Adaptive Control in Multi-Task Mobile Robotic Applications

Daniel Hernández Sosa
IUSIANI (ULPGC),
Edificio del Parque Tecnológico,
Campus de Tafira,
Las Palmas G.C.
(dhernandez@iusiani.ulpgc.es)

Jorge Cabrera Gámez
(jcabrera@dis.ulpgc.es)

Antonio Carlos Domínguez Brito
(acdbrito@dis.ulpgc.es)

Cayetano Guerra Artal
(cguerra@iusiani.ulpgc.es)

March 9, 2004

Abstract

Mobile robotic systems are often conditioned by limitations in available resources. Normally these systems try to offer a robust behaviour while pursuing several goals simultaneously, which leads to the execution of several tasks competing for CPU time and other shared hardware resources as sensors and actuators. If not properly managed, these limitations may provoke poor performance and, even worse, blocking of the entire system operation. In this work we propose a series of control policies for dynamic adaptation that have been integrated on a modular architecture. We also present two demonstrators to evaluate working implementations on real world applications.

Introduction

Mobile robotic systems are often affected by shortage of resources. This problems is aggravated by the configuration of these systems in tactical multi-objective designs that require the robust execution of multiple tasks in parallel [10]. In such an environment, with multiple - normally periodic - components executing concurrently, interactions among them can lead to low performance situations [3] [9]. If the problem receives no attention, the whole system reactivity/security may be threatened.

In a hard real-time system a worst-case off-line analysis can guarantee a correct execution. However, for soft real-time systems this is not a good solution, as available resources often become wasted. As a consequence, alternative on-line dynamic control approximations try to modify the system behavior during execution time. Some adaptive strategies that have been applied in this area include anytime algorithms [4], imprecise computation [11], design-to-time techniques [5] or deliberative scheduling [1].

Our proposal offers a set of integrated resources and policies aimed at obtaining run-time system adaptability, including a graceful degradation when there are not enough available resources and a maximum performance status recovery whenever possible. Additional objectives are reactivity, stability and coordination to avoid system imbalances.

This paper is organized as follows: first, the system architecture is presented, including both structural and functional organization. Then the adaptation capabilities proposed are described and, finally, the experiments and the conclusions.

1 The System Architecture

Integrated design plays a very important role in our system. The complexity of software development for robotic system has been addressed by means of the use of special purpose languages and architectures [6] [2]. Often, adaptation problems are considered after the system architecture has been completely defined. We consider that this approximation is not the most convenient, as it can reduce the flexibility and compromise the performance of the resulting control solutions.

1.1 Structural organization

The system architecture builds up from the interconnection of general purpose BU-TD-COM modules (see figure 1). Each module contains three interconnected units: Bottom-Up or BU, Top-Down or TD, and Communication or COM. The BU unit transforms the received data by means of processing algorithms, generating results to be distributed to consumer modules. The TD unit supervises the module operation, generating control orders to modify its behavior. The COM unit is in charge of data distribution from producer modules to consumers.

The main features of this structural organization include a clear functional separation between processing (BU), control (TD) and communication (COM),

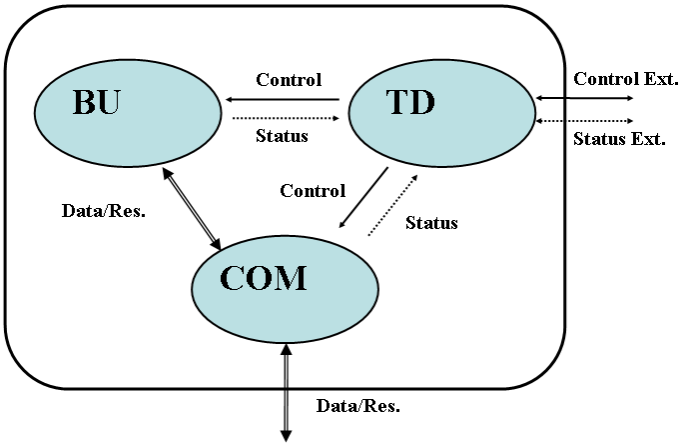

Figure 1: Generic BU-TD-COM module

a high level of autonomy, and a system-wide homogeneous interface among modules.

1.2 Functional organization

From a functional point of view, the system defines five types of functional modules. These basic objects are implemented by the BU-TD-COM modules.

- Sensors.
- Diagnostics.
- Actions.
- Actuators.
- Supervisors.

These basic objects are combined in tasks which, in turn, are organized in states to complete high-level functionality. Among basic system objects, Sensors provide source data for other modules abstracting physical sensors. Diagnostics objects encapsulate algorithms for data transformation, taking inputs from either sensors or other diagnostics. Action objects are in charge of the control of the execution, analyzing the results from diagnostics to generate control commands. Actuators abstract physical effectors, translating commands into real actions. Finally, Supervisors play the role of high level controllers in the system.

All system modules follow internally the same state graph (see figure 2). A module transits from IDLE state to READY state when assigned to a specific functional module. The module remains in this state until commanded for execution, transiting then to RUNNING state. From this point, the module can finish correctly its execution returning to READY state, interrupt temporally

the execution on high level command entering SUSPENDED state, or reach ERROR state after the detection of an running error condition. This common structure allows for a clear code organization, simplifying implementation.

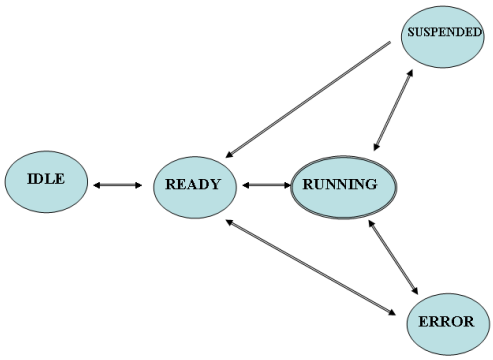

Figure 2: Module state graph

The tasks combine functional modules to form closed perception-action loops. A typical configuration is the connection of a sensor module to a diagnostics module for processing raw sensor data. The results are delivered to an action module that analyzes them to generate control commands directed to an actuator module. This module is attached to a physical effector to complete the control loop.

The states represent different groups of tasks that must be executed simultaneously. A set of transitions control the evolution of the system through the defined states. A more detailed description of the functional objects can be found in [8].

In this work we will concentrate on periodic tasks, that is, repetitive processing loops that must verify a frequency of operation given by the system designer.

2 Adaptive Control

The dynamic adaptation of the tasks inside the system is designed around two controllable variables: frequency of operation and quality level (see figure 3). On the frequency axis, a supervisor can modify the period associated to any of the tasks under its control, for example, increasing their values to face CPU saturation. On the quality axis, the supervisors can command lower qualities (sensor resolution, accuracy of computations, exhaustiveness, etc.) to reduce CPU load and latencies at the cost of increasing uncertainty or decreasing results quality. Due to the reduction of performance associated to frequency or quality degradation, the system always tries to restore the nominal parameter values as soon as resource limitations disappear.

The system uses several monitoring variables to detect when adaptive control actions are needed. These signals include both internal or direct processing results, and external stimuli. Some examples are the following:

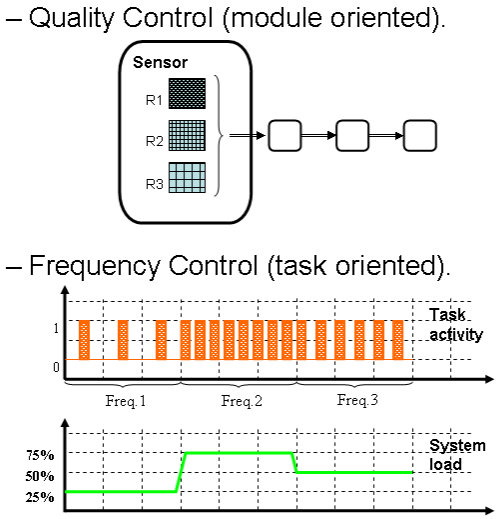

Figure 3: Quality and frequency control

- Internal.
  - Obstacle density in navigation applications.
  - Target speed in tracking applications.
- External.
  - Timeouts.
  - System load.
  - Battery level.

Each signal is monitored by a control loop that compares its value with reference levels. Control aspects are associated naturally to TD units inside each module, at low level, and supervisors, at high level. The early separation of control, processing and communication areas facilitates the implementation, promoting and preserving modularity.

Several control policies have been designed to organize system adaptation. Their objectives include:

- Avoid an unbalanced system degradation/promotion.
- Reduce settling times.
- Stability.

The control sequence begins with the activation of one or more control loops detecting out-of-range variables. Whenever possible, control actions are triggered hierarchically, local actions or task-scope first, state-scope later and finally, if problems persist system scope actions are issued. The TD units are in charge of local scope, while supervisors operate at state and system levels.

2.1 Computational adaptation

We will describe now in more detail two adaptive control strategies for controlling timeouts and system load.

2.1.1 Timeout control

Timeouts control adapts, on a hierarchical basis, the computational demands of the system in order to guarantee the specified frequencies of operation. Firstly, period violations are detected locally inside the time-pressured task. The task supervisor selects a candidate module for degradation and sends the corresponding order. To avoid systems unbalance, however, local control actions are limited to a scope defined by two homogeneity thresholds. If local adaptation resources are not enough, the supervisor notifies the problem to upper state level. If state supervisor also reaches its limit, the timeout situation is raised to system level supervisor, where global actions can be executed.

2.1.2 CPU load control

The load control loop operates only at global level. The system load is estimated and compared with a certain reference level fixed externally. Promotion and degradation actions are generated accordingly to maintain the desired load level.

Candidate selection for targeting control actions plays an important role on adaptation performance. In general, an agreement between reactivity and stability must be reached. The most intense reactions are obtained when one or more of the following conditions are met:

- High frequency tasks.
- CPU demanding modules.
- Multiple destination modules.
- High-resolution sensor modules.

The supervisors evaluate these parameters to select target tasks for adaptation, either in local state scope or in system global scope. The TD units, in turn, evaluate the parameters to select target modules inside the task (see [8] for a more complete description).

3 Experiments

We have implemented two demonstrators to illustrate the operation of the adaptation mechanisms on real-world applications: a visual tracking system and a mobile robotic application.

3.1 Tracking

The first application consists in a correlation-based tracking system for a robotic head. A USB web-cam (3Com HomeConnect) has been mounted on a pan-tilt capable neck (DirectedPerception PTU) controlled via serial port. The figure 4 shows the hardware configuration used for this demonstrator.

Figure 4: Robotic head

The goal of the application is to detect first, and keep centered on the image later, a certain pattern. A correlation-based measure [7] is used to localize the target on the image. The application is organized in three states: "Tracking", "Active Searching" and "Passive Searching". When the pattern is localized the system executes in the tracking state trying to keep the target centered on the image. If the target is lost, the system transits to the active searching state, where a scanning of the area in front of the head is performed. On target recovery the system returns to the tracking state. Otherwise the execution transits to the passive searching state, where the system remains static until the target is detected again. Several secondary tasks have been added to some states to perform additional calculations (image feature extraction). Figure 5 shows the states, transitions and tasks configuring this application.

Within this system many alternatives for computational adaptation are possible. For the main correlation task:

- Multiple resolution sensor.
- Correlation pattern size.
- Searching area

Figure 6 illustrates how quality level commands directed to the correlation task affect global system load. In this case, it is the sensor resolution that has been modified.

For the secondary tasks, variable resolution and frequency of operation allow for the modification of computational demands. In case of saturation the adaptation policies try to degrade first secondary tasks and later correlation task. When recovering, the higher priority correlation task promotes first and secondary tasks later.

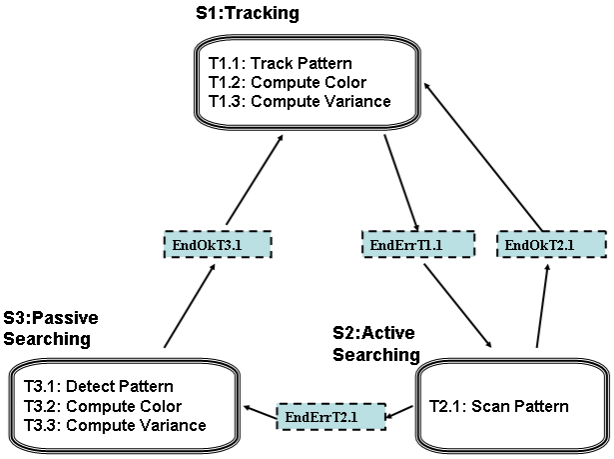

Figure 5: Tracking tasks and states

Figure 7 illustrates the variations on system load as the execution evolves through the application states.

3.2 Mobile robot

The second demonstrator mounts the mechanical head described on the previous application on a mobile robot (Pioneer). A notebook has been added for running the application, being connected via USB and serial ports to the head and the robot. Figure 8 shows the resulting hardware platform.

A tactic multi-purpose application has been designed combining two main objectives: line following and object detection. The robot must follow, as tight as possible, a trajectory defined by a line traced on the floor. At the same time, the robot must look at both sides of the route trying to detect some colored balls. The first task is considered to have a higher priority than the second one, so the adaptation strategy operates modifying the frequency of execution and quality level of the object detection task.

On straight-line segments both tasks can be performed alternatively at a predefined frequency. On curved segments, however, the risk of separation from or loosing the track increases. To avoid this, the movement amplitude and activation period of the object detection task is modulated according to the curvature of the line that the robot must follow. The modification of the scanning amplitude can be considered a quality-based adaptive control, as processing times are shortened at the cost of reducing the probability of finding color objects. The modification of the period, however, corresponds to a frequency-based adaptive control.

The figure 9 represents the executions of the secondary task along the trajectory. On curved segments, both frequency and amplitude of scanning take lower values. On straight segments both parameters are increased.

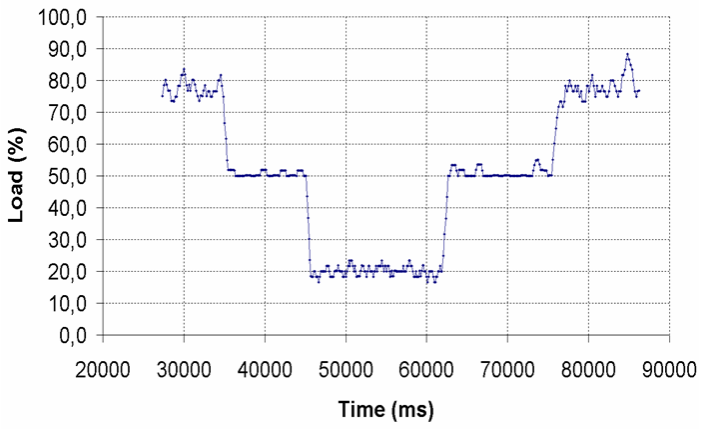

Figure 6: Quality control on correlation

4 Conclusions

The rational use of shared resources becomes a crucial factor in robotic systems. We have presented here a set of mechanisms to endow adaptation control to robotic applications using a generic module based architecture. The system degrades its performance level to accommodate resource shortage, and recovers whenever possible. The common internal concept benefits system designers in terms of modular code organization and facilitates the programming of control policies.

The adaptive control resources have been considered from the beginning the system design to avoid architecture-dependant limitations. This integrated development of architecture and adaptation mechanisms becomes a factor of main importance in the success of the control scheme.

Two demonstrators illustrate the effectiveness of the proposed mechanisms on real-world applications.

References

[1] M. Boddy and T. Dean, *Decision-theoretic deliberation scheduling for problem solving in time-constrained environments*, Artificial Intelligence **67** (1994), no. 2, 245–286.

[2] Eve Coste-Manière and Reid Simmons, *Architecture, the backbone of robotic systems*, Proceedings of the IEEE International Conference on Robotics & Automation (San Francisco, CA, USA), Abril 2000, pp. 67–72.

[3] B. D'Ambrosio, *Resource bounded-agents in an uncertain world*, Proceedings of the Workshop on Real-Time Artificial Intelligence Problems (Detroit, MI, USA), Aug. 1989.

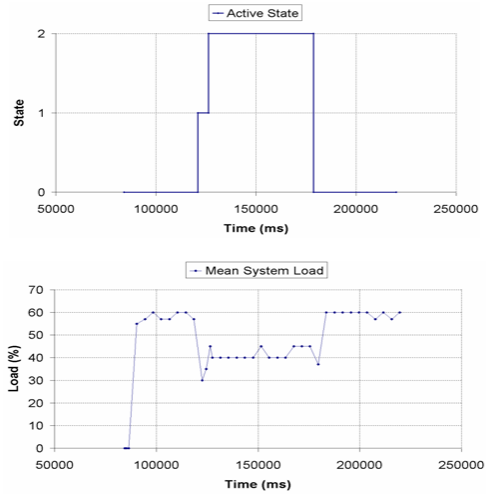

Figure 7: Load and states

[4] T. Dean and M. Boddy, *An analysis of time-dependent planning*, Proceedings of the 7th National Conference on Artificial Intelligence, AAAI (St. Paul, MN, USA), 1988, pp. 49–54.

[5] Alan J. Garvey and Victor Lesser, *Design-to-time real-time scheduling*, IEEE Transactions on Systems, Man and Cybernetics **23** (1993), no. 6, 1491–1502.

[6] E. Gat, *On three-layer architectures*, Artificial Intelligence and Mobile Robots. MIT/AAAI (1997), 195–210.

[7] Cayetano Guerra-Artal, *Contribuciones al seguimiento visual precategórico*, Ph.D. thesis, Universidad de Las Palmas de Gran Canaria, 2002.

[8] D. Hernández-Sosa, *Adaptación computacional en sistemas percepto-efectores. Propuesta de arquitectura y políticas de control*, Ph.D. thesis, Universidad de Las Palmas de Gran Canaria, 2003.

[9] E. J. Horvitz, G. F. Cooper, and D. E. Heckerman, *Reflection and action under scarce resources: Theoretical principles and empirical study*, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89) (Detroit, MI, USA), 1989, pp. 1121–1127.

[10] Stephen D. Jones, *Robust task achievement*, Ph.D. thesis, Institut National Polytechnique de Grenoble, 1997.

[11] J. Liu, K. Lin, R. Bettati, D. Hull, and A. Yu, *Use of imprecise computation to enhance dependability of real-time systems*, pp. 157–182, Kluwer Academic Publishers, 1994.

Figure 8: Mobile robot

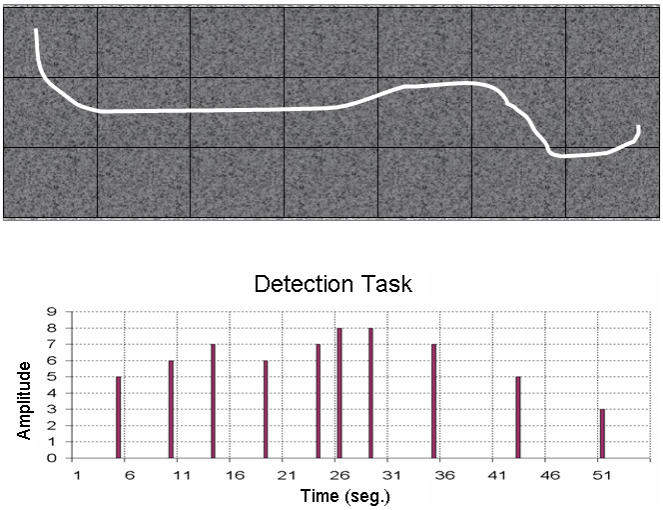

Figure 9: Secondary task execution