Target-sensitive systems: Analysis and Implementation Issues

Giorgio Buttazzo, Carmelo Di Franco, Mauro Marinoni
Scuola Superiore Sant'Anna, Pisa, Italy
g.buttazzo@sssup.it, c.difranco@hotmail.it, m.marinoni@sssup.it

***Abstract*—Several real-time applications include tasks in which the output must be produced at precise time instants, rather than "within" a deadline, and the overall system performance significantly degrades when the task is executed too late or too early with respect to the desired time.**

This paper illustrates one of such applications and takes it as a reference case study to propose a general approach to show how to derive the timing constraints from the application requirements, how to implement the application on top of a real-time kernel, identifying the operating system features necessary to enforce such constraints, and how to analyze the schedulability of the task set. A set of experimental results are also presented to validate the proposed approach.

I. Introduction

Most of the real-time scheduling theory has been developed for time sensitive applications in which computational activities have to be executed *within* a specified deadline. Examples of such tasks include periodic sensory acquisition and control in robotics applications, image processing activities, video encoding and decoding in multimedia applications, and many others.

There are other real-time applications, however, in which the output must be produced at a precise time instant (*target time*), and the execution jitter strongly affects the performance of the system. A number of perceptual studies have shown that for streams of individual audio events, the human ear is capable of perceiving a time jitter on the order of one millisecond, particularly in the context of rhythmically complex and syncopated ensemble music [17], [14]. As a consequence, in computer music, the notes generated by the program must be played at precise time instants, with a tolerance of a few milliseconds. To express such a property, Dannenberg and Jameson [7] proposed a task model in which the deadline is considered the best time at which a task should execute (non preemptively), rather than the maximum time instant at which the execution should complete. Hence, a performance degradation is perceived by the listener not only when a note is played too late, but also when it is played too early with respect to the specified time. Then, Brandt and Dannenberg [5] discussed the main features that a real-time operating system should have to support low-latency music software.

Other applications in which tasks must be executed "on time" with a given tolerance are those in which a moving target has to be caught by a robot system [13], [1], [6], [8], [16]. In this case, the tolerance left to the scheduler depends on the target size and speed.

The problem of scheduling real-time tasks at precise time instants has been addressed by many authors from different perspectives. For instance, Farzinvash and Kargahi [9] introduced the concept of Instant Value Function (IVF), according to which the exact instant where a job is executed affects the accrued value, and presented a scheduling algorithm that tries to maximize the total accrued value of the system.

Guerra and Fohler [12], [11] proposed a gravitational model to schedule overlapping events around the same time instant. To do that, each task is assigned a different importance value, considered as a mass hanging on a pendulum centered at the desired time and used to compute the mass distribution along the timeline based on the equilibrium state.

Tidwell et al. [19] solved a Markov Decision Process formulation of the scheduling problem and derived value-optimal scheduling policies for periodic task sets and stochastic non-preemptive execution intervals.

a) Contributions of the work: This paper describes how to implement and analyze real-time applications in which one or more tasks must be executed at precise time instants, with a given maximum tolerance, and may contain non preemptive regions. Starting from a real application, the paper first shows how to derive timing parameters from the application requirements, including periods, activation times, safety intervals, and tolerances. Task implementation issues are also discussed, illustrating the main operating system features necessary to enforce such constraints. Then, a method for analyzing the schedulability of the task set is proposed both under fixed priorities and Earliest Deadline First (EDF). Finally, we present some experimental results that show the performance of the system and validate the proposed approach.

b) Organization of the paper: Section II illustrates the application used as a case study throughout the paper. Section II-C discusses some implementation issues; Section III presents the schedulability analysis; Section IV reports some experimental results carried out on the physical system; and Section V states our conclusions.

II. A case study

Let us consider the system illustrated in Figure 1, where a ball has to be shot against a moving target consisting of a hole on a rotating disc, whose angular velocity ω is unknown and must be estimated by a sensor (e.g., a camera located in front of the disc). The value of ω is hypothesed to change slowly and can be considered constant during a rotation.

The shooting device is assumed to be correctly aligned to hit the target when it is located in the upper position of the disc (firing position), as shown in the figure. Hence, the objective of the computing system is to determine the exact time instant at which the gun has to fire to hit the target in the upper disc position.

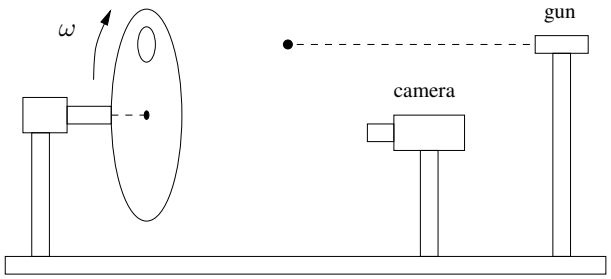

Fig. 1. The system used as a case study.

It is worth observing that the real-time issues presented under the considered case study can also be found in many other real world applications, from ignition control in automotive systems to target tracking in defense military systems.

A. Deriving timing constraints and tolerances

A simple method for estimating the angular velocity ω of the disc is to divide the difference of two consecutive angular readings ($\theta - \theta_{old}$) by the task period, that is

$$\omega = \frac{\theta - \theta_{old}}{T_s}. \tag{1}$$

If ε_θ is the position error due to the angular measurement, the error in the computed speed is equal to

$$\varepsilon_\omega = \frac{2\varepsilon_\theta}{T_s}. \tag{2}$$

Note that, if the rotation direction is unknown, the difference of two consecutive target samples cannot exceed π, that is

$$|\omega| \leq \frac{\pi}{T_s}. \tag{3}$$

Moreover, to guarantee different consecutive angular readings (i.e., to guarantee that $\theta \neq \theta_{old}$), the angle covered by the disc in the sampling period T_s must be greater than $2\varepsilon_\theta$, which gives a lower bound on ω:

$$|\omega| \geq \frac{2\varepsilon_\theta}{T_s}. \tag{4}$$

For a given T_s, equations (4) and (3) provide a feasibility range for ω to be correctly estimated. Vice versa, if the rotation speed is known to be in a limited range [ω_{min}, ω_{max}], then the sampling period T_s has to be bounded by

$$\frac{2\varepsilon_\theta}{\omega_{min}} \leq T_s \leq \frac{\pi}{\omega_{max}}. \tag{5}$$

The angular difference with respect to the firing position θ_0 can be computed as

$$\Delta\theta = \theta_0 - \theta + \begin{cases} 2\pi & \text{if } \frac{\theta_0 - \theta}{\omega} < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Note that $\Delta\theta$ is always positive and represents the angular distance that the disc has to cover from the current target position θ to the firing position θ_0 in the direction of rotation.

The corresponding time needed to cover such an angular distance (assuming a constant rotation speed) is given by

$$\Delta t = \frac{\Delta\theta}{|\omega|}. \tag{7}$$

Hence, the absolute time t_0 at which the hole will reach the firing position θ_0 is given by

$$t_0 = t + \Delta t = t + \frac{\Delta\theta}{|\omega|}. \tag{8}$$

Time t_0 also represents the time at which the ball must be in the hole, hence the gun must be triggered a time t_a in advance to allow the ball to cross the distance D at speed v_B, that is

$$t_a = \frac{D}{v_B}. \tag{9}$$

Hence, the shooting time must be set at $t_s = t_0 - t_a$; that is,

$$t_s = t + \frac{\Delta\theta}{|\omega|} - \frac{D}{v_B}. \tag{10}$$

It is worth observing that the time interval Δt estimated by Equation (7) is affected by an error ε_t, which is given (in the worst case) by

$$\varepsilon_t = \frac{\Delta\theta + \varepsilon_\theta}{|\omega| - \varepsilon_\omega} - \frac{\Delta\theta}{|\omega|}. \tag{11}$$

If the center of the hole is at distance R from the disc center, such a timing error, at speed ω, generates a maximum target displacement of $\varepsilon_t|\omega|R$, which can be tolerated only if it does not exceed the difference between the hole radius R_H and the ball radius R_B, that is, if

$$\varepsilon_t|\omega|R \;\leq\; R_H - R_B. \tag{12}$$

Note that the difference $R_H - R_B$ is actually the chord approximation of the corresponding arc, which is acceptable for small angular values. For this reason, its value represents radians, and the angular tolerance ρ that guarantees a successful shot the target is

$$\rho = \frac{R_H - R_B}{R}. \tag{13}$$

Hence, the inequality expressed in Equation (12) can be written as

$$\varepsilon_t \;\leq\; \frac{\rho}{|\omega|}. \tag{14}$$

Substituting Equation (11) into Equation (14), it is possible to derive a safety condition as a function of $\Delta\theta$:

$$\Delta\theta \;\leq\; |\omega|T_s\frac{\rho - \varepsilon_\theta}{2\varepsilon_\theta} - \rho. \tag{15}$$

Hence, the maximum angular difference $\Delta\theta_{max}$ at which a prediction still guarantees centering the target, at speed ω, is given by

$$\Delta\theta_{max} = |\omega|T_s\frac{\rho - \varepsilon_\theta}{2\varepsilon_\theta} - \rho. \tag{16}$$

Observe that a value $\Delta\theta_{max} > 2\pi$ means that the target hit can be guaranteed even by planning the shoot more than one rotation ahead.

Also note that $\Delta\theta_{max}$ decreases with ω; in fact, small speed values increase the interval Δt in which the speed error ε_ω is integrated, hence the $\Delta\theta_{max}$ must be reduced to keep the angular drift within the tolerance ρ.

Similarly, a minimum angular difference $\Delta\theta_{min}$ is required to allow the ball reaching the target, given by

$$\Delta\theta_{min} = |\omega| t_a = |\omega| \frac{D}{v_B}. \quad (17)$$

In this case, a value $\Delta\theta_{min} > 2\pi$ indicates that the target hit must be planned at least one rotation ahead.

It follows that to guarantee a correct behavior, the shooting time can be set only when, at time t, the angular distance $\Delta\theta$ of the target from the firing position is within the safe interval $[\Delta\theta_{min},\ \Delta\theta_{max}]$.

B. Task structure and functional behavior

The application can be organized into three tasks interacting through a shared buffer, as illustrated in Figure 2:

- Task τ_1 (`estimate_speed`) is a periodic task that reads the camera with a period $T_1 = T_s$, computes the current angle θ of the target and, using the previous angular value (θ_{old}), estimates the current angular velocity ω of the disc according to Equation (1). The estimated value is written into a buffer shared with task τ_2.
- Task τ_2 (`plan_shooting`) is a periodic task that at each period T_2 reads the angular velocity ω from the buffer, reads the current target position θ and, if the target is within the safe region, computes the shooting time t_s at which the gun has to be triggered, using Equation (10). Such a value is used to post an event at time t_s that will activate task τ_3, which actually triggers the gun to fire.
- Task τ_3 (`fire`) is an aperiodic task that just triggers the gun to fire as soon as it is activated.

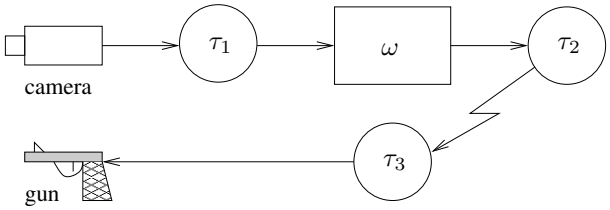

Fig. 2. Tasks structure of the application.

The application functional behavior can be described by the state diagram shown in Figure 3. The system starts in the `IDLE` state waiting for a "fire request". When the user issues a "fire request", the system enters the `ARMED` mode, where the disc is periodically sampled until the target is found in the safe region. As soon as the target is found in the safe region, task τ_2 computes the next firing instant t_s, posts the firing event, and moves the system to the `LOCKED` state, where it waits for the gun to fire. At time t_s, after shooting the ball, task τ_3 brings the system back to the `IDLE` state.

C. Implementation issues

This section discusses some timing issues related to the task implementation of the activities. The following functions are used in the pseudo code:

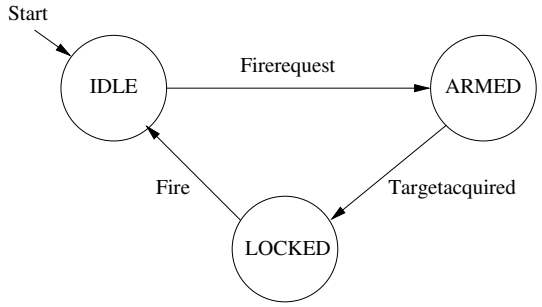

Fig. 3. States diagram of the application.

- `current_time()` returns the current time of the system;
- `set_state(value)` sets the system state at the value of the argument, which can be `IDLE`, `ARMED`, or `LOCKED`;
- `read_sensor()` returns the current angular position of the target;
- `write_buffer(value)` writes the passed value in a shared buffer;
- `read_buffer()` returns the value of the variable in the shared buffer;
- `post_task(`τ`,`t_s`)` activates task τ at time t_s;
- `wait_for_next_activation()` suspends the execution of the task until the beginning of the next period.

Moreover, we assume that at system initialization, the state is set to `IDLE` and that the system is brought in the `ARMED` state by a fire request coming from the user (e.g., by pressing a button).

1) Estimating the target velocity: If task τ_1 (`estimate_speed`) is executed at the highest priority, then two consecutive job executions are exactly separated by an interval of time equal to the task period T_s, hence the disc angular velocity can be estimated using Equation (1). A possible implementation of the task (referred to implementation A) is shown in Figure 4. Function `read_sensor()` acquires the image from the camera, identifies the hole, and returns its angular position, while function `wait_for_next_activation()` is an operating system call that suspends the task until the beginning of its next period.

Note that implementation A, assumes that the task is executed at the highest priority, which guarantees a constant interval between consecutive samples of exactly T_s. When the task has an intermediate priority and is executed together with other activities, the interval between consecutive sensor readings may be different than the task period, due to the interference of higher priority tasks or to blocking delays from lower priority tasks. In this more general case, the disc rotational speed has to be estimated using the actual time difference between consecutive readings:

$$\omega = \frac{\theta - \theta_{old}}{t - t_{old}}. \quad (18)$$

The corresponding task implementation is shown in Figure 5.

```
task  estimate_speed_A {
float theta;      // current position
float theta_old;  // previous position
float omega;      // estimated speed

      theta_old = read_sensor();
      wait_for_next_activation();

      while (1) {
          theta = read_sensor();
          omega = (theta − theta_old) / task_period;
          write_buffer(omega);
          theta_old = theta;
          wait_for_next_activation();
      }
}
```

Fig. 4. Implementation A of task τ_1 for estimating the disc rotation speed.

```
task  estimate_speed_B {
float theta;      // current position
float theta_old;  // previous position
float t;          // current time
float t_old;      // previous time
float omega;      // estimated speed

      theta_old = read_sensor();
      t_old = read_current_time();
      wait_for_next_activation();

      while (1) {
          theta = read_sensor();
          t = current_time();
          omega = (theta − theta_old) / (t − t_old);
          write_buffer(omega);
          theta_old = theta;
          t_old = t;
          wait_for_next_activation();
      }
}
```

Fig. 5. Implementation B of task τ_1 for estimating the disc rotation speed.

Note that implementation B provides correct estimations only if the current time is always acquired immediately after the sensor acquisition, as specified in the code. If the task is preempted between the two instructions, the acquired time is affected by a delay and the corresponding speed will be wrong. To avoid such a possibility, the sensor and the current time readings must be encapsulated within a non-preemptive region, so they are executed as an atomic action. The corresponding task implementation is shown in Figure 6.

The presence of non-preemptive regions in the task code, however, creates a potential blocking time (δ) to higher priority tasks that need to be taken into account in the feasibility analysis, as shown in Section III.

```
task  estimate_speed_C {
float t;          // current time
float t_old;      // previous time
float theta;      // current position
float theta_old;  // previous position
float omega;      // estimated speed

      disable_preemption();
      theta_old = read_sensor();
      t_old = current_time();
      enable_preemption();
      wait_for_next_activation();

      while (1) {
          disable_preemption();
          theta = read_sensor();
          t = current_time();
          enable_preemption();
          omega = (theta − theta_old) / (t − t_old);
          write_buffer(omega);
          theta_old = theta;
          t_old = t;
          wait_for_next_activation();
      }
}
```

Fig. 6. Implementation C of task τ_1 for estimating the disc rotation speed.

This also affects the bounds for the safety region, because δ has to be added to ε_t in Equation (12), so $\Delta\theta_{max}$ becomes:

$$\Delta\theta_{max} = |\omega| T_s \frac{\rho - \varepsilon_\theta}{2\varepsilon_\theta} - \rho - |\omega|\delta\frac{|\omega|T_s - 2\epsilon_\theta}{2\epsilon_\theta}. \qquad (19)$$

2) Shooting the target: Once the disc speed is correctly estimated, task τ_2 (`plan_shooting`) must compute in advance the time instant t_s at which the gun must fire to catch the target. Such a time can be computed using Equation (10), which takes into account the time needed by the target to reach the uppermost position (θ_0) from the current position (θ), and the delay needed by the bullet to cross the distance D. Note that, if the target is outside the safe interval, the time of fire cannot be precisely estimated, hence no action is performed by τ_2, which suspends itself until the next period. As soon as the target is found in the safe zone, then t_s is estimated by Equation (10) and the activation of τ_3 is posted at time t_s. When activated, task τ_3 (`fire`) just triggers the gun to fire and brings the system in the IDLE state. The pseudo code corresponding to tasks τ_2 and τ_3 is shown in Figure 7 and Figure 8, respectively.

Observe that the period T_2 of task τ_2 presents a constraint, since at least one task execution must take place inside the safe interval $[\Delta\theta_{min}, \Delta\theta_{max}]$, that is

$$T_2 + I_2 \leq \frac{\Delta\theta_{max} - \Delta\theta_{min}}{|\omega|}, \qquad (20)$$

where I_2 is the maximum start time delay due to the interfer-

```
task  plan_shooting_C {
float t;          // current time
float t_s;        // shooting time
float theta;      // current position
float omega;      // estimated speed

      while (1) {
          if (state == ARMED) then
              omega = read_buffer();
              <compute Δθ_max by Eq. (16)>;
              <compute Δθ_min by Eq. (17)>;

              disable_preemption();
              t = current_time();
              theta = read_sensor();
              <compute Δθ by Eq. (6)>;

              if (Δθ ∈ [Δθ_min, Δθ_max] ) then
                  <compute t_s by Eq. (10)>;
                  post_task(fire, t_s);
                  set_state(LOCKED);
              }
              enable_preemption();

          }
          wait_for_next_activation();
      }
}
```

Fig. 7. Pseudo code of task τ_2 for computing the shooting time.

```
task  fire {

      trigger_gun();
      set_state(IDLE);
}
```

Fig. 8. Pseudo code of task τ_3 for firing.

ence of high priority tasks (τ_1 and τ_3). Substituting equations (16) and (17) we have:

$$T_2 \leq T_s \frac{\rho - \varepsilon_\theta}{2\varepsilon_\theta} - \frac{\rho}{|\omega|} - \frac{D}{v_B} - I_2. \tag{21}$$

Note that the correct system behavior assumes that task τ_3 is precisely activated at time t_s and immediately executed by the operating system. This means that τ_3 must be the highest priority task, preempting all the other tasks in the system. This task has also significant impact on the schedulability of the other periodic tasks, which need to be analyzed by properly extending the feasibility test as shown in the next section.

III. Schedulability Analysis

This section illustrates how to verify the schedulability of such a class of target sensitive applications, both under fixed priorities and EDF.

To perform an off-line guarantee of the task set, it is necessary to treat the aperiodic task τ_3 as a sporadic task, evaluating the minimum interarrival time T_3, given the application characteristics. From the functional behavior illustrated in Section II-B, it is clear that τ_3 can be activated at most once for each turn of the disc, hence $T_3 = 2\pi/\omega_{max}$.

In addition, since τ_3 must execute as soon as it arrives, it cannot experience interference from the other tasks, thus it can be assigned a relative deadline equal to its computation time: $D_3 = C_3$. The other two periodic tasks, τ_1 and τ_2, are implemented to be tolerant to interference, hence their relative deadlines can be set equal to their periods: $D_1 = T_1$ and $D_2 = T_2$.

A. Analysis under fixed priorities

In the presence of non-preemptive regions, each task τ_i can experience an additional blocking factor B_i equal to the longest non-preemptive region belonging to lower priority tasks. If q_i denotes the length of the largest non-preemptive region in task τ_i, assuming that tasks are ordered by decreasing priorities, each blocking time B_i can be computed as

$$B_i = \max_{k>i}\{q_k\}. \tag{22}$$

Under fixed priorities, the schedulability analysis of a periodic or sporadic task set in the presence of blocking factors can be performed using the workload analysis [4], which can be restated as follows by considering non-preemptive regions [20]:

Theorem 1 (Bini and Buttazzo, 2004)**.** *Let Γ be a set of n periodic tasks in which each task τ_i may include non-preemptive regions of maximum length q_i. Then, Γ is schedulable with a fixed priority algorithm if and only if for all $\tau_i \in \Gamma$ there exists a time $t \in (0, D_i]$ such that*

$$B_i + \sum_{k=1}^{i} \left\lceil \frac{t}{T_k} \right\rceil C_k \leq t. \tag{23}$$

where

$$B_i = \max_{k>i}\{q_k\} \tag{24}$$

B. Analysis under EDF

Under EDF, the feasibility analysis can be carried out using the processor demand approach [2], considering the highest priority task τ_3 as an equivalent task scheduled by EDF having deadline equal to its computation time: $D_3 = C_3$.

Hence a task set Γ is feasible under EDF if and only if for all $L > 0$,

$$\sum_{i=1}^{n} \left\lfloor \frac{L}{T_i} \right\rfloor C_i + \left\lfloor \frac{L + P_a - C_a}{P_a} \right\rfloor C_a \leq L \tag{25}$$

In the presence of blocking terms, the Processor Demand Criterion has been extended by Baruah [3], using the concept of *Blocking Function* $B(L)$, defined as the largest amount of time for which a task with relative deadline $\leq L$ may be blocked by a task with relative deadline $> L$.

If δ_{jh} denotes the maximum length of time for which τ_j holds a resource that is also needed by τ_h, the blocking

function can be computed as follows:

$$B(L) = \max\{\delta_{j,h} \mid (D_j > L) \text{ and } (D_h \leq L)\}. \quad (26)$$

Then, a task set can be scheduled by EDF if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_a}{T_a} < 1$$

and

$$\forall L \in \mathcal{D} \quad B(L) + \sum_{i=1}^{n} \left\lfloor \frac{L}{T_i} \right\rfloor C_i + \left\lfloor \frac{L + P_a - C_a}{P_a} \right\rfloor C_a \leq L \quad (27)$$

where $\mathcal{D}$ is the set of all absolute deadlines no greater than a certain point in time, given by the minimum between the hyperperiod H and the following expression:

$$\max\left(D_{max}, \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U}\right).$$

where $D_{max} = \max\{D_1, \ldots, D_n\}$.

IV. Experimental results

This section presents some experiments performed on a real platform, developed to test the proposed approach and compare the behavior of different implementation solutions of the tasks described in Section II-B.

A. Hardware platform

The application has been developed using a small low-cost plant, in which a compact disc is rotated by a dc motor actuated by a PWM motor controller. The angular position of the target is measured by an optical encoder having a resolution of 360 clicks per rotation.

The plant is controlled using a Flex[1] board equipped with a Microchip 16-bit dsPIC microcontroller running the application tasks and the real-time operating system.

For practical reasons, the gun has been substituted with a laser pointer, whose beam points to a photoresistor located behind the disc and acquired by a 12-bit analog-to-digital converter. A digital output line is used for firing the target with the laser. When firing, the laser is turned on for 500 μs, which is the time just sufficient to activate the photoresistor. The hole (i.e., the target) has a radius of 2.5 mm and its center is located at 53 mm from the center of the disc.

In this setting, $R_B = 0$ and the shooting time can be neglected (i.e., $t_a = 0$). Moreover, the finite sensing area of the photoresistor, whose radius is R_P = 2.5 mm, has the effect of enlarging the target radius of exactly R_P. Therefore, in all the computations, the target radius is considered to be $R_H = 5$ mm.

Given the resolution of the optical encoder, the angular position error results to be

$$\varepsilon_\theta = \frac{2\pi}{360} = 17.45 \cdot 10^{-3}. \quad (28)$$

The maximum rotation speed of the motor is 860 deg/s (that is, 15 rad/s), but to stress the system beyond the theoretical limits, we considered ω_{max} = 600 deg/s (10.47 rad/s). We also imposed a minimum speed ω_{min} = 130 deg/s (2.27 rad/s) to compute a safe value of period T_2, based to Equation (21).

According to equations (17) and (16), the bounds of the safe interval result to be

$$\begin{cases} \Delta\theta_{min} = 0 \\ \Delta\theta_{max} = 2.2|\omega|T_s - \rho \end{cases}$$

where, according to Equation (13), $\rho = 94.34 \cdot 10^{-3}$ rad.

B. Software implementation

The application software has been implemented on the ERIKA Enterprise kernel [10], which is an open-source (GPL2 with linking exception) multiprocessor real-time operating system, with a programming interfaces compatible with the OSEK/VDX standard [18]. ERIKA tasks are implemented as functions and scheduled according to a scheduling policy selected by the user at compile time. Both fixed priority and EDF scheduling algorithms are supported, and the time resolution can be selected by the application developer to match the granularity required by the plant. The results reported in this section have been obtained under fixed priority scheduling, where priorities were assigned to tasks according to the Deadline Monotonic algorithm [15].

To stress the system, the sampling period T_1 of task τ_1 has been set as the maximum allowed by Equation (5), that is

$$T_1 = T_s = \frac{\pi}{\omega_{max}} = 300\ ms,$$

while the period T_2 has been selected within the bound expressed in Equation (21) ($T_2 \leq 660\ ms$). In our experiments, we set $T_2 = 300\ ms$. Considering that task τ_3 can be activated at most once for each turn of the disc, its minimum interarrival time T_3 has been set as $T_3 = 2\pi/\omega_{max} = 600\ ms$.

The application also includes a data logging task, τ_{log}, which monitors the main variables and send them to a PC through a serial line for the analysis.

Finally, to test the various implementations under different workload conditions, 4 disturbing tasks (τ_1^d, τ_2^d, τ_3^d, and τ_4^d) have been added, with a computation time equal to 4 ms and a frequency that can be proportionally increased by a factor k to modify their utilization and increase the interference on the application tasks:

$$T_i^d(k) = T_i^d/k.$$

The total utilization of the disturbing tasks is denoted by U_d and their parameters have been defined so that for $k = 10$ the system reaches a total utilization of 1.0. The timing parameters of the task set are reported in Table I.

Note that, without disturbing tasks, the schedulability of the application is guaranteed under both RM and EDF. With disturbing tasks the application is guaranteed under RM for $k \leq 6$.

C. Experiments

In a first experiment, the three implementations of task τ_1 (A, B, and C) presented in Section II-B have been compared to

[1]Flex board web site: http://www.evidence.eu.com/products/flex.html

Task	C_i (ms)	T_i (ms)	U_i
τ_1	0.028	300	$9.33 \cdot 10^{-5}$
τ_2	0.040	300	$1.33 \cdot 10^{-4}$
τ_3	0.003	600	$5.33 \cdot 10^{-6}$
τ_{log}	4.2	1000	$4.20 \cdot 10^{-3}$
τ_1^d	3.92	131	$2.99 \cdot 10^{-2}$
τ_2^d	3.92	151	$2.59 \cdot 10^{-2}$
τ_3^d	3.92	171	$2.29 \cdot 10^{-2}$
τ_4^d	3.92	191	$2.05 \cdot 10^{-2}$

TABLE I
TASK PARAMETERS USED IN THE EXPERIMENTS.

evaluate their effectiveness in catching the target as a function of the rotation speed ω, which was varied between 6.63 rad/s (380 deg/s) and 12.50 rad/s (720 deg/s). In this test, the disturbing tasks were not activated (i.e., $U_d = 0$). For each rotation speed, the measure was obtained by performing 100 shots and counting the number of times the targets was missed. The results are presented in Figure 9, where the ratio between the number of missed shots and the number of fired shots, denoted as *miss ratio*, is plotted on the y-axis.

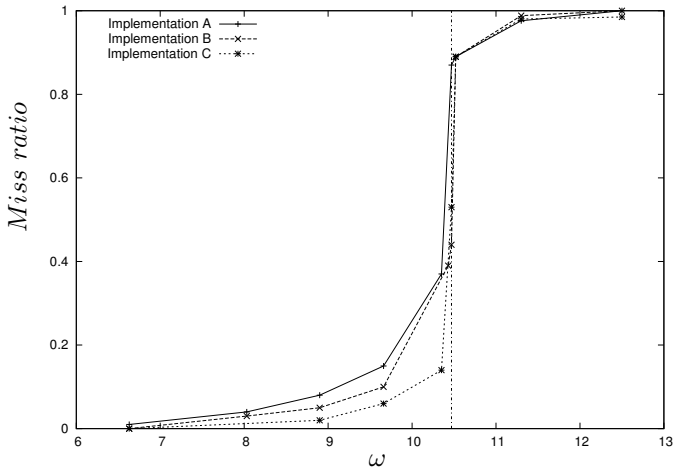

Fig. 9. Miss ratio as a function of ω for the three implementations (A, B, and C) of task τ_1.

The three implementation present a miss ratio that quickly reaches 100% as soon as the rotation speed crosses the value $\omega_{max} = 10.47$ rad/s. When the angular speed is below such a value, the differences among the three solutions are significant. Implementation A starts presenting pronounced shooting errors ($> 10\%$) for values of ω that are more that 20% smaller than ω_{max}. Implementation B presents a similar behavior, but characterized by a smaller miss ratio. Finally, Implementation C presents a negligible number of missed targets even for ω values quite near to ω_{max}.

A second experiment has been performed to evaluate the robustness of the three solutions of task τ_1 against an increasing interference. This has been done by measuring the miss ratio for different values of the disturbing load, obtained by increasing the frequencies of tasks τ_i^d's by the factor k, varied from 0 to 9. Note that the computation of t_s performed by τ_2 is not affected by the disturbing tasks, because it only requires the coherence between t and θ, which is guaranteed by the non preemptive region. In this test, the angular velocity has been fixed to $\omega = 6.63$ rad/s (380 deg/s), which is the highest value that still guarantees a negligible miss ratio for all the three implementations without disturbing workload ($U_d = 0$). As in the first experiment, for each value of the disturbing load (i.e., of k), the measure was obtained by performing 100 shots and counting the number of times the target was missed.

The results are presented in Figure 10, which shows that increasing the utilization of disturbing tasks all the implementations increase their miss ratio. However, Implementation A presents a higher number of failed shots.

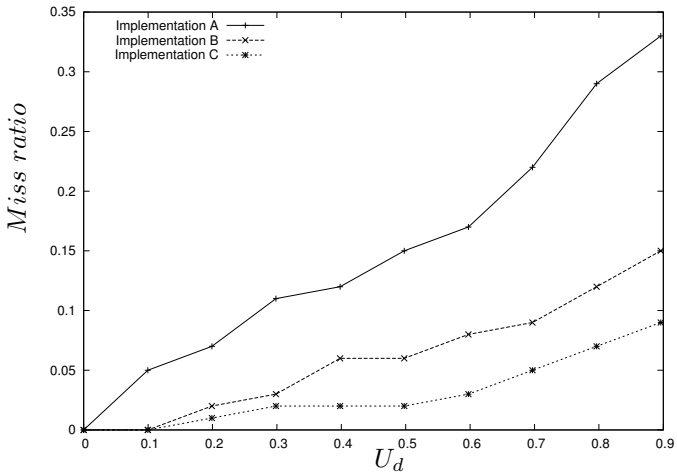

Fig. 10. Miss ratio as a function of U_d for the three implementations (A, B, and C) of task τ_1.

A third experiment was carried out to evaluate the behavior of the system around the border $\Delta\theta_{max}$ of the safe zone, that is, when the shooting time t_s is predicted when the hole is located at an angular distance $\Delta\theta$ (from θ_0) comparable with $\Delta\theta_{max}$. This has been done by measuring the miss ratio for different $\Delta\theta$ around the value $\Delta\theta_{max}$. The measurements have been performed only for implementation A, which is the one presenting a stronger performance degradation near the bound. In this test, 100 shots were fired for each value of $\Delta\theta$, and the disc rotation speed was fixed at $\omega = 5$ rad/s, leading to a value $\Delta\theta_{max} = 3.31$ rad. The results of this experiment are reported in Figure 11.

V. CONCLUSIONS

This papers discussed some implementation and analysis issues related to target-sensitive applications, in which the output of one or more tasks must be produced at precise time instants, rather than within a deadline. A specific real-time system involving a prediction task to catch a moving target has been used as a case study to show how to derive the timing constraints from the application requirements, how to implement the application on top of a real-time kernel and how to analyze the schedulability of the task set.

Three alternative implementations have been proposed and compared to evaluate the effect of the interference produced by other activities.

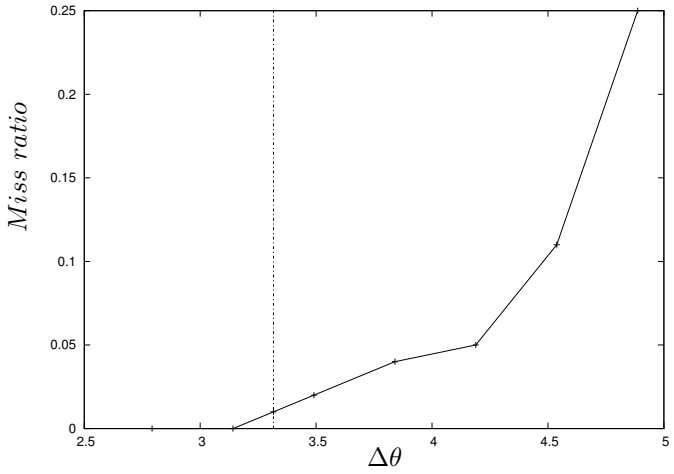

Fig. 11. Miss ratio as a function of $\Delta\theta$ for the implementation A of task τ_1.

The experimental results reported in the paper, besides confirming the effectiveness of solution C, can also be used to extract a set of general implementation guidelines that should be used to drive the design of such systems:

- Predict a future system state using a sequence of state values (state history), associating each sensor value with the corresponding acquisition time; to ensure consistency between times and state values, both readings must be performed atomically, within a non-preemptive region.
- Post the action in the future at the predicted time (target time) and trigger it as an aperiodic task executed at the highest priority.
- Handle overlapping events through a suitable shifting policy (e.g., the gravitational method [12], [11]) taking event importance into account.
- Verify the schedulability of the system off-line by taking into account the periodic task set, the aperiodic load activated at the minimum interarrival time, and the blocking intervals introduced by the non-preemptive regions.

As a future work, to verify the correctness of predictions in a more realistic setting, we plan to replace the laser pointer with a pneumatic shooting device that launches plastic spheres on the target. We also plan to evaluate the performance of the system under different scheduling algorithms (e.g., Deadline Monotonic and EDF).

We also plan to analyze the effect of other factors, such as the errors due to simplistic assumptions in the modelling phase, the effect noises in the measurements, and so on. Moreover, in this work, time predictions were computed assuming a constant angular velocity. However, the system should also be able to cope with variable speeds. In this case, predictions could be computed with a Kalman filter, as proposed for instance by Facchinetti et al. [8]. Another problem to be addressed is to find a trade off between accuracy of predictions and available processing power. In fact, good predictions require algorithms with high computational complexity that could be too demanding for a small embedded processor. On the other hand, simple predictions increase the error ε_ω on the estimated speed, thus reducing the safe bound $\Delta\theta_{max}$ and imposing stronger timing constraints on the tasks.

REFERENCES

[1] P. K. Allen, A. Timcenko, B. Yoshimi, and P. Michelman. Trajectory filtering and prediction for automated tracking and grasping of a moving object. In *Proceedings of the IEEE Int. Conference on Robotics and Automation, (ICRA '92)*, pages 1850–1856, Nice, France, May 12-14, 1992.

[2] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems*, 2, 1990.

[3] S. K. Baruah. Resource sharing in EDF-scheduled systems: a closer look. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, Rio de Janeiro, Brazil, December 5-8, 2006.

[4] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[5] E. Brandt and R. Dannenberg. Low-latency music software using off-the-shelf operating systems. In *Proceedings of the International Computer Music Conference*, pages 137–141, San Francisco, California, USA, 1998.

[6] G. C. Buttazzo, B. Allotta, and F. Fanizza. Mousebuster: a robot for catching fast objects. *IEEE Control Systems Magazine*, 14(1):49–56, February 1994.

[7] R. B. Dannenberg and D. H. Jameson. Real-time issues in computer music. In *Proceedings of the 14th Real-Time Systems Symposium (RTSS '93)*, Raleigh-Durham, North Carolina, USA, December 1-3, 1993.

[8] T. Facchinetti and G. Buttazzo. A real-time system for tracking and catching moving targets. In *Proceedings of the 5th IFAC International Symposium on Intelligent Components and Instruments for Control Applications (SICICA 2003)*, pages 251–256, Aveiro, Portugal, July 9-11, 2003.

[9] L. Farzinvash and M. Kargahi. A scheduling algorithm for execution-instant sensitive real-time systems. In *Proceedings of the 15th IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '09)*, Beijing, China, August 24-26, 2009.

[10] P. Gai, G. Lipari, L. Abeni, M. di Natale, and E. Bini. Architecture for a portable open source real-time kernel environment. In *Proceedings of the Second Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial*, November 2000.

[11] R. Guerra and G. Fohler. On-line scheduling algorithm for the gravitational task model. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS 09)*, Dublin, Ireland, July 1-3, 2009.

[12] R. Guerra and G. Fohler. A gravitational task model for target sensitive real-time applications. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS 08)*, Prague, Czech Republic, July 2-4, 2008.

[13] N. Houshangi. Control of a robotic manipulator to grasp a moving target using vision. In *Proceedings of the IEEE Int. Conference on Robotics and Automation, (ICRA '90)*, pages 604–609, Cincinnati, Ohio, USA, May 13-18, 1990.

[14] V. Iyer, J. Bilmes, M. Wright, and D. Wessel. A novel representation for rhythmic structure. In *Proceedings of the International Computer Music Conference*, pages 97–100, San Francisco, California, USA, 1997.

[15] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[16] M. Linderoth, A. Robertsson, K. Åström, and R. Johansson. Vision based tracker for dart-catching robot. In *Proceedings of the 9th IFAC International Symposium on Robot Control (SYROCO'09)*, pages 883–888, Gifu, Japan, September 9-12, 2009.

[17] H. M. W. Lunney. Time as heard in speech and music. *Nature*, 249:592, 1974.

[18] OSEK. *OSEK/VDX Operating System Specification 2.2.1*. OSEK Group, http://www.osek-vdx.org, 2003.

[19] T. Tidwell, R. Glaubius, C. D. Gill, and W. D. Smart. Optimizing expected time utility in cyber-physical systems schedulers. In *Proceedings of The 31st IEEE Real-Time Systems Symposium (RTSS 2010)*, San Diego, California, USA, November 30 - December 3, 2010.

[20] G. Yao, G. C. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *Proc. of the 15th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA'09)*, pages 351–360, Beijing, China, August 24-26, 2009.