Proceedings of the 5th IFAC International Symposium on Intelligent Components and Instruments for Control Applications (SICICA), Aveiro, Portugal, July 9-11, pp. 251-256, 2003.

A REAL-TIME SYSTEM FOR TRACKING AND CATCHING MOVING TARGETS

Tullio Facchinetti, Giorgio Buttazzo

University of Pavia, Italy
Email: tullio.facchinetti@unipv.it, buttazzo@unipv.it

Abstract: In this paper we describe the real-time issues related to the development of a control system for tracking and catching moving targets by vision. After describing the main hardware and software components of the system, we discuss a number of interesting scheduling problems that arise in these kind of control applications. A possible solution to these problems is also outlined and has been implemented on top the Shark operating system, used to develop the real-time application. The experimental results obtained on the system prototype show the effectiveness of our approach and encourage further research on the specific scheduling problems. *Copyright @ 2003 IFAC.*

Keywords: Real-time, Robot control, Tracking applications, Kalman filters, Neural networks.

1. INTRODUCTION

Most of today's real-time control applications are developed on top of priority based kernels. The most common priority assignment used to handle real-time periodic tasks is the Rate-Monotonic (RM) scheduling algorithm, which assigns priorities to tasks proportionally to their rates (Liu, 1973). RM is known to be optimal among the class of fixed priority algorithms, nevertheless, there are several problems that may arise when implementing RM on top of commercial real-time kernels. First of all, RM does not allow a full processor utilization (except for special harmonic period configurations), as its schedulability bound is approximately equal to 69%, which is a very low value, if compared with the one achieved by other scheduling techniques (like EDF, that is able to exploit the full processor power). Secondly, if the kernel does not adopt a specific concurrency control protocol for accessing shared resources, tasks may experience a priority inversion phenomenon (Sha, 1990), for which a higher priority task can be blocked by a lower priority task for an unbounded amount of time. Finally, if the kernel does not allow interrupt scheduling, periodic tasks can suffer long delays due to bursty interrupt arrivals.

When a control application requires high performance and efficient resource utilization, special real-time features need to be adopted at the operating systems level to avoid the problems outlined above. This is especially true when real-time applications need to be developed on small embedded microprocessors, with limited memory and processing power.

In the application described in this paper, there are two additional problems that need to be solved at the kernel level, to achieve a desired level of performance. The first problem is due to the presence of tasks with highly variable computation time. Guaranteeing these tasks assuming the worst-case execution time would be too pessimistic, and would cause a waste of processing power. On the other hand, considering their average-case behavior is not safe, because execution overruns may cause other tasks to miss their deadlines. The second problem is given by the presence of a particular sporadic task that needs to be executed at a specific time instant, neither before, nor later. Notice that the desired execution instant cannot be specified by a classical deadline. In fact, the classical deadline semantics does not apply to this case, where an earlier execution is also not acceptable.

The application presented here has been implemented using a research kernel, Shark (Gai, 2001), which allows to combine EDF with other scheduling techniques for avoiding priority inversion. The EDF algorithm (with shared resources) is also available in MicOS (Carlini, 2003), a real-time kernel for developing embedded applications on top of small microcontrollers (e.g., the Motorola 68HC11).

The problem of handling tasks with variable computation times has been addressed by Caccamo et al. in (Caccamo, 2002). The solution proposed by the authors consists in adopting a reservation-based approach, combined with an efficient reclaiming algorithm to exploit early completions. In particular, tasks are isolated through a Constant Bandwidth Server (Abeni, 1998), so that overruns do not interfere with the other tasks, and residual budgets due to early completion are used to handle sporadic overruns more efficiently.

The issue of accounting for interrupt handling costs in the analysis of periodic task scheduling has been addressed by Jeffay and Stone in (Jeffay, 1993). They studied the case of periodic tasks that must compete for the processor with interrupt handlers running at the highest priority, and derived a schedulability test under EDF.

An alternate solution for scheduling fixed priority tasks together with EDF is to partition the processor into two applications: one including the fixed priority tasks (running at the highest priority level), and the other including the periodic tasks scheduled by EDF at a lower priority. The issue of partitioning the processor bandwidth into different applications has been considered by Deng and Liu in (Deng, 1997), and by Lipari and Bini (Lipari, 2003).

The system described in this paper is a tracking device with two degrees of freedom, capable of orienting a laser pointer on a moving target and following its trajectory by vision. A pneumatic shooting device is used to launch a plastic sphere on the target to verify the correctness of the prediction. This system can be useful whenever an object needs to be not only tracked, but also catched in some manner, like in dynamic grasping operations. The approach can also be useful when the exact location of a target in some future time needs to be estimated from its current trajectory, as in done in astronomical applications for tracking heavenly bodies.

We report our experience in developing the robot system and present a number of interesting scheduling problems that arise from these kinds of control applications. The paper also outlines the possible solutions to these problems, that we have implemented on the Shark operating system, used to support the real-time application. In particular, Section 2 presents an overview of the robot system, Section 3 illustrates the characteristics of the real-time kernel, Section 4 illustrates the main real-time issues involved in the application, Section 5 reports some experimental results, and Section 6 states our conclusions and future work.

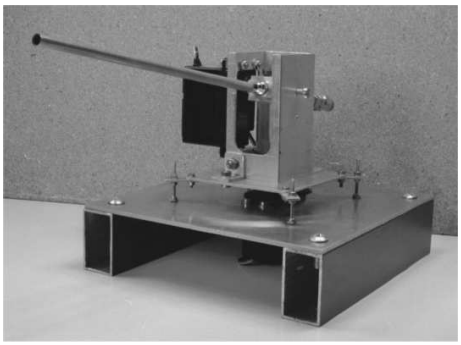

Fig. 1. The pointing device.

2. SYSTEM DESCRIPTION

2.1 General overview

The visual tracking system consists of a two-degrees-of-freedom device driven by two independent servomotors. The servo motors are controlled by a PIC-based board, connected to the PC by via a standard RS232 serial port. A 20cm blowpipe, with a laser pointer aligned with its axis, can be rotated to track a moving target to follow its trajectory. A pneumatic valve attached to the pipe is used to launch a plastic sphere on the target in order to verify the correctness of the prediction. A single fixed camera is used to monitor the target position. This solution does not allow us to evaluate the distance of the target, so we assume that the target moves in a vertical plane with known distance. A block diagram of the system is illustrated in Figure 1.

2.2 System calibration

Coordinate conversion from the image space to the actuation space is performed by a multi-layer neural network, trained using the back propagation algorithm in a calibration phase. The use of a neural network allows avoiding (i) the complexity of an optical distortion modelling of the camera lens and (ii) the three dimensional modelling of the physical environment containing both the robot and the target motion plane. The first advantage (i) makes the system independent from the lens optical parameters, so that the lens can be freely changed and the system re-calibrated by a simple network training. The second advantage (ii) regards the robot and target motion plane position and orientation: they can be changed by an off-line network re-calibration.

The neural network is trained by an automatic calibration procedure, which generates the training set by moving the motors in such a way that the laser spot is positioned in several points of the visual field. For each position of the spot, the system records the coordinates in the image plane and the corresponding values of the servomotors. Once trained, the network is able to associate a pair of motor coordinates to each

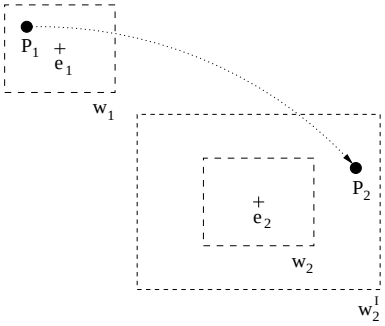

Fig. 2. Adaptive tracking window.

point in the visual field, so driving the motors in the image plane.

2.3 Visual Tracking

During tracking, image processing is performed at frame rate (50 Hz) by performing a simple threshold operation. Both the laser spot and the target are extracted from the background just on the basis of the difference in the pixel grey levels. Once an object is isolated, its position is detected by computing its center of mass. To reduce the scanning area, the moving target is searched in a small window centered in its predicted position, rather than in the entire visual field. If the target is not found in the predicted area, the search is performed in a larger region until, eventually, the entire visual field is scanned in the worst case. If the system is well designed, the target is found very quickly in the predicted area most of the times. However, in rare situations (corresponding to abrupt trajectory changes), a larger area need to be scanned, causing the task execution time to increase substantially (computation time increases quadratically as a function of the number of trials).

Clearly, attempting to guarantee the visual tracking task based on its worst-case execution time would drastically reduce its frequency, causing a severe performance degradation with respect to a soft guarantee based on the average execution time. Hence, in this application, is more convenient to perform a less pessimistic analysis and accept some sporadic overrun as a natural system behavior. However, in order to avoid side effects, the operating systems has to guarantee that overruns in a task do not interfere with the execution of the other tasks. In Shark, this can be ensured by using the Constant Bandwidth Server (Abeni, 1998), which provides a sort of temporal isolation among tasks to prevent reciprocal interference. In addition to the CBS, the CASH reclaiming algorithm (Abeni, 1998) tries to absorb overruns by exploiting the spare time saved by tasks that complete earlier than expected.

Target positions are estimated through a Kalman filter, which uses the past trajectory coordinates to provide the target position at a desired future time. In this way, the blowpipe can point at the correct direction and shoot at the correct time.

Kalman filtering theory requires a physical model of the target motion. The model we used is based on the material point kinematics equations, and generalizes up to systems with uniform acceleration, using a third order differential equations system. More accurate and reliable estimators could be obtained if a specific knowledge is available on the motion features, but they would be less general and effective in the presence of noise, or when the trajectory differs from the expected one.

The current estimator receives as input the target coordinates at each instant and computes as output the future target position at a given time, filtering the noise of the measures. Noise sensitivity can be adjusted by acting on a single parameter, ρ. Low values of ρ make the noise filter more effective, but the estimator takes more time to adapt to abrupt trajectory changes. High values of ρ cause a faster adaptation of the estimator, but the noise can be interpreted as a trajectory change, causing a higher prediction error.

Achieving a precise shot requires the evaluation of two important parameters: the valve activation delay and the sphere flight time. The valve activation is delayed by the inertia of its electro-mechanical components. This causes a delay between the shot signal triggered by the system and the actual sphere shooting. Such a delay is not constant, but slightly varies from one shot to the other, thus causing a random error on the target. Typical trajectories for the target and the sphere during a successful experiment are illustrated in Figure 3. In the figure, t_{locked} denotes the time at which the target prediction is considered reliable (i.e., when the error with the previous predictions is kept below a given value), t_{shot} denotes the time at which the sphere is shot, and t_{hit} is the time at which the target is catched. The sphere flight time depends on two parameters: the distance of the target and the air pressure involved in the shot. Since the distance can be approximately considered as a constant, we assume that the flight time only depends on the sphere speed, related to the air pressure. If the pressure is maintained constant, then the speed can be assumed constant as well. The experimental results were evaluated by a linear regression analysis, considering the air friction effect on the sphere approximately null, and a uniformly linear motion, neglecting the effect of gravity.

2.4 Real-time software

The software developed for the tracking device involves three main phases: a manual calibration, consisting in parameter tuning and scan window sizing; an automatic calibration, consisting in the training set generation and the neural network training; and the visual tracking phase. As scheme of the main phases is shown in Figure 4.

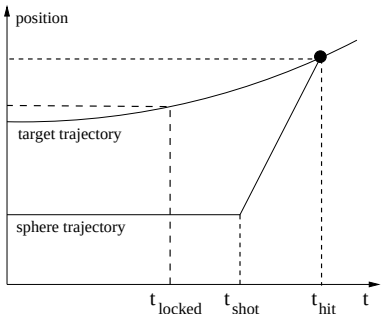

Fig. 3. Delay evaluation in the aiming problem.

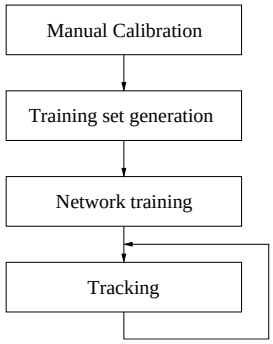

Fig. 4. Main execution phases of the system.

The application was developed in a modular fashion, in order to allow the execution of the different phases only if necessary. The calibration must be done whenever the environmental conditions change or when the robot is moved from its position. Once the calibration is performed, the tracking step can be executed for several times maintaining the same setting. Moreover, in the case of variations of certain parameters, the system can re-calibrate them only. For instance, if the brightness level suddenly changes in the target motion environment, i.e. turning the light on and off in a room, then the image processing threshold values can simply be reset independently of the other parameters.

The preliminary calibration step must begin with the procedure for setting the limits of the motors, which are essential to safely control the servos: the software driver is designed to avoid potentially damaging commands; the image processing threshold values are set in order to separate both the laser spot and the target pixels from the image background; the target size is evaluated to optimize the center of mass computation; the motor starting position is set; the Kalman filter is manually calibrated by assigning the best fitting values to its parameters, i.e. by deciding the appropriate number of integrators to use in the model.
The second step consists in the automatic training of the neural network, in order to make it able to drive the servos without the laser pointer assistance. During this stage, the training set is automatically generated and then used in the training procedure. This step is an off-line operation that does not require any real-time management.
After the system tuning, the tracking step can start. It tracks the target trajectory and estimates the proper time to catch the target.
The application consists of several calibration steps and a number of control activities for achieving the tracking behavior. The tasks and the resources involved in the tracking application are illustrated in 5. Shared resource (except for the image buffer) are accessed through the Stack Resource Policy (SRP), a concurrency control protocol proposed by Baker (Baker, 1991) for avoiding priority inversion phenomena under EDF scheduling. The image is accessed through a Asynchronous Communication Buffer (CAB), which uses state message semantics and a memory replication technique for avoiding blocking during data exchange.
The Tracker task processes the image at frame rate, feeding the Kalman filter, which, at any instant, estimates the future target positions. When the estimation is judged to be predictable enough (that is, when successive estimations match within a given error) the system computes the absolute time and position of the target and the Gunner task is posted to be executed at that precise instant. In the meanwhile, the Adjuster task continues to scan the image to check whether the prediction is correct until the shooting time. If this is not the case, the Tracker task is reactivated, otherwise the Gunner task is executed at the planned instant to open the pneumatic electro-valve.
Another task is dedicated to handle commands for the servomotors. The application includes several other tasks that, however, are not essential for catching the target. For example, a low priority task displays the output variables and manages some graphical representation on the screen, whereas a soft sporadic task handles the input from the keyboard.

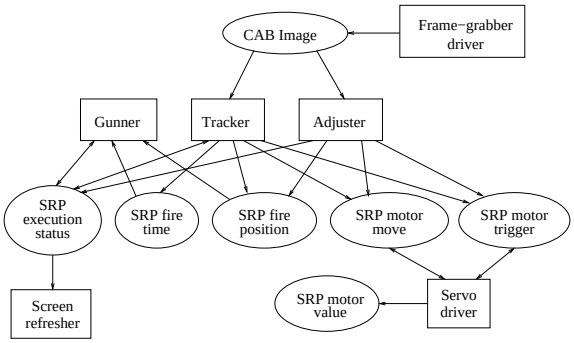

Fig. 5. Tasks and resource interaction.

3. THE SHARK REAL-TIME KERNEL

Shark is a free real-time kernel developed at the Scuola Superiore S. Anna of Pisa (Gai, 2001). The main feature that distinguishes Shark with respect to other real-time kernels is its high configurability, which allows the user to combine different schedul-

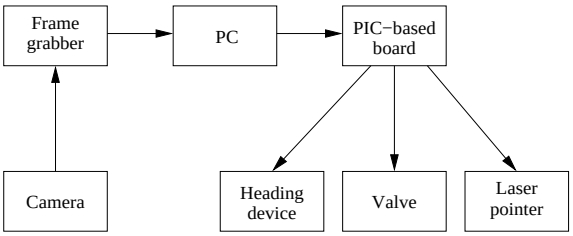

Fig. 6. Essential hardware components of the system.

ing algorithms to conform with specific application requirements. Such a flexibility also applies to resource access protocols and aperiodic service algorithms. Each policy is seen as a scheduling module that can be selected at system initialization to work in combination with the others modules according to a multi-level scheduling architecture. Several modules are already available for immediate use, however the user can easily define new scheduling modules for more specific purposes.

The kernel supports several I/O peripheral devices, including network cards, frame grabbers, and data acquisition boards. The interrupt mechanism is quite flexible and allows splitting an interrupt handler in two parts: a fast handler, which executes in the context of the running task, and a safe handler, which is scheduled by the system as an ordinary aperiodic task with given deadline. This method permits a much better balancing between predictability and responsiveness in the presence of interrupts.

The kernel provides Cyclical Asynchronous Buffers, or CABs, to exchange data among periodic tasks with different periods. In a CAB, read and write operations can be performed simultaneously without causing any blocking. Hence, a task can write a new message in a CAB while another task is reading the previous message. Mutual exclusion between reader and writer is avoided by means of memory duplication.

The main structure of the real-time application is illustrated in Figure 6.

4. REAL-TIME ISSUES

4.1 Variable computation times

The use of the adaptive window for centering the moving target significantly increases the performance of the system. With respect to a full image scan, the execution time of the tracking task in the target window can be reduced up to 95 percent in most scans, making possible to follow the object at video rate (50 Hz). However, the adaptive window approach causes the tracking task to have a highly variable computation time, that must be properly handled, in order to avoid deadline misses during long overruns.

In order to reduce the response time of the tracking task during long overruns, a reclaiming mechanism, the CASH algorithm (Caccamo, 2002), is used in combination with the CBS. Using CASH, an overrun can be partially (or possibly totally) absorbed by exploiting the spare time saved by tasks that complete earlier than expected. In fact, any spare time saved by early completions is stored in a queue with a corresponding deadline, and can be reused later, if needed by other tasks.

4.2 On-time scheduling

The correct behavior of the catching system requires the execution of a task (the Gunner) at a specific time instant, planned in advance, to open the pneumatic electro-valve. This poses a new scheduling problem that creates an additional constraint in the feasibility analysis. In fact, this activity can be modelled as a sporadic task with a long inter-arrival time (equal to the interval between two consecutive shots), but a very short relative deadline, equal to its computation time.

A possible way to guarantee the task set is by using the feasibility analysis provided by Jeffay and Stone in (Jeffay, 1993), where the Gunner task is treated as a higher priority interrupt that always preempt periodic tasks at its arrival.

In Shark, the task is handled by an RM scheduling module defined at the highest priority level, whereas all periodic tasks are scheduled by EDF at a lower level of priority.

5. EXPERIMENTAL RESULTS

Several tests have been performed on the system to verify the correct behavior of the tracking and catching algorithms, both for fixed and moving targets. The system was also tested on targets with different size and located at different (known) distances. The results showed that, under the current assumptions and system setting, catching is guaranteed for targets larger than 3 cm located at a maximum distance of 1.5 m. For smaller targets or higher distances, the errors are basically due to the limited precision of the servomotors used for positioning the device.

In the tests with a fixed target with the characteristics described above, located in any position of the visual field, the system never failed to catch the object. For the dynamic tests, the target was mounted at the end of a 80 cm stick, rotating on a vertical plane at a desired speed. The parameters are reported in Table 1.

Table 2 show the errors resulting in a typical experiment on a moving target rotating at a constant speed of 0.5 rotations per second. Numbers denote the distance of hitting point from the target center in centimeters. Figure 7 illustrates the graphical meaning of the results.

Table 1. Values of the testing parameters.

Parameter	Value
Target distance	150 cm
Speed	30 RPM
Trajectory radius	20 cm
Valve delay	15 ms
Sphere delay	0.764 ms
Hidden layer neurons	20
Kalman filter's ρ	$8.6{\cdot}10^5$
Forward estimation	12 step
Hooking radius	3 pixel
Hook number	10

Table 2. Robot testing results. Distance of hitting points from target.

Shot number	Fixed target [cm]	Moving target [cm]
1	1	2.4
2	0.2	1.5
3	1.2	1.4
4	1.4	1.8
5	0.7	2.3
6	2.1	0.7
7	1.6	0.4
8	2	1.2
9	1.3	2.5
10	1.1	2.2

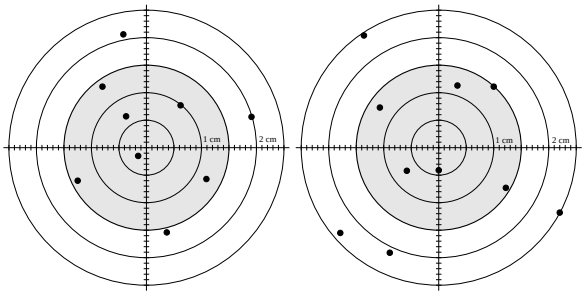

Fig. 7. Test results with a fixed target (left) and moving target (right). The darkest zone is the target area.

6. CONCLUSIONS

This paper presented a real-time visual tracking system able to follow a moving target, predict its future positions, and catch it with a plastic sphere launched through a pneumatic blowpipe. Trajectory prediction is performed by a Kalman filter that has been properly tuned to work under different realistic conditions. Servomotors are driven by a neural networks, trained to associate points in the image space to motor set points. To achieve a correct real-time system behavior, we hade to address a number of scheduling problems derived from the peculiar characteristics of the application tasks. In particular, we had to deal with a periodic task with highly variable computation time and with a sporadic task requiring to be executed at the highest priority, with respect to the other EDF tasks. The proposed solutions to these problems resulted to be very effective and were implemented in the Shark real-time kernel.

As a future work, we plan to improve the calibration phases for those procedures that now require a manual intervention, such as the threshold setting and the Kalman filter tuning. This would allow the system to self-calibrate in different working conditions. To reduce the pointing error, we plan to replace the actual servomotors with more precise dc motors with high resolution encoders and external position control loop. Finally, the use of stereo vision system would allow working with unknown target distances, and the use of a more powerful computer would allow to run more sophisticated image processing algorithms for tracking targets with different shapes.

REFERENCES

Abeni L. and G. Buttazzo (1998). Integrating Multimedia Applications in Hard Real-Time Systems. *Proc. of the IEEE Real-Time Systems Symposium*, Madrid, Spain.

Baker T.P. (1991). Stack-Based Scheduling of Real-Time Processes. *The Journal of Real-Time Systems* 3(1), pp. 67–100.

Caccamo, M., G.C. Buttazzo, L. Sha (2002). Handling Execution Overruns in Hard Real-Time Control Systems. *IEEE Transactions on Computers*, Vol. 51, No. 7, pp. 835-849.

Carlini A. and G. Buttazzo (2003). An Efficient Time Representation for Real-Time Embedded Systems. *Proc. of the 18th ACM Symposium on Applied Computing*, Track on Embedded Systems: Applications, Solutions, and Techniques, Melbourne, Florida, USA.

Deng Z. and J. W. S. Liu (1997). Scheduling Real-Time Applications in an Open Environment. *Proceedings of the IEEE Real-Time Systems Symposium*, San Francisco.

Gai, P., L. Abeni, M. Giorgi and G. Buttazzo (2001). A new kernel approach for modular real-time system development. *Proc. 13th IEEE Euromicro Conf. on Real-Time System.*, Delft, Netherland.

Jeffay K. and D. L. Stone. Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 212-221.

Lipari G. and E. Bini (2003). Resource Partitioning among Real-Time Applications. *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal.

Liu, C.L. and Layland, J.W. (1973). Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM* 20(1), pp. 40–61.

Sha, L., L.R. Rajkumar and J.P. Lehoczky (1990). Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers* 39(9).