and its usefulness as a test bed for the robot software is lost. The problem is further exacerbated when the simulator is expected to handle several independent entities simultaneously, such as several robots, each with its own dynamics. In this case, the use of a modular approach supported on a multi-tasking infrastructure is particularly well suited.

This paper describes a simulator for the Micro-Rato Contest that is based on a multi-tasking real-time kernel called HARTIK (Buttazzo, 1993). Section 2 presents a discussion on the advantages of using a real-time kernel while section 3 introduces the contest rules. Section 4 is devoted to the explanation of the models used for the robots sensors, dynamics and kinematics. Section 5 presents the structure of the simulator and section 6 concludes the paper.

2. WHY A REAL-TIME KERNEL?

The simulation of a simple reactive programmable mobile robot is carried out by means of a cyclic execution of the robot software, fed with data generated by adequate models of the robot sensors, kinematics and dynamics. Basically, the simulator uses the virtual speed of the robot wheels to calculate its position after a given time interval and then calculate the corresponding sensor readings. These readings are then supplied to a copy of the control software that generates new motor speeds and so forth (Fig. 2). Each cycle in the simulation corresponds to one cycle of the control software. This correspondence establishes the relationship between simulation time and real time.

When the simulation is expected to look realistic, while displayed at run-time on a screen, both times should be equivalent, i.e. both cycles should execute at the same rate. Thus the simulator cycle should be triggered periodically, according to the rate at which the robot control loop is or will be executed. However, when the simulation results are displayed at the end, only, the correspondence between simulation time and real time is not required. In this case, the simulator can spin on its cycle in order to generate the output as soon as possible. In other situations, a fixed relationship between simulation time and real

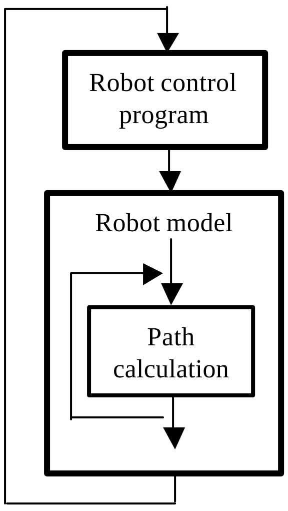

Fig. 2 – Single mobile robot simulator

time can be imposed in order to accelerate or slow down the simulator with respect to reality.

Apart from the previous considerations, there is also an inherent problem of accuracy with the robot kinematics related to the path calculation. Given an initial position and a pair of functions for the linear and angular velocities of the robot (considering differential drive), the robot position after a given interval of time is obtained by integrating simultaneously the referred functions. In order to reduce the error of such simultaneous integration, it is usually carried out in shorter steps within the main simulation cycle.

Hence, the simulation of a simple mobile programmable robot normally requires two chained cycles, an outer one corresponding to the robot control and a faster inner one corresponding to the path calculation embedded in the robot model (Fig. 2). The number of inner cycle executions per outer cycle execution depends on the average distance travelled by the robot in each outer cycle and on the level of accuracy desired for the integration. An example of a simulator that uses this approach is given in (Almeida, 1997).

When simulating just one robot, the software structure required to execute the referred chained cycles at the appropriate rates is very simple. However, it becomes more complex when the number of robots to be simultaneously simulated increases. In this situation, each robot may have a different control cycle period and thus the merging of all cycles into just one main cycle (i.e. the main simulator cycle) may be difficult depending on the actual periods.

The solution adopted in the ROBOCUP simulation league (ROBOCUP, 2000), and recently in the Ciber-Rato contest (CR, 2000), easily supports multiple robots. Each robot control program is implemented as a different process (clients) under the Linux operating system. Another process (server), the *world simulator*, joins the models of all the robots and periodically updates the *status of the world*, i.e. the robots current positions and statuses (Fig. 3). The clients, i.e. the robots, cyclically request new sensor readings from the server and, based on those, generate new commands, which, in turn, are used by the server to update the status of the world. If a robot generates more than one command within one server cycle, just the last command will be used in the next cycle and if it does not generate any command, the one of the previous cycle will be used. This means that the robots are simulated with periods that are integer multiples of the basic server cycle.

This approach to multiple robot simulation has the advantage of being secure since the *status of the world* is local to the world simulator and thus, it is protected from the other processes, i.e. the competitors. On the other hand, there is no protection

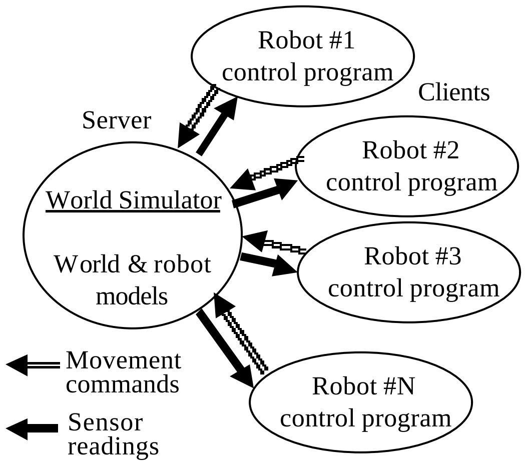

Fig. 3 – A possible approach to simulate N robots.

against temporal interference among robots. In fact, a robot that decides to saturate the world simulator with new commands at a high rate may well prevent other robots to pass their latest commands through.

The problem of mutual interference can be fixed if a real-time kernel is used instead of a general-purpose operating system. In this case, such interference is controlled and bounded by making use of adequate scheduling techniques. Furthermore, the periodic activation of tasks is also simplified, making it easier to manage tasks with different activation periods.

In the next sections, a simulator is described based on a multi-tasking real-time kernel. Basically, each cyclic activity, either robot control program or robot model, is encapsulated into a separate task. These tasks are activated periodically by the kernel, at the required rates, transparently to the programmer. This separation of activities in tasks is very intuitive and simplifies both the programming and the program maintenance. The result is a graphical simulator with a very realistic behaviour and a high level of flexibility. On the other hand, the status of the robots and their positions are global. Thus, this approach is not secure but this is not a negative aspect since this simulator is not used to support a software competition but just the teams that build robots for the Micro-Rato Contest.

3. MICRO-RATO CONTEST: THE RULES

The Micro-Rato Contest has been organised in the University of Aveiro, Portugal, since 1995 with the purpose of helping students of electronics and computer engineering to develop their technical skills in an informal way. In order to achieve this goal, a set of rules was made up that allows the participation of teams of students with very different levels of skills. On one hand, the technical problem has to allow simple solutions, possibly inefficient, so that the less skilled students can deal with it. Simultaneously, the problem must allow more complex solutions, with better performance, so that the more skilled students feel challenged.

The choice fell on a maze-type contest where several small-sized robots have to achieve a given goal in the shortest time.

The goal is to find an infrared beacon that is placed at the centre of a black circle (the finishing area), somewhere in the maze, without colliding with any of the obstacles that are placed along the way. The robots run three at a time and they finish their run after completely entering the black circle and stopping there. Each run can take at most 3 minutes.

The dimensions of the robots cannot exceed 30x30x40cm and the maze is 5 by 5 meters wide. The obstacles are boxes placed inside the maze and they reflect infrared light as well as the maze walls.

4. MODELLING THE ROBOTS

4.1. Previous considerations

In order to simplify the modelling of the robots used by the simulator, a few assumptions have been made. First, all the robots are round; second they have two independent motors for traction and drive; and third they have a set of sensors to detect the obstacles, the beacon and the finishing area. Furthermore, the obstacle sensors are infrared reflection based.

4.2. Operational parameters

In order to model the robots in the maze the operational parameters associated to each object in the system that is to be simulated must be clearly identified. These objects are the obstacles, beacon and goal area that form the maze, and the robots with their sensors and electric motors.

In what concerns the modelling of an obstacle sensor two parameters are needed: the intensity of the beam emitted and the angle of the receiver visual field (Fig. 4a). The output is the value read by the sensor and it is an analogue value.

The beacon sensor is also infrared based and it also returns an analogue output value similar to the obstacle sensor. It has only one parameter that is the angle of the receiver visual field (Fig. 4b). In this case, the infrared light is emitted by the beacon.

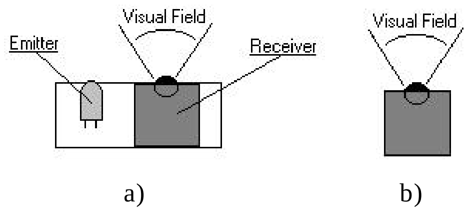

Fig. 4 - Obstacle (a) and beacon (b) sensors.

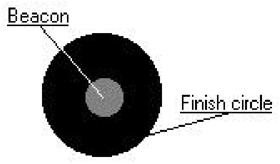

Fig. 5 – Beacon and finishing area

The beacon is static in the maze and it is in the centre of a black circle that represents the finishing area (Fig. 5). To model the beacon we need the position where it is placed in the maze, its intensity and the radius of the finishing area.

The last of the sensors is the one especially dedicated to detecting the finishing area. It is also infrared based and only reacts to the black ground of that area. In this case, there is no parameter and the output is digital.

The obstacle sensors are placed on the periphery of the robot while the beacon sensor is placed in the centre, on top of the robot, and the finishing area sensor is placed also in the centre but beneath the robot. The orientation of the sensors is defined using polar coordinates relative to the robot orientation.

In what concerns the motors, it is assumed that they are mounted on the robot according to Fig. 6. For each one the distance to the robot centre must be specified.

Finally, the robot itself is modelled by its position in the maze, direction, external radius, a set of sensors and a pair of motors.

4.3. Sensor dynamics

The obstacle sensors respond according to the distance from the robot to the obstacle, it means the closer they are, the higher the output value is. In sake of simplicity, it was considered that the output value varies linearly between a maximum value, obstacle near the sensor, and a minimal value, no reflection received which means that there is no obstacle within a certain range.

Likewise, the beacon sensor responds linearly with the distance, between a maximum value, near the beacon, and a minimum one, outside a given range.

The finishing area sensor returns '0'outside of that area and '1' inside it.

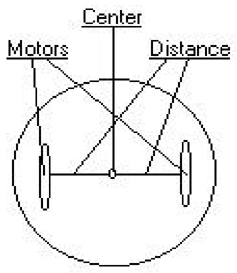

Fig. 6 – Motors layout

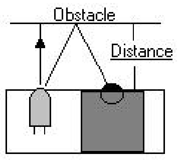

Fig. 7 – Obstacle detection.

All the sensors also obey to a first order model with a time-constant of 10 ms.

4.4. Robot kinematics and dynamics

The robot kinematics is determined by the rotating speeds of the motors. The position and orientation of the robot in the maze is obtained by integration of its linear and angular velocities.

The movement of the robot can be decomposed into two components, displacement and rotation. The equations (1) allow determining the linear velocity (v), that causes the displacement, as well as the angular velocity (w), that causes the rotation. These equations take into account the distance between the motors (b) as well as the rotating speeds of both motors, v_0 and v_1 for the left and right one, respectively.

$$v=(v_0-v_1)/2 \qquad w=(v_0-v_1)/b \qquad (1)$$

In what concerns the dynamics of the robot a simplified linear motor model is considered where the rotating speed is proportional to the control signal, i.e. the voltage applied to the DC motor. To take into account the robot inertia, we used a first order model with a time constant of approximately 100ms. This value is coherent with the external behaviour of our robot but varies according to several aspects, like the weight of the robot, its moment of inertia, the torque of the motors.

When a signal (u_i) is applied to motor i, a torque is generated and then converted into velocity (v_i) by the respective wheel. The proportional relationship is expressed below.

$$v_i=K.u_i \qquad (2)$$

To take into account the first order model, the result of (2) is passed through an equivalent discrete low-pass filter which parameters are the time-constant of 100ms ($\boldsymbol{t}$) and the sampling period of 2ms (h). The filter is of the IIR type (i.e. recursive) that allows a very compact implementation. Its transfer function in the Z domain is given by (3).

$$V_i'(Z) / V_i(Z) = (b_1 + b_2.Z^{-1}) / (a_1 + a_2.Z^{-1}) \qquad (3)$$

This transfer function can easily be converted to the time domain as in (4) where k is the sample index.

$$v_i'(k) = 1/a_1*(b_1.v_i(k) + b_2.v_i(k-1) - a_2.v_i'(k-1)) \qquad (4)$$

By using Matlab, the parameters a_i and b_i can be readily obtained for the given values of $\boldsymbol{t}$ and h. The final expression is given in (5).

$$v_i'(k) = 0.1813\, v_i(k-1) + 0.8187\, v_i'(k-1) \qquad (5)$$

The values $v_i'(k)$ are those used in expressions (1) to update the robot's angular and linear velocities (w and v respectively). The entry values are the control signals u_i in (2) which are determined by the control algorithm implemented by the robot's programmer.

4.5. The guiding algorithm

The main purpose of this simulator is to allow the testing of different guiding algorithms before the robot actual construction. Just to test the simulator, a basic algorithm was used which already allows to solve simple mazes although it may fail in face of more complex obstacle patterns. Basically, the robot follows the walls of the obstacles, or of the maze, and when the beacon is detected in the front of the robot, it goes straight to it.

5. THE SIMULATOR IMPLEMENTATION

The simulator has been developed under the framework of the HARTIK kernel (Buttazzo, 1993), using the C language and the GNU compiler. The HARTIK kernel schedules real-time tasks according to the Earliest Deadline First (EDF) criterion which allows for a high processor utilisation with guaranteed timely behaviour. The kernel also allows for the coexistence of hard real-time tasks, i.e. those which schedulability is guaranteed (given certain assumptions of worst-case execution time), soft real-time tasks, i.e. those for which there are no guarantees but are scheduled according to a best-effort approach with respect to their timing requirements, and non-real-time tasks, which execution is carried out in priority order and if the processor has no hard or soft real-time tasks to execute, only. The tasks can also be periodic, i.e. activated automatically by the kernel, or sporadic, i.e. activated asynchronously by external events. In the case of this simulator, all tasks are periodic. Furthermore, those with tighter time constraints have deadlines equal to their periods.

5.1. Defining the tasks

Taking into account that the robots are microprocessor based, the respective software and hardware parts have been separated. These parts, for each robot, are simulated by two independent periodic tasks. The first one, named *Mousetask*, is a hard real time task that implements the control algorithm. It reads the values of the sensors, does the necessary computations according to the robot-guiding algorithm and acts over the motors, i.e. it generates motor voltages. The second one, named *Calculate-_Position*, is also a hard real time task that handles the physical motion of the robot. It reads the motors voltages generated by Mousetask, and updates the robot position and direction according to the model explained in section 4.4.

The graphical environment is maintained by two other tasks. The *Redraw_Mouse_Task*, which is a soft real time task that takes care of all the required drawing and redrawing of each robot, and the *Redraw_Maze_Task*, which is a non real time task that handles the redrawing of the maze.

The last task, named *Referee*, monitors the paths of the robots in the maze, like the referee does in the real contest. It is a hard real time task that counts the number of collisions of each robot against the obstacles or other robots. The tasking structure of the simulator is represented in Fig. 8.

5.2. Time constraints

As explained in the previous section, each robot is simulated by two tasks: Mousetask and Calculate-_position. The latter takes into account the physical aspects of the robot to update its position. This calculation is achieved by simultaneously integrating the linear and angular velocities of the robot. This integration contains an intrinsic error that is smaller when the integration steps are shorter. Therefore, the faster the task runs the smaller will be the errors of path calculation.

Furthermore, for every new position of the robots the Referee must verify whether new collisions occurred. This implies that the Referee task must run faster than the Calculate_position task, which updates the position of the robot.

With a system tick of 1ms, the Referee and

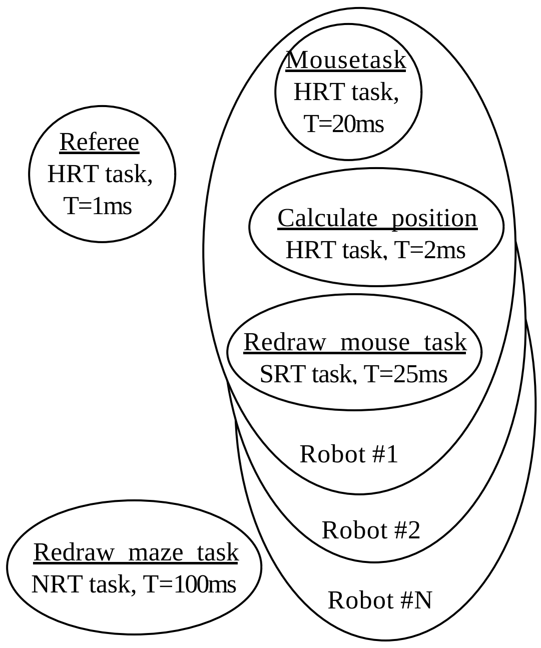

Fig. 8 – The simulator tasking structure.

Calculate_Position tasks have been given periods of 1 and 2ms respectively.

The Mousetask is not so constrained in terms of period specification. In fact, its period represents the rate of execution of the robot control algorithm. In this case, we have considered a value so that, for a maximum speed of about 50cm per second and each pixel representing 1cm, the control algorithm would run at least once for each position increment of one pixel. Therefore, its period was set to 1/50s or 20ms.

Both Redraw_Mouse_Task and Redraw_Maze_Task just handle the graphical representation of the competition and thus have no hard time constraints. Notice that, even if they miss a few activations, the correct position of the robots is still properly updated by the hard real-time tasks. Thus, the impact of such miss is minimal and confined to the display, causing no error in the underlying simulation. Nevertheless, to obtain a smooth displacement of the robot, the period of Redraw_Mouse_Task was set to 25ms. In what concerns the Redraw_Maze_Task, its action is seen only when the robots collide with the obstacles. Its period was set to 100ms.

5.3. User interface

According to its main function, the simulator output is a 2-D dynamic graphical representation of the contest where the virtual robots evolve in a similar fashion to reality. Fig. 9 shows an example of the simulator graphical output. Apart from the graphical representation, the simulator also informs the user, in a textual form, about the current number of collisions of each robot as detected by the Referee. Notice that, in the competition, these collisions are considered undesired and cause extra penalty time to be added to the actual time taken by the robot in its run.

The input to the simulator is carried out by means of a text file which contains either the maze description, i.e. position of the beacon and number, position, size and type of the obstacles, as well as number of robots to run simultaneously and their starting positions.

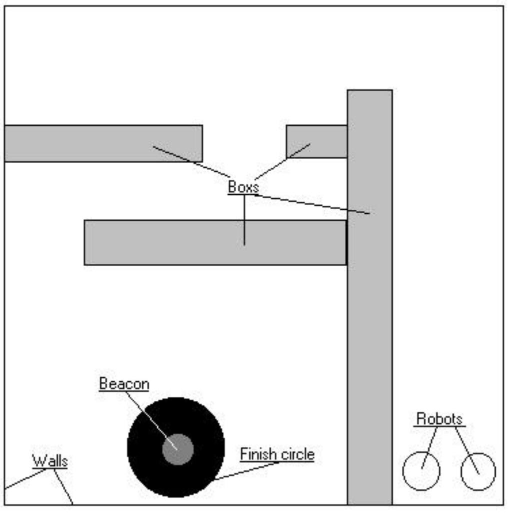

Fig. 9 – Simulator graphical output

6. CONCLUSION

The purpose of this work was to develop a tool that allowed the simulation of the Micro-Rato contest in a simple but efficient way. With such a tool it is possible to easily test different robot control algorithms that can be used to enhance their performance in the real contest.

This paper describes such a simulator built on top of a real-time kernel. The advantages of using such software infrastructure are also discussed, namely the simplicity of managing multiple cyclic activities with different periods as well as the enforcement of the specified temporal behaviour of each activity. These aspects contribute to a more robust and realistic simulation.

In this first version of the simulator, a simplified first order model was used for the behaviour of both robots and sensors. Although somehow simplistic, this approach already yields reasonably realistic results. The number of sensors available in each robot is parameterised and can be easily changed. In the current implementation, the robots have three obstacle sensors, two beacon sensors, one finishing area sensor and two motors, placed like described in section 4.2.

Two tasks were implemented to simulate each robot, one for all the physical aspects, i.e. the robot model, and another for the control algorithm. A third task per robot was also used to handle its graphical representation in the maze.

Moreover, two other tasks were used, one to monitor all the movements of the robots, counting also their collisions, and another for redrawing the maze upon collisions.

The simulator was tested with several different robot control algorithms, in several different mazes, and it worked as expected with a realistic look. Future work will be carried out in order to establish and improve the accuracy of the robot and sensor models used in the simulator.

7. REFERENCES

Almeida, L., P. Fonseca and J.L. Azevedo. (2000). The Micro-Rato Contest: a popular approach to improve self-study in electronics and computer science. *Proc of SMC'2000 (IEEE Conf. on Systems, Man and Cybernetics)*, Nashville, USA, October 2000.

Buttazzo, G. (1993). HARTIK: A real-time kernel for robotics applications. *Proc of RTSS'93 (IEEE Real-Time Systems Symposium)*, Raleigh-Durham, USA, December 1993.

Almeida, L. (1997). Modelização de pequenos robots autónomos: um exemplo. *Revista do DETUA*, Vol. 2, Nr. 1, pp. 133-140, Setembro de 1997.

ROBOCUP (2000). The ROBOCUP simulation league. http://www.robocup.org

CR (2000). Regras da modalidade Ciber-Rato - Concurso Micro-Rato 2000. *Arquivo técnico Ciber-Rato*.http://microrato.ua.pt/docs/ documentos.html