Human-Computer Cooperation Platform for Developing Real-Time Robotic Applications

Carlos Domínguez, Juan-Miguel Martínez, Jose V. Busquets-Mataix, Houcine Hassan
Department of Computer Engineering (DISCA)
Universidad Politécnica de Valencia (UPV) - Spain
Tel:(+34) 96-3877578, Fax:(+34) 96-3877579,
E-mail:{carlosd, husein, jmmr, vbusque}@disca.upv.es

Abstract:

This paper presents a Human-Computer Cooperation (HCC) platform, which permits the coordination between the user and the tool to improve the development of real-time control applications (e.g. mobile robots). These applications have functional (robot objectives) and temporal requirements to accomplish (deadlines guarantee of tasks). The simulation tool has been designed in order to permit the testing and validation of these two requirements. To this end, the tool is composed of two independent simulators interconnected through a shared memory: the robot simulator (functional level) and the real-time task scheduler simulator (task execution level). Robotic applications can be defined with the robot simulator while the real-time scheduler simulator permits to analyze the schedulability of the robotic tasks. The real-time task simulator incorporates a flexible task model where the task temporal parameters (e.g. computation time) adapt to the requirements of the application (e.g. number of objects in scenes), thus, the use of the CPU is not overestimated. A key issue of the framework is the human-computer interface, which allows the monitoring of different parameters of the application: robot objectives, task schedule, robot speed, computation time, CPU utilization, deadline misses. The usefulness of the simulation tool is shown through different robotic navigation experiments. Finally, the simulation tool has been used to evaluate the proposed flexible model of tasks compared to a traditional fixed temporal parameters task model. Results show that the robot fulfills the objectives earlier, about 32% on average, and consumes on average about 15% less CPU to accomplish the objectives.

Keywords: human-computer cooperation, real-time systems, mobile robots, scheduling, simulation tool, graphical user interface, object oriented programming, schedulability analisys

I. Introduction

Complex real-time control applications (e.g. robotics, avionics) are composed mainly of hard and soft real-time tasks [1]. Missing a deadline in hard real-time tasks may result in severe malfunction of the system, however in soft real-time tasks, deadline misses may be tolerated with a graceful degradation of system performance. This paper is concerned with hard real-time tasks involved in the design of real-time control applications. These tasks are composed of three main parameters: deadline, period and computation time. Control applications (i.e. mobile robots) perform their activities in dynamic and unpredictable environments. The number of objects contained in the scenes should have influence in the variation of the computation time of the tasks, that is, the computation time should vary proportionally to the number of objects detected in the scenes. Moreover, the speed should inversely influence the deadlines and periods of the tasks: increasing the speed implies a decrease of the periods/deadlines and vice-versa. The malfunction of mobile robots in a factory where coexist with personnel, may inflict human accidents and hardware damage, thus this application requires hard real-time categorization.

Regarding the robotic application, task models with fixed temporal parameters [2] are not appropriate because the design of the temporal parameters are overestimated (i.e. the computation time is the worst case execution time that corresponds to an environment with a plethora of objects), this situation can lead to a waste of the resources of the system (i.e. CPU). However, if the parameters are designed according to the requirements of the application (i.e. computation depends on the number of objects) then the resources are used conveniently [3]. In this paper presents a human-computer framework where the temporal parameters adapt to the requirements of the application. Using functions, the programmer can define the relation between the parameters of the application and the temporal parameters of the tasks. These functions can be programmed in the development tool that has been designed. The tool permits to define on the one hand the robotic application (e.g. objectives, obstacles, etc.) and on the other hand, the real-time tasks that support the execution of the application (i.e. obstacle avoidance).

Even though simulation tools of control applications (i.e. robots) are fully available [4], [5], they do not integrate the scheduling level that allows to prove the guarantee of the correct temporal behavior of the processes of the application. Instead, this paper presents a simulator which implements the hard real-time

schedulers [6]–[9]. The framework implements two independent simulators, which are interconnected through a shared memory: the robot simulator and the real-time task scheduler simulator. The robot simulator allows the user to define the working scene and the robotic application (functional level). The real-time task scheduler (task execution level) permits the user to define the set of tasks composing the application and to configure the functions relating the temporal parameters with the application. The scheduler selects in run-time the next task to be executed according to EDF policy [10] and guarantee that the set of tasks is schedulable at any time. Whenever a change on the workload arises, the EDF test is applied. We have selected EDF due to its low run-time overhead, which is convenient for the adjustments of the proposed flexible model of tasks. EDF is based on a periodic task model that permits to define robotic applications.

The actions performed by the tasks in the scheduler simulator (e.g. change speed) are transmitted to the robotic simulator for visualizing the simulation (i.e. increase/decrease the speed of the robot) through human-computer interface. Moreover, the robot simulator provides to the scheduler simulator through the shared memory, information related to the robot and its environment (i.e. crowded zone that implies speed reduction), which affect the change of the temporal constraints of the tasks (i.e. increase of the deadlines). Likewise, the monitoring of parameters of both of the simulators: robot progression, task execution schedule, speed, computation time, CPU utilization, deadline misses can be tracked in the framework.

Experiments of robot navigating in different environments (i.e. varying number of obstacles) are carried out to show how the tool serve as useful test-bed for the analysis of the behaviour of robotic applications. The presented tool is used to compare the proposed flexible task model that adapts to the environment to a task model where the temporal parameters are fixed (e. g. cannot adapt to speed changes). The analysis is performed with the different windows provided by the tool. The final results show that the flexible model improves the utilization of CPU in 15% on average, and the time of accomplishing the robot objectives, on average, in 32%.

The paper is organized as follows: After the introduction, section 2 discusses related work. Section 3 presents the real-time tasks scheduler simulator. Section 4 details the robot simulator and the possibilities to define the scenarios. Section 5 details the simulators interaction. Robotic navigation experiments and

evaluation of the performances of two task models is performed in Section 6. Section 7 summarizes the conclusions.

II. Related work

Rui Liu et al, describes in [11] and extensive review of the Natural-language-facilitated human-robot cooperation (NLC), where natural language and human movements are captured by robot sensors to cooperate in different fields: manufacturing, personal assistance and health cares. They also analyze the robot cognition level during the cooperation task, either training, control, task manager and social companion. Compared to this work, our platform is focused on the human interaction with robots to accomplish real-time deadlines and optimize the resource utilization.

Important amount of research have addressed the Human-Computer Cooperation (HCC) in robotic applications like imitation learning, learning by demonstration, cognitive systems, natural language recognition etc. [12][13][14]. However, more related to our work, there are some papers that focus on task planning related to human-computer coordination. Agostini, Torras, and Wörgötter [15] proposes a decision-making algorithm to schedule human-robot tasks based on STRIPS-like operators. Kwon and Suh [16] proposes a proactive planner with the use of temporal Bayesian networks to find feasible allocations of tasks. Their probabilistic estimation makes it not applicable to hard real-time tasks, what is the focus of our contribution.

Sekiyama, et al. in [17], allocate assembly tasks to humans and robots based on the stochastic petri nets. Multiple-Objective Optimization is conducted based on task finish time and payment cost of each assembly strategy configuration.

Gombolay et al. [18] propose a scheduling algorithm for the coordination of heterogeneous multi-agent systems, composed of humans and robots. As in our work, multiple robots working and moving in the same physical area, and a mixed-integer linear program is formulated to solve the problem of task assignment. Compared to the EDF scheduling proposed in our work, the linear programing produces a static solution before the start of operation, and does not adapt to the changes that arise during the run. The same authors [19] also present intelligent scheduling techniques showing that humans proceeded as safe guards of robot

behavior, and have a large range of improvisation in the decision-making, sometimes contrary to the decision-making authority.

Frontoni [20] presents a framework developed in Matlab to experiment and explore algorithms, which perform typical tasks carried out by a mobile robot, such as obstacle avoidance. The main feature of this simulator is the possibility to transmit these algorithms to a real robot.

To the best of our knowledge, none of them solve dynamically the accomplishment of hard real-time tasks for robot coordination and at the same time propose and interactive HCC framework to help in this interaction.

III. Real-time task scheduler simulator

The scheduler is an essential part of the simulation tool, because it guarantees the execution of the hard real-time tasks of the robot. Moreover, the scheduler has to face the temporal constraint variations when the system interacts with the environment.

A. Task model

The robot executes hard real-time periodic tasks. The set of periodic tasks is defined as: $T = \{T_1, T_2, ...T_n\}$. Each periodic task is defined by its temporal constraints: $T_i = \{C_i, P_i, D_i\}$ where C_i is the computation time, P_i the period and D_i the deadline. In each instant, the scheduler selects the next task to be executed following the EDF policy [10]. That is, the task with the nearest deadline. Finally, the scheduler simulator communicates to the robot simulator which task has been selected. And then the robot simulator shows the action performed by this task. The scheduler simulator permits the user to introduce a set of n periodic tasks using the interface shown in Fig. 1.

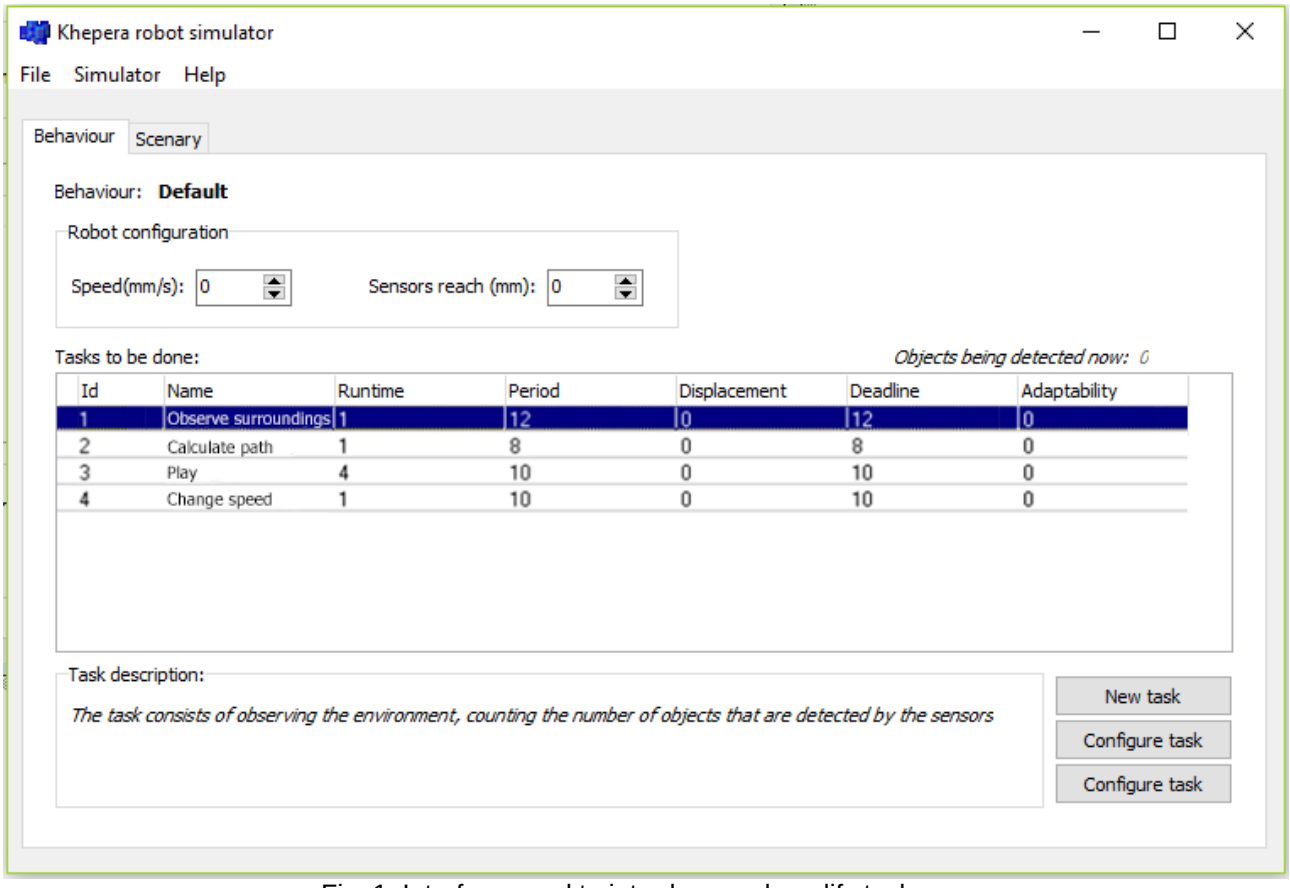
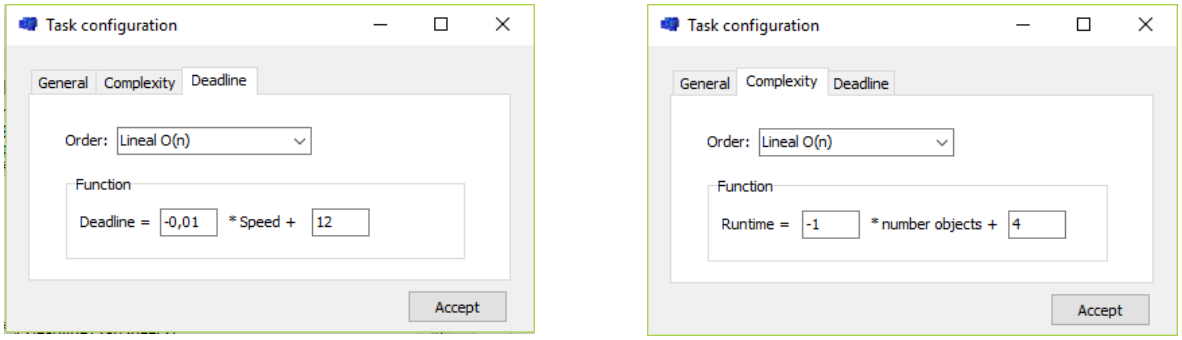


Fig. 1: Interface used to introduce and modify tasks.

The proposed flexible task model takes into account two kinds of environment variations that influence the temporal constraints of the tasks: the variations on the number of obstacles detected and the variations on the robot speed. The variation on the number of obstacles detected affects the computation time of the tasks. This relation is defined by the linear function $C_i = m \cdot \delta + n$ where C_i is the computation time of the task i, δ is the number of obstacles detected and m, n are values defined by the user.

In the same way, a robot speed variation influences the deadline of the tasks. In this case, the variation is defined by the linear function $D_i = k \cdot \gamma + t$ where D_i is the deadline of the task i, γ is the speed of the robot and k, t are values defined by the user. As mentioned above, the user defines the parameters m, n, k and t using the interface shown in Fig. 2.



a b

Fig. 2: Interface used to define the parameters: a) computation time function C. b) deadline function D.

The computation time variations and/or the deadline variations of the tasks could lead to the unschedulability of the set of tasks. This may happen if the deadline is reduced and/or the computation time is increased. To tackle this problem, the simulator checks that the schedulability test $U = \sum_{i=1}^{n} \frac{C_i}{P_i} < 1$ is satisfied whenever a change of the load arises.

In the schedulability test, U represents the processor utilization, n is the number of tasks, C_i is the computation time of the task i and P_i is the period of the task i. When the processor utilization U is close to 1, the scheduler reduces the speed of the robot. This reduction implies an increase of the deadlines of some tasks, therefore the system workload is reduced.

The graphical user interface designed facilitates the control and monitoring of the simulation. Regarding the control, the user can pause, stop and resume the execution at any time. In addition, the user can perform the simulation step by step. All these controls can be seen in the Fig. 3.

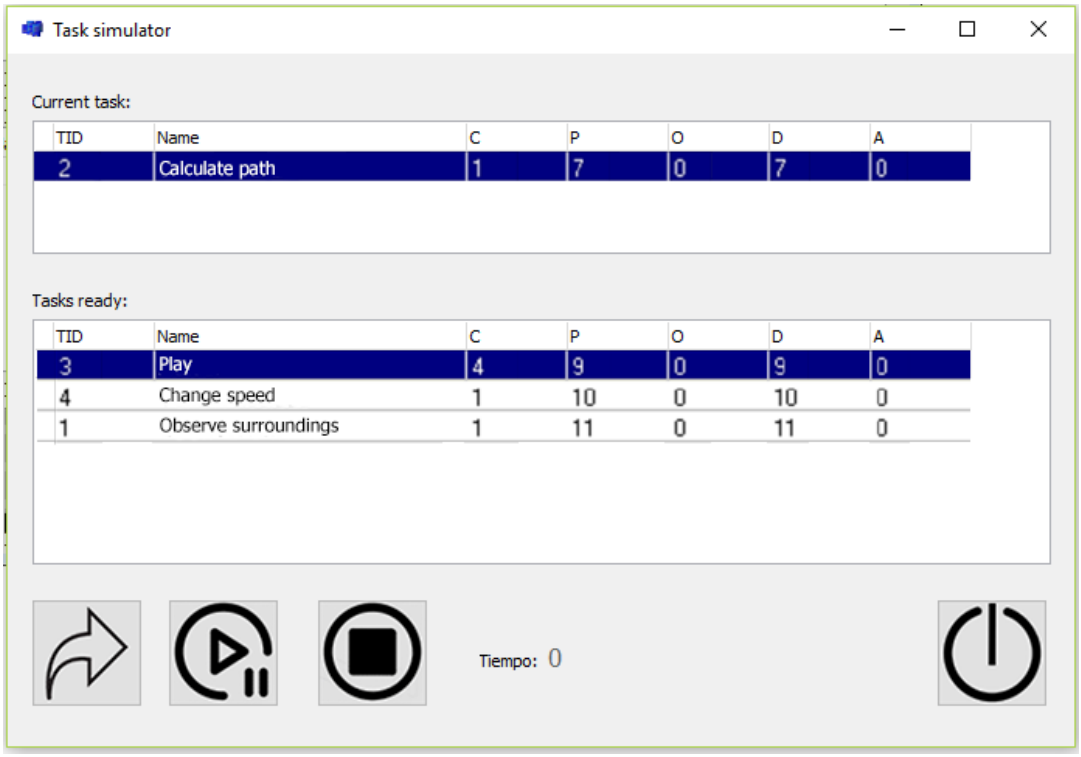


Fig. 3: Simulator interface.

Regarding the simulation monitoring, the simulator shows a set of graphs where the evolution of the parameters is represented. One of these graphs, the task execution schedule (Fig. 4), shows on a timeline which task is being executed in each moment. In this graph, the horizontal axis shows the time in seconds and the vertical axis shows the defined tasks. The Idle task represents the idle time of the processor.

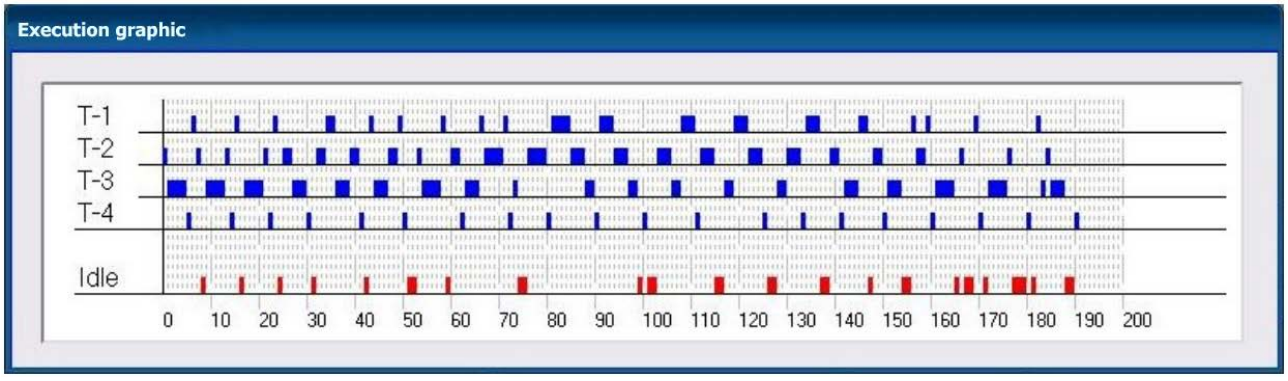


Fig. 4: Task execution schedule.

In the graphs of Fig. 5 the computation time and deadline of each task, the percentage of processor utilization and the robot speed are shown.

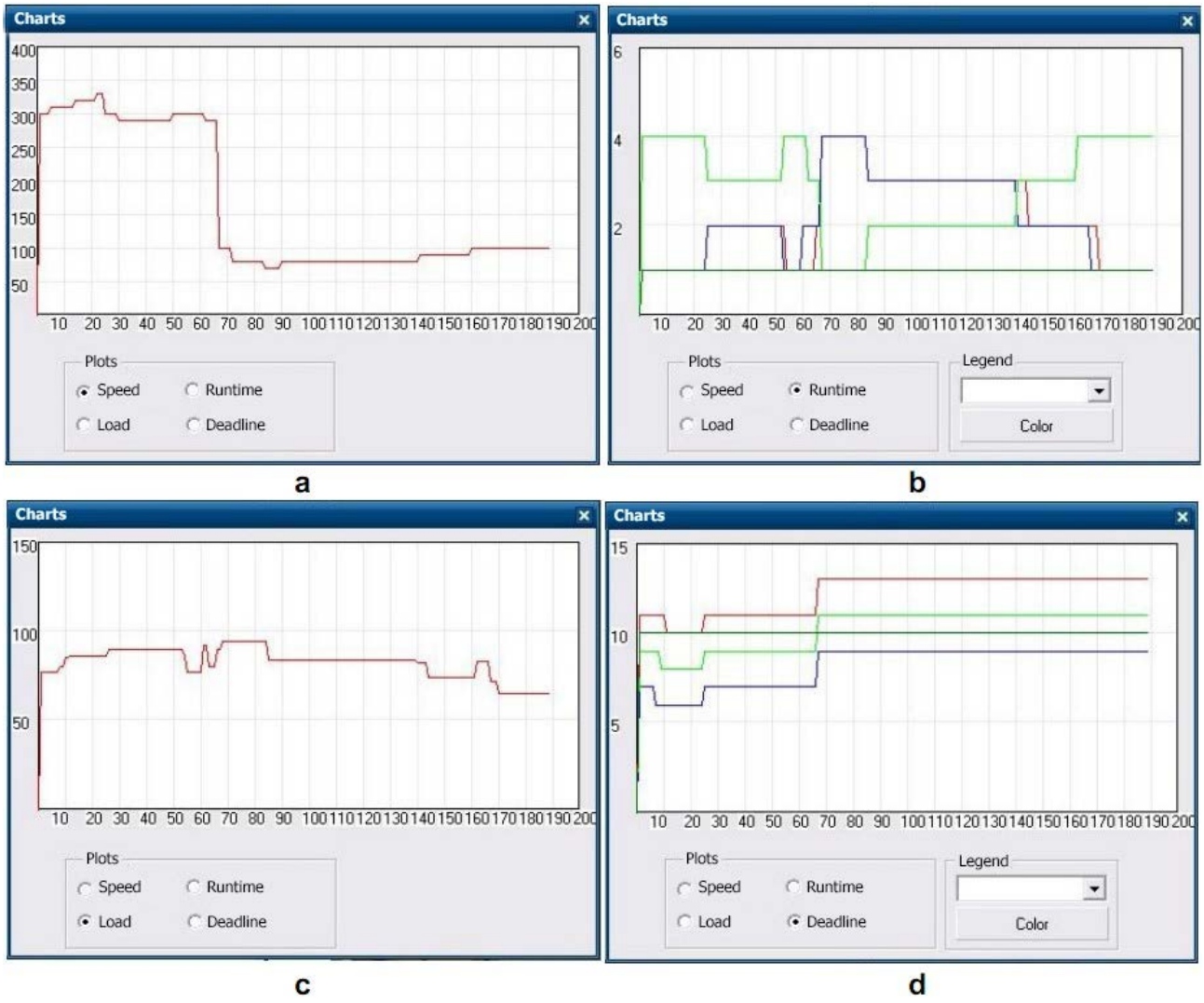


Fig. 5: Graphs that represent the evolution of the parameters: a) Robot speed b) Task Computation c) Processor Utilization d) Task Deadline.

IV. Robot simulator

The robot simulator executes the task selected by the scheduler. In this simulator, the aim of the robot is to reach a list of control points that are specified by the user. In order to reach these points, the implemented tasks cooperate. Moreover, these tasks may involve an interaction with the environment that affects the result of their execution. The following paragraph explains in detail how the simulator works and its graphical interface.

Firstly, the user introduces the information about the environment (the size of the scenario, the position and radius of the obstacles and the position of the control points) and then the information about the robot (its initial position and the range of its sensors). The interface of Fig. 6 shows the information introduced by

the user through a drawing of the scenario. Moreover, the user can save and load scenarios, allowing to combine different scenarios with different set of tasks in a simulation. The interface used to define the environment and the robot information is shown in Fig. 6.

Fig. 6: HCC Interface used to design scenarios.

Secondly, and once all the information is defined, the simulator stores it in a data structure. When the simulation starts, the scheduler simulator indicates to the robot simulator what task should be executed. Immediately, the robot simulator executes the code associated with this task. When the executed task interacts with the environment, the data structure mentioned previously is accessed. Then, the robot simulator reads and/or modifies this information. Finally, the simulator updates the environment and the robot at any time, so that the user knows in run-time the state of the scenario during the simulation.

As mentioned above, the tasks implemented in the simulator are oriented to achieve an overall goal: the robot has to reach the different control points defined by the user. Thus, the simulator implements the following tasks:

- Environment observation: reads the sensors to detect the nearby obstacles. If an obstacle is found within the range of the sensor, it is counted.
- Path calculation: simulates the calculation of the path that the robot has to follow to reach the next control point. The task takes into account the obstacles location provided by the task “Environment observation” to calculate the path.
- Actuation: simulates the activation of the actuators to move the robot. The task uses the path provided by the task “Path calculation”.
- Speed variation**:** varies the robot speed depending on the number of obstacles counted by the task “Environment observation”. If there are no detected obstacles, the task considers that the speed could be increased, because there is no danger of collision and vice versa. The fact that the robot does not find obstacles is used to predict that it is in an obstacle-free environment, with a high probability of continuing not encountering obstacles. When obstacles are detected, the robot predicts that it is operated in an environment with less freedom of movement.

Other tasks can be included easily in the simulator for other applications. Also, as future work, authors want to introduce an easier interface for describing tasks.

The user interface of the simulator shows the scenario at any moment (obstacles, position of the robot, range of its sensors, control points to reach, robot speed, etc.). Therefore, the user can verify and understand the context of the simulation. Moreover, the user can direct the movement of the robot during the simulation, deviating it of its trajectory. All these features can be seen in Fig. 7.

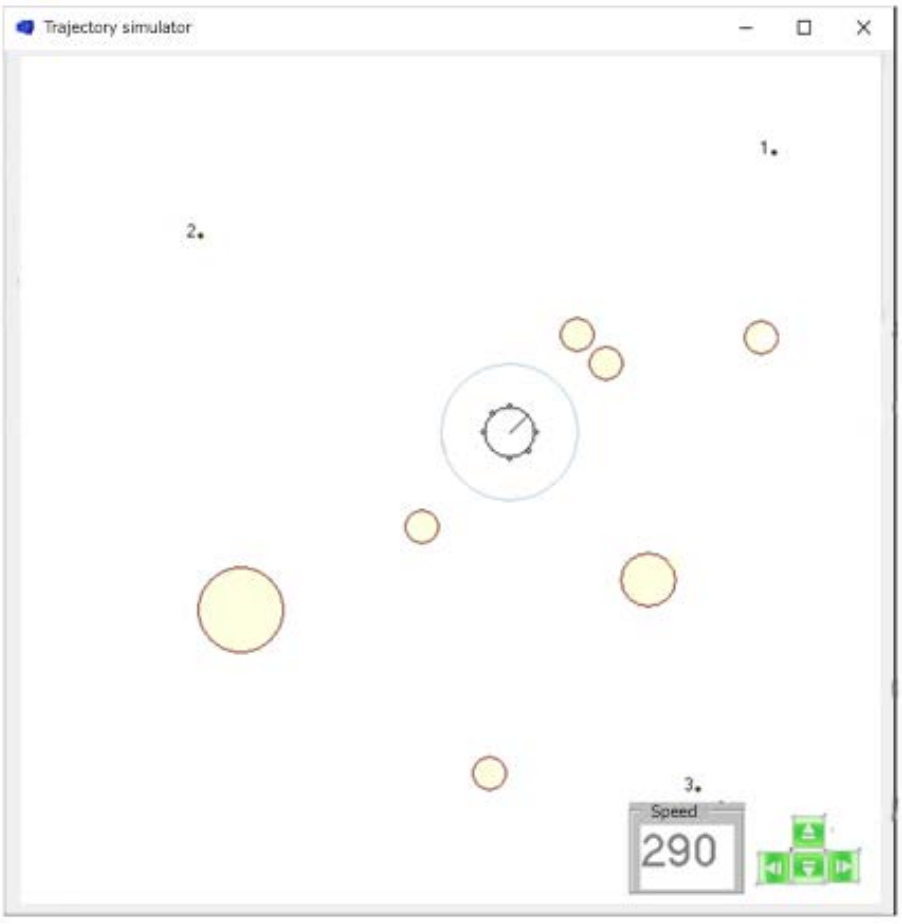


Fig. 7: Robot simulator interface.

V. Interaction between simulators

Once both simulators are implemented, they are interconnected through a shared memory and message passing as can be seen in Fig. 8. When the robot has to achieve a goal, it defines the processes involved in that goal. These processes are mapped to specific tasks that the scheduler has to execute.

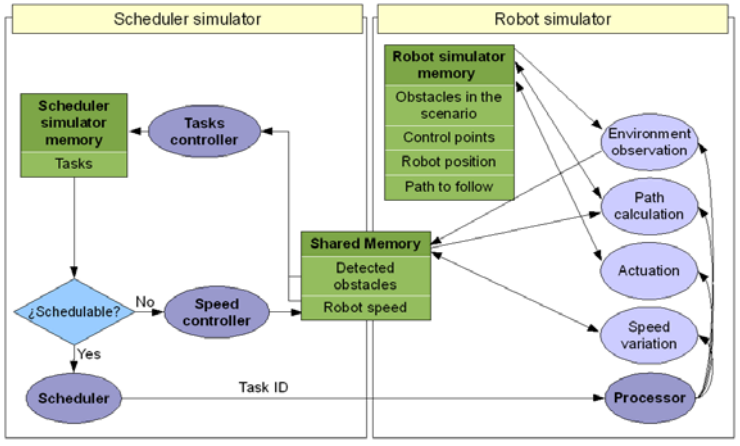


Fig. 8: Simulators interconnection.

The internal memory of the robot simulator stores information, which is only used by the robot as the obstacles of the scenarios, the control points, the position of the robot and the last path calculated. On the other hand, the shared memory stores information that is needed by both, the scheduler and the robot simulators such as, the robot speed and the detected obstacles.

When a task reaches its deadline, the task controller is started (Fig. 8). The task controller then reads the detected number of obstacles and the robot speed from the shared memory. Using this information and the C_i and D_i functions, it calculates the new computation time and the new deadline of the task. Later on, the scheduler simulator checks that the new set of tasks is schedulable. If so, then the next task to be executed is selected. Otherwise, the speed controller reduces the speed until the schedulability is achieved.

For the proposed example, the tasks use the memories as follows:

- Environment observation: reads from the robot simulator memory all the obstacles of the scenario, the current position of the robot and the range of the sensors. Using this information, this task calculates the obstacles detected by the robot. The number of obstacles is stored in the shared memory.
- Path calculation: reads from the robot simulator memory the next control point to be reached and the current position of the robot. It also reads the detected obstacles from the shared memory. Using this information, the path to be followed by the robot is calculated. This path is stored in the robot simulator memory.
- Actuation: reads from the robot simulator memory, the last path calculated. From this information, the task calculates the new position of the robot and stores it in the memory of the robot simulator.
- Speed variation: reads from the shared memory the number of detected obstacles. Based on this information, this task calculates the new speed, which is stored in the shared memory.

The simulators have been developed in c++ language using de Integrated Development Environment C++ Builder from Embarcadero [21], an integrated development environment which allows to combine easily the design of graphical interfaces and the low-level programming. The framework is composed of a set of cooperating classes with a well-defined functionality (Fig. 9).

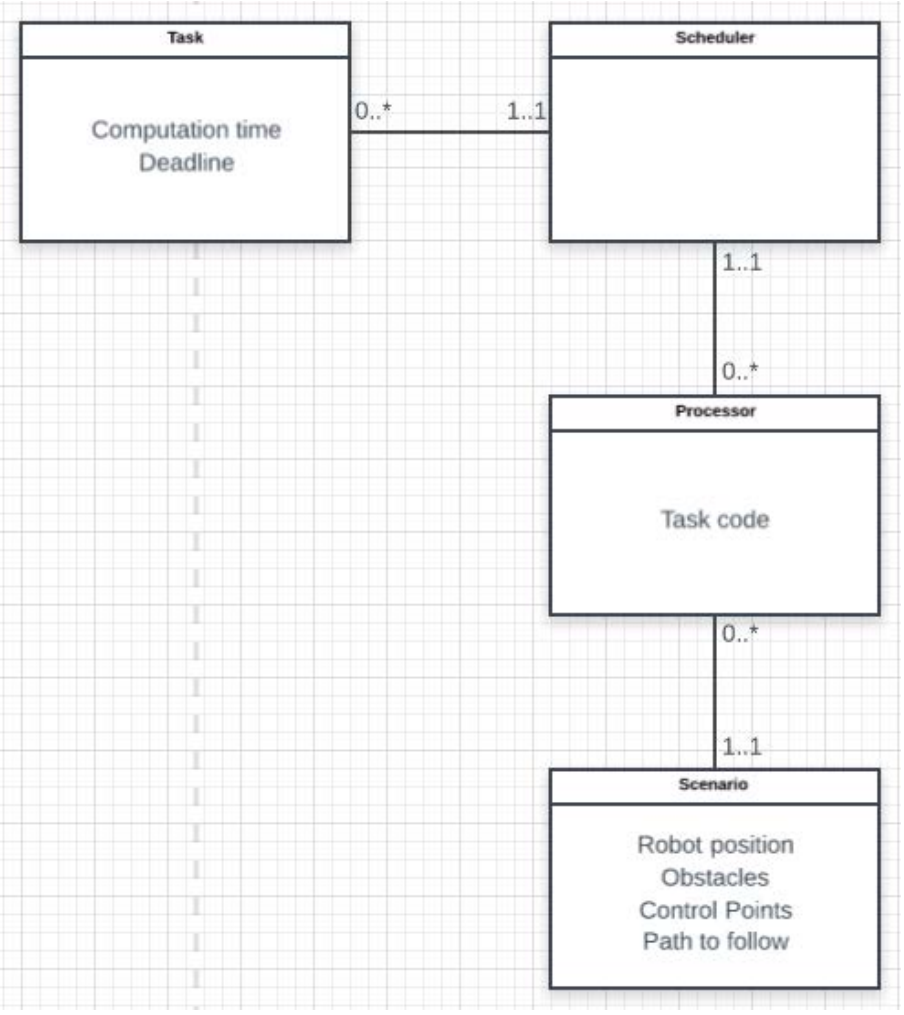


Fig. 9: Relationship between classes. R: read. W: write.

The most relevant classes are:

- Task class: contains the temporal constraints of a task (deadline, computation time, etc.) and its associated computation time C_i and deadline D_i functions.
- Scheduler class: implements the scheduler algorithms to execute the tasks, which remain stored in a linked list. Moreover, it updates the temporal constraints of the tasks using the information stored in the shared memory and the C_i and D_i functions. The methods of the class permit to know which task should be executed at any time.

- Processor class: implements the code of the tasks. When it receives the id of a task (provided by the scheduler class), it executes its associated method. The execution can alter the status of the scenario, in this case the scenario class is accessed. Some tasks, like environment observation and speed variation, update the information stored in the shared memory.
- Scenario class: stores the information related to the robot and the environment (obstacles, control points, robot position, etc.) in a data structure. This class provides methods to introduce, modify and retrieve the status of the scenario. Some relevant structures used by this class, such as the robot, the obstacles or the control points, are implemented using defined types.

VI. Experimental evaluation

In order to show the usefulness and flexibility of the simulation tool when analyzing the performance of algorithms involved in the design of real-time robotic applications, the proposed flexible task model is compared to a traditional task model with fixed temporal parameters. The arrival time of the robots and the CPU utilization are evaluated to test the performance of our proposal (configuration A), which is compared with a system of fixed temporal constraints (configuration B). In the definition of both experiments, the risk level considered is the same. A simplified example of only 4 tasks has been used to illustrate the adjustment mechanism of the proposed model A.

The two different configurations are set up as follows:

- Configuration A: The temporal constraints of the tasks vary depending on the robot speed and the number of obstacles detected as shown in equation C_i and D_i.
- Configuration B: The temporal constraints of the tasks are fixed. Their values are defined off-line as:

  - The computation time is the worst-case execution time corresponding to a situation when the robot detects the maximum number of obstacles.

  - The robot speed is chosen in a way that minimizes the risk of collision (90 mm/s)

  - The deadline is calculated according to the previous robot speed.

The simulated robot is inspired in the khepera robot that has two wheels and 6 infrared sensors for obstacle avoidance. The speed of the robot has a range of [0, 1m/s].

A. Experiment definition

The experiment is defined as follows: the robot must reach a control point in an environment plenty of obstacles. The parameters used to define the scenario are shown in Table 1. It is worth noting that the origin of the coordinates (0, 0) is located in the upper left corner of the scenario, as is shown in Fig. 10.

Parameters of the scenario		
Robot parameters	Initial position (mm):	(150,850)
	Initial angle:	0º
	Range of the sensors:	100 mm
	Initial speed:	100 mm/s
Environment parameters	Obstacles position:	(525,617), (420,495), (635,420), (535,340), (388,278), (478,425), (333,493), (710,388), (598,620), (630,730)
	Obstacles radius:	30 mm
	Control points (mm):	(875,125)

Table 1: Scenario parameters of the experiment.

The initial position of the robot is (150 mm, 850 mm) and its initial speed is 100 mm/s. Its goal is to reach the control point (875 mm, 125 mm) located at the other end of the scenario. During its trajectory, the robot will find many obstacles. The Fig. 10 shows the scenario during the simulation of this experiment.

As mentioned above, this experiment is executed using A and B configurations. Next, the parameters of the configurations are described.

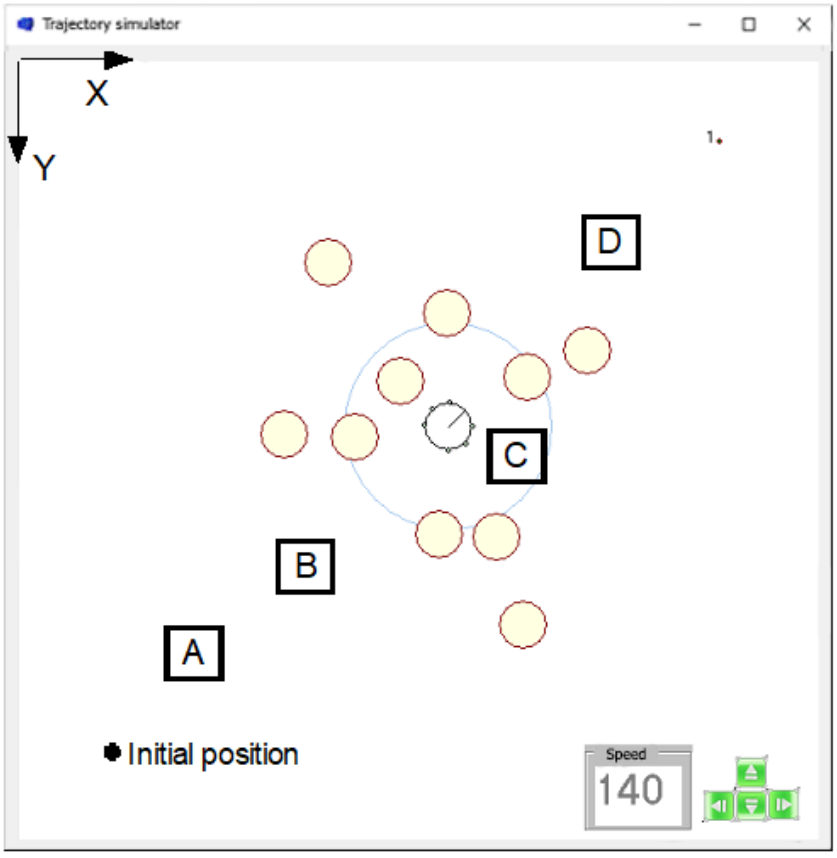


Fig. 10: Scenario during the simulation of the experiment

- Configuration A: Using variable temporal constraints.

This configuration uses the characteristics of the flexible task model. Thus, the temporal constraints of the tasks vary depending on the speed of the robot and/or the number of obstacles detected. These variations are defined by the functions C_i and D_i which are expressed in the Table 2. We assume that the frequency of the sensory tasks depends on the speed of the robot because as the speed increases the reaction time to avoid the potential collisions decreases. Therefore the frequency of these tasks must increase. In relation to the computation time of the observation tasks, it increases with the number of obstacles detected in the neighbor environment of the robot because the task has to calculate the distance to each obstacle to obtain a map of the environment.

Regarding the C_i function, the value of the independent term corresponds to the computation time when the robot does not detect obstacles $\delta = 0$. These values have been assigned taking into account the nature of each task. The dependent term of the function C_i determines how the computation time will vary with

respect to the number of detected obstacles variations. “Environment observation” and “Path calculation” tasks have different dependent terms to cause different behavior in the simulation. “Actuation” and “Speed variation” tasks have their dependent term equal to zero, because it is considered that their computation time is constant.

Temporal requirements	
Task 1: Environment observation	$C_1 = 0.5 \cdot \delta + 4$
	$D_1 = -0.011 \cdot \gamma + 14$
Task 2: Path calculation	$C_2 = 0.25 \cdot \delta + 3$
	$D_2 = -0.011 \cdot \gamma + 19$
Task 3: Actuation	$C_3 = 1$
	$D_3 = 19$
Task 4: Speed variation	$C_4 = 2$
	$D_4 = -0.01 \cdot \gamma + 15$

Table 2: Functions used for the tasks in the configuration A.

Regarding the D_i function, the value of the independent term corresponds to the deadline when the robot speed is equal to zero $\gamma = 0$. These values have been assigned taking into account the nature of each task. The dependent term of the D_i function determines how the deadline will vary with respect to the speed variations. The dependent term is negative in order to allow the deadline decrease when the speed increases. The values have been assigned taking into account that the speed range is normally between 0 mm/s and 1000 mm/s. “Speed variation” task has a different dependent term to cause different behavior in the simulation (change its frequency). “Actuation” task has the dependent term equal to zero, because it is considered that its deadline is constant.

- Configuration B: Using fixed temporal constraints.

In this case, the computation time is selected for the more critical situation of the experiment (i.e. worst-case execution time) and the deadline is selected according to a speed that minimizes the risk of collision (90 mm/s). So, the values resulting for each task are expressed in the Table 3.

Temporal requirements	
Task 1: Environment observation	$C_1 = 7$
	$D_1 = 13$
Task 2: Path calculation	$C_2 = 4$
	$D_2 = 18$
Task 3: Actuation	$C_3 = 1$
	$D_3 = 19$
Task 4: Speed variation	$C_4 = 2$
	$D_4 = 14$

Table 3: Functions used for the tasks in the configuration B.

B. Experimental results

- Configuration A: Using variable temporal constraints.

After the execution of the experiment, the simulator shows the following results:

While there are no obstacles, the task 4 (speed variation) increases the speed, because there is no danger of collision. This can be seen in the Fig. 11a.

At time 5s (point A in Fig. 10 and Fig. 11), the robot speed exceeds 100 mm/s (Fig. 11a). Consequently, the deadline of the task 4 (speed variation) is reduced in one unit (Fig. 11d). As a result of this decrease, the percentage of processor utilization is increased (Fig. 11c).

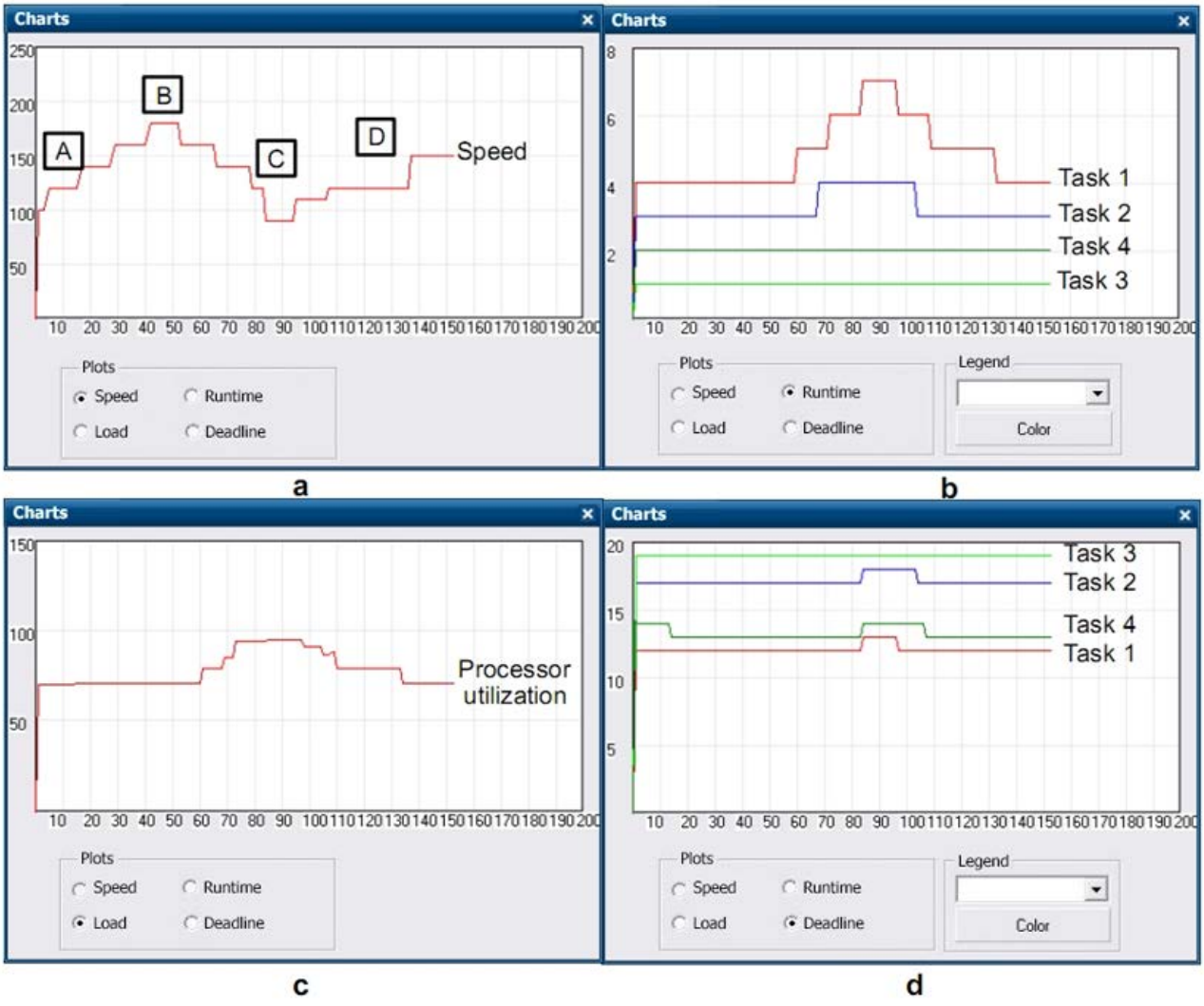


Fig. 11: Evolution of the parameters using the configuration A: a) Robot speed b) Task computation time c) Processor utilization d) Task deadline.

At time 52 s (point B in Fig. 10 and Fig. 11) the robot detects two obstacles. After that, the computation time of the task 1 (environment observation) is increased in one unit. This increase is applied when the task reaches its deadline, which is at time 60 s (Fig. 11b). In this way, the percentage of processor utilization is increased again. When the task which varies the speed (Task 4) is executed at time 54 s, it decreases the speed, because there is danger of collision.

At time 65 s the robot detects four obstacles. Consequently, the task 1 and 2 increase their computation time in one unit. This increase is applied when the tasks reach their deadline, which is at time 68 s and 72 s respectively (Fig. 11b). In this way, the percentage of processor utilization is increased again.

At time 85 s (point C in Fig. 10 and Fig. 11) the robot detects six obstacles and the task 1 increases its computation time in one unit. This increase implies an excessive workload for the processor. So, in order to secure that the percentage of processor utilization does not exceed 100%, the system reduces the speed of the robot to 90 mm/s (Fig. 11a). Therefore, the deadlines of the tasks increase and the percentage of processor utilization remains bellow 100%.

Thereafter, the robot leaves the area with obstacles (point D in Fig. 10 and Fig. 11), so the computation time of the tasks and the workload of the processor are decreased. Consequently, the speed is increased again.

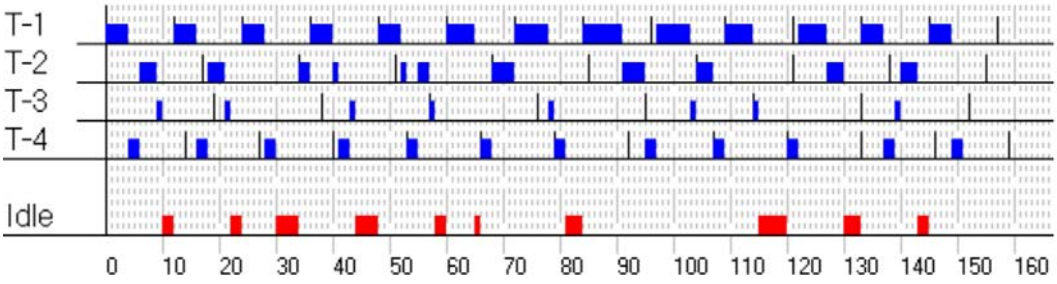


Fig. 12: Task execution schedule of experiment using the configuration A.

It can be seen in Fig. 12 how the computation time, deadlines and periods of task 1 and task 2 vary along the time to accomplish the robot goal.

- Configuration B: Using fixed temporal constraints.

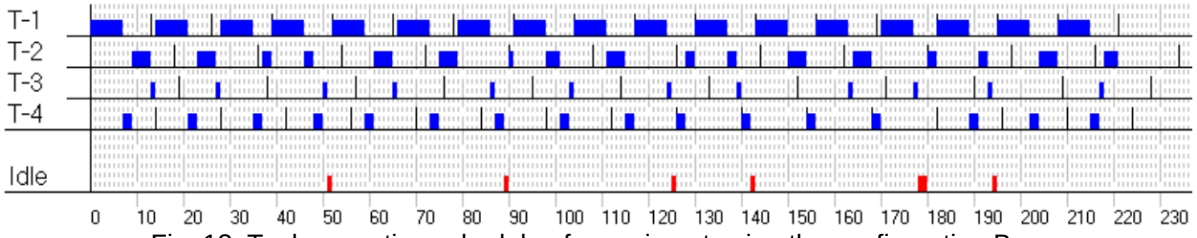


Fig. 13: Task execution schedule of experiment using the configuration B

It can be seen in Fig. 13 that the temporal constraints of the tasks are constant and equal to the values defined in Table 3.

C. Comparative analysis

The two task models have been evaluated over 200 different scenarios using a varied number of tasks (4-20). In these scenarios, the numbers of obstacles and their positions have been selected randomly as well

as the robot position. Likewise, the computation time, the deadlines and the periods have been randomly generated by composing utilization task sets in the range of 40% to 90% CPU utilization. The arrival time of the robot to the destination and the processor utilization of the system to accomplish the objective are evaluated using both configurations. The average results can be appreciated in the Figure 14 and 15.

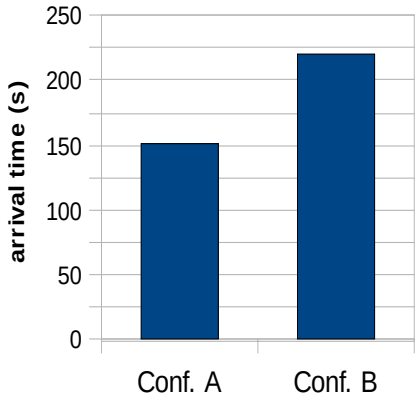


Fig. 14: Arrival time

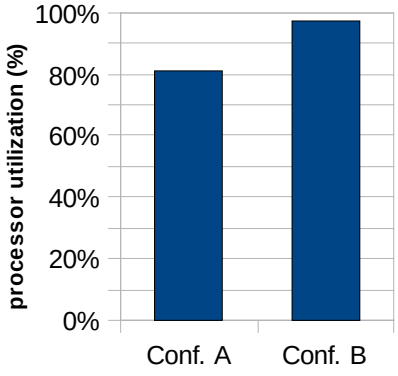


Fig. 15: Processor utilization

From the comparative chart shown in Fig. 14 and Fig. 15, it can be concluded that, for the same robot application (defined in Fig 10):

- The configuration A reduces the processor utilization in 15%.
- Using the configuration A, the robot needs a 32 % less time to achieve its goal than using the configuration B.
- The configuration A reduces in 42% the time dedicated by the CPU to execute the tasks.

Consequently, the flexible task model of the developed simulator allows to conclude that the time of achieving the robot goal and the workload of the processor have been decreased significantly compared to a fixed-characteristic task model.

VIII. Conclusions

This paper presents a human-computer cooperation framework for real-time robotic applications. This tool is composed of a scheduler simulator and a robot simulator. The robot simulator permits to define robotic applications and the scheduler simulator allows defining task models and scheduling algorithms for the guarantee of the execution of the tasks of these applications. The implementation and the interconnection of

the two simulators have been presented. A flexible scheduler has been incorporated in the simulation environment, which allows the tasks adapt their temporal constraints to the environment characteristics.

In order to show the usefulness of the proposed simulation framework for analyzing and evaluating real-time robotic systems, different robotic applications have been simulated. The tool allows us the monitoring and visualization of the execution of the robot objectives, as well as the trace of the detailed execution of the tasks using the chronograms. Moreover, it has permitted us to compare the behaviour of two different schedulers. One using fixed temporal parameters and a flexible scheduler using adaptive parameters depending on the environment of the robot. The results show that the proposed flexible algorithm improves on 15% the processor utilization and reduces the time needed to achieve the goals in 32%.

IX. Future work

As a future work, we plan to improve the way the user may define the tasks in the system, moving from C functions to a graphical interface.

References

[1] C. Dominguez, H. Hassan, and A. Crespo, “Real-Time Embedded Architecture for Pervasive Robots”, in *The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007)*, 2007, pp. 531–536.

[2] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, “Fixed priority pre-emptive scheduling: An historical perspective", *Real-Time Systems*, vol. 8, no. 2–3, pp. 173–198, 1995.

[3] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar, “Opportunities and obligations for physical computing systems", *Computer*, vol. 38, no. 11, pp. 23–31, Nov. 2005.

[4] Z. Zhen, C. Qixin, C. Lo, and Z. Lei, “A CORBA-based simulation and control framework for mobile robots", *Robotica*, vol. 27, no. 3, p. 459, May 2009.

[5] G. Ferretti, G. Magnani, P. Porrati, G. Rizzi, P. Rocco, and A. Rusconi, “Real-Time Simulation of a Space Robotic Arm", in *IROS*, 2008.

[6] A. Qadi, S. Goddard, Jiangyang Huang, and S. Farritor, “A Performance and Schedulability Analysis of an Autonomous Mobile Robot", in *17th Euromicro Conference on Real-Time Systems (ECRTS’05)*,

pp. 239–248.

[7] G. R. Goud, N. Sharma, K. Ramamritham, and S. Malewar, “Efficient Real-Time Support for Automotive Applications: A Case Study", in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA’06)*, 2006, pp. 335–341.

[8] P. Pedreiras and A. Luis, “The flexible time-triggered (FTT) paradigm: an approach to QoS management in distributed real-time systems", in *Proceedings International Parallel and Distributed Processing Symposium*, p. 9.

[9] Huan Li, J. Sweeney, K. Ramamritham, R. Grupen, and P. Shenoy, “Real-time support for mobile robotics", in *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings.*, pp. 10–18.

[10] H. Chetto and M. Chetto, “Some Results of the Earliest Deadline Scheduling Algorithm", *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1261–1269, Oct. 1989.

[11] R. Liu and X. Zhang, “Systems of natural-language-facilitated human-robot cooperation: A review", Jan. 2017.

[12] P. Tsarouchi, S. Makris, and G. Chryssolouris, “Human-robot interaction review and challenges on task planning and programming", *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 8, pp. 916–931, Aug. 2016.

[13] A. Moniz, “Organizational concepts and interaction between humans and robots in industrial environments", in *Conference: IEEE-RAS-IARP Joint Workshop on Technical Challenges for Dependable Robots in Human Environment, At Tokyo*, 2013.

[14] M. P. Mayer, B. Odenthal, M. Faber, C. Winkelholz, and C. M. Schlick, “Cognitive Engineering of Automated Assembly Processes", *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 24, no. 3, pp. 348–368, May 2014.

[15] A. Agostini, C. Torras, and F. Wörgötter, “Integrating Task Planning and Interactive Learning for Robots to Work in Human Environments.", *IJCAI*, 2011.

[16] W. Kwon and I. Suh, “Planning of proactive behaviors for human–robot cooperative tasks under uncertainty", *Knowledge-Based Systems*, 2014.

[17] Fei Chen, K. Sekiyama, H. Sasaki, Jian Huang, Baiqing Sun, and T. Fukuda, “Assembly strategy

modeling and selection for human and robot coordinated cell assembly", in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4670–4675.

[18] M. Gombolay, R. Wilcox, A. Diaz, and F. Yu, “Towards successful coordination of human and robotic work using automated scheduling tools: An initial pilot study", *Robotics: Science and …*, 2013.

[19] M. C. Gombolay, R. A. Gutierrez, S. G. Clarke, G. F. Sturla, and J. A. Shah, “Decision-making authority, team efficiency and human worker satisfaction in mixed human-robot teams", *Autonomous Robots*, vol. 39, no. 3, pp. 293–312, Oct. 2015.

[20] E. Frontoni, A. Mancini, F. Caponetti, and P. Zingaretti, “A framework for simulations and tests of mobile robotics tasks", in *2006 14th Mediterranean Conference on Control and Automation*, 2006, pp. 1–6.

[21] I. Embarcadero Technologies, “C++ Builder 10.2.” [Online]. Available: https://www.embarcadero.com/.