Autonomic Computer Vision Systems

James L. Crowley[1], Daniela Hall[2], Remi Emonet[1]

[1] INP Grenoble, Project-Group PRIMA, INRIA Rhône-Alpes, France
[2]Société ActiCM, Moirans, France
{James L. Crowley, Remi Emonet}@inrialpes.fr, Daniela.Hall@free.fr

Abstract. For most real applications of computer vision, variations in operating conditions result in poor reliability. As a result, real world applications tend to require lengthy set-up and frequent intervention by qualified specialists. In this paper we describe how autonomic computing can be used to reduce the cost of installation and enhance reliability for practical computer vision systems. We begin by reviewing the origins of autonomic computing. We then describe the design of a tracking-based software component for computer vision. We use a software component model to describe techniques for regulation of internal parameters, error detection and recovery, self-configuration and self-repair for vision systems.

Keywords: Robust Computer Vision Systems, Autonomic Computing, Perceptual Components, Layered Software Architecture

1 Towards Robust Computer Vision

Machine perception is notoriously unreliable. Even in controlled laboratory conditions, programs for computer vision generally require supervision by the programmer or another highly trained engineer. To meet constraints on reliability, vision system developers commonly design systems with parameters that can be adapted manually. In the best case, such parameters are controllable from an on-line interactive interface. All too often such parameters are simply embedded within the source code as magic numbers. Such systems perform well in controlled or static environments, but can require careful set up and "tuning" by experts when installed in new operating conditions. The need for installation and frequent maintenance by highly trained experts can severely limit the market size for such systems. Robust operation in changing, real world environments requires fundamental progress in the way computer vision systems are designed and deployed.

We believe that autonomic computing offers a theoretical foundation for practical computer vision systems. Furthermore, it appears that machine perception is an ideal domain for the study of autonomic computing techniques, because of requirements for robust real time response under different operating conditions, and the availability of feedback on quality of processing. In this paper, we describe how autonomic computing can be used to build computer vision systems.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

1.1 Autonomic Computing

Autonomic computing has emerged as an effort that takes inspiration from biological systems to render computing systems robust [1]. According to Klephard and Chess, [2] the term "Autonomic Computing" has been introduced by IBM vice president Paul Horn, in a keynote address to the National Academy of Engineers at Harvard University in March 2001. Horn presented autonomic computing systems as systems that can manage themselves given high-level objectives from administrators. The term autonomic computing was adapted as a metaphor inspired by natural self-governing systems, and in particular, from the autonomic nervous system found in mammals.

The autonomic nervous system (ANS) is a part of the nervous system that is not consciously controlled and serves to regulate the homeostasis of organs and physiological functions. The ANS is commonly divided into three subsystems: the sympathetic nervous systems (SNS), parasympathetic nervous system (PNS) and enteric nervous systems (ENS). The sympathetic nervous system acts primarily on the cardiovascular system and activates the sympatho-adrenal response of the body, also known as the fight or flight response. The parasympathetic system (PNS), also known as the "rest and digest" system, complements the sympathetic system by slowing the heart, and returning blood circulation in the lungs, muscles and gastro-intestinal system to normal conditions after reactions by the sympathetic nervous system. The enteric nervous system (ENS) autonomously regulates digestion, and is increasingly referred to as a "second brain".

Inspiration from biological models has lead to efforts to design computing systems with a number of autonomic properties. These include:

- Self-monitoring: The ability of a component or system to observe its internal state, including such quality-of-service metrics as reliability, precision, rapidity, or throughput.
- Self-regulation: The ability of a component or system to regulate its internal parameters so as to assure a quality-of-service metric such as reliability, precision, rapidity, or throughput.
- Self-repair: The ability of a component or system to reconfigure itself so as to respond to changes in the operating environment or external requirements [3].
- Self-description: The ability of a component or system to provide a description of its internal state.

Designing systems that have these properties can result in additional computing overhead, but can also return benefits in system reliability and usability.

In order to fully exploit an autonomic approach, computer vision systems must be embedded as part of a larger system for user services designed according to autonomic principles. Most important is the ability to automatically launch, configure and reconfigure components and adjust parameters for component systems. Modern tools for software ontologies can be used to provide a component registry to record available sensors and components according to their capabilities, and to provide diagnostic and configuration methods to the larger system. An ontology server can provide a registry for the data types and events consumed and produced by components, thus making it possible for a system to detect and repair errors, either by modifying data or by reconfiguring the interconnection of modules or components.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

With such an approach, systems can respond to failure or degradation by shutting down the failed component and launching an alternate.

A number of authors have experimented with such techniques for constructing robust computer vision systems. Murino [4] addresses the problem of automatic parameter regulation for vision systems within a multi-layered component architecture. Each layer has its own set of parameters that are tuned such that the evidence (coming from the lower level) and the expectation (coming from the higher level) are consistent. Scotti [5] proposes an approach based on self-organizing maps (SOM). A SOM is learned by registering good parameter settings. During runtime, the automatic parameter selection chooses the closest setting in SOM space that performed best during training.

In [6], Min proposes an approach for comparing the performance of different segmentation algorithms by searching the optimal parameters for each algorithm. He proposes an interesting multi-loci hill-climbing scheme on a coarsely sampled parameter space. The segmentation system performance is evaluated with respect to a given ground truth. This approach is designed for the comparison of algorithms and requires testing a large number of different parameter settings. For this reason, the utility of this approach for on-line parameter regulation is less appropriate.

Robertson and Brady [7] propose an architecture for self-adaptive systems. They consider an image analysis system as a closed-loop control system that integrates knowledge in order to be self-evaluating. Measuring and comparing the system output to the desired output and applying a corrective force to the system leads to increased performance. The difficult point is to generate a model of the desired output. They demonstrate their approach on the segmentation of aerial images using a bank of different filter operators. The system selects automatically the best filter for the current image conditions.

The system and approach described below have been worked out in a series of European projects over the last 6 years. Experimental validation for techniques for parameter initialization can be found in Hall et a. [8]. Techniques and validation for parameter regulation are reported in [9]. In this paper we build on and extend this earlier work.

1.2 Some Terminology

Unfortunately, the vocabulary for component-oriented software engineering and autonomic systems are still emerging, and the literature is replete with inconsistent uses. Thus it is important for us to clarify some of the terms used in this paper.

Auto vs. Self: In the software engineering literature, one can often find an almost interchangeable use of the terms "auto" and "self". The situation is simpler in French, where only the term "auto" exists, and the scientific culture seems to shy from anthropomorphic notions. In this paper, we have adopted the following usage: "Self" is used to refer to autonomic abilities that are provided using an explicit (declarative or symbolic) description of a system. Thus a component can be said to be "self-descriptive" when it contains a declarative description of its function and principles of operation, or of its internal components and their interconnections. The term "Auto" will be used for techniques that do not employ an explicit declarative description, but

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

simply on a measured property. Thus an auto-regulatory system may be implemented as a hardwired feedback process that regulates parameters without explicit description of their meaning or use while a self-repairing system will use a declarative model to reconfigure its internal function.

Homeostasis and Autonomic Control: Homeostasis or "autonomic regulation of internal state" is a fundamental property for robust operation in an uncontrolled environment. A component is auto-regulated when processing is monitored and controlled so as to maintain a certain quality of service. For example, processing time and precision are two important state variables for a tracking process. These two may be traded off against each other. The process supervisor maintains homeostasis by adapting module parameters using the auto-critical reports.

Modules, Components and Services: Process architectures have been explored for computer vision systems since the 1990's [10]. Such architectures are a form of data flow models for software architectures [11]. We apply a component model [12] to the definition of software at three distinct layers, as shown in figure 1. These three layers are the service layer, the component layer and module layer. Programming style tends to vary for these three layers. A software agent architecture, using tools such as the JAVA JADE environment is often appropriate for the service layer. The software components are generally programmed as autonomous computer programs under the control of a component supervisor. Modules can be programmed as a class with methods using an object-oriented style or as a procedure or subroutine in a more classical procedural language. Each layer provides an appropriate set of communication protocols and configuration primitives, as well as interface protocols between layers.

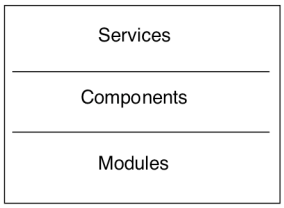

Fig. 1. Three Layers in a component-based software architecture

Modules: Modules may be formally defined as synchronous transformations applied to a certain class of data or events, as illustrated in Figure 2. Modules generally have no state. They are executed by a call to a method (or a function or a subroutine depending on the programming language) accompanied by a vector of parameters. The parameters specify the data to be processed, and describes how the transform is to be applied. The output from a module is generally output to a serial stream or posted as an events to an event dispatcher.

A module may be considered as auto-descriptive when it returns a report that describes the results of processing. Examples of information contained in such a report may include elapsed execution time, confidence in the result, or any exceptions that were encountered.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

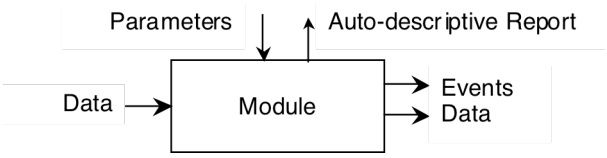

Fig. 2. Modules apply a transformation to data and return a report

An example of such an auto-descriptive module is a module that transforms RGB color pixels within a region of interest of an image into a scalar values that represents the probability that the pixel belongs to a target. Such a transformation can defined as a lookup table representing a ratio of color histograms [13]. The command to run the module is accompanied by a parameter vector that includes a pointer to the input image buffer, a pointer for an output image buffer, a pointer to a ROI (Region of interest) data structure, a pointer to the lookup table, and a step size at which the transform is to be applied within the ROI. Computing time for this process may be reduced by restricting processing to one pixel out of S (S represents a "step size") [14]. Most computer vision algorithms can be implemented as an assembly of such modules.

Perceptual Components: Modules may be assembled into software components. A component is an autonomous assembly of modules executed in a cyclic manner controlled by a supervisor as shown in figure 3. The component supervisor interprets commands and parameters, supervises the execution of the components, and responds to queries from the components with a description of the current state and capabilities of the component. The auto-critical report from modules allows the supervisor to adapt the execution schedule for the next cycle so as to maintain a target for quality of service, such as execution time or number of targets tracked. In our group, we embed C++ modules in a modified "scheme" environment [15] to implement supervised perceptual components.

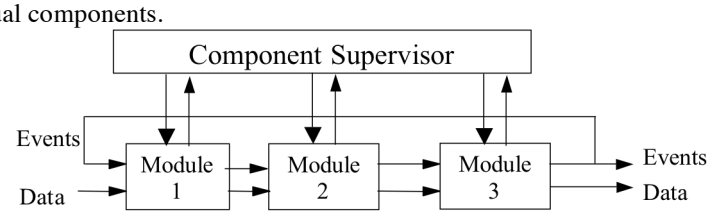

Fig. 3. A perceptual component composed of a set of modules under the cyclic control of a component supervisor.

User services can be designed as software agents that observe human activity and respond to events. An example of such a service would be an activity logging service in an office environment, that maintains a journal of classes of activities such as talking on the phone, typing at a computer or meeting with visitors. In the CHIL project [16] we have implemented a number of such services based on dynamic assembly of perceptual components. Such services can be driven by a situation model [17] that integrates information from different perceptual components. In this paper we will concentrate on the software component design model for computer vision to provide such services.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

2. Autonomic Perceptual Components

As described above, perceptual components are software components that observe a scene or a recorded data stream in order to measure properties, to detect and recognize entities or activities, or to detect events. Perceptual components based on tracking constitute an important class of components with many applications. Such components integrate information over time, typically through calculations based on statistical estimation. Tracking components may be designed for nearly any detection or recognition method, and provide important benefits in terms of robustness of detection and focus of processing resources. In this section, we propose a general software model for tracking-based perceptual components.

2. 1. Tracking-Based Perceptuel Components

Tracking systems constitute an important class of perceptual components. The architecture for a typical tracking-based perceptual component is shown in figure 4. Tracking is a cyclic process of recursive estimation, classically defined as a Bayesian estimation process composed of three phases: Predict, Detect, and Update. A well-known framework for such estimation is the Kalman filter [18]. The prediction phase uses the previously estimated attributes for observed targets (or entities) to predict current values in order to control observation. The detection phase applies the prediction to the current data to locate and observe the current values for properties, as well as to detect new targets. The update phase tests the results of detection to eliminate erroneous or irrelevant detections (distracters), recalculate the latest estimates for parameters for targets, and amend the list of observed targets to account for new and lost targets.

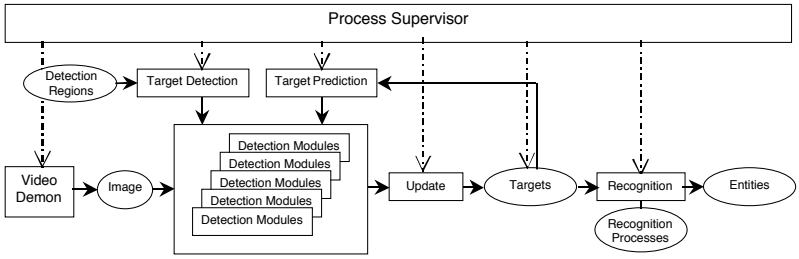

Fig. 4. A tracking-based perceptual component.

We add a recognition phase, an auto-regulation phase, and a communication phase to the classical tracking phases in our tracking-based perceptual component. In the recognition phase, recognition algorithms are applied to the current list of targets to verify or determine labels for targets, and to recognize events or activities. The auto-regulation phase determines the quality of a service metric, such as total cycle time or confidence and adapts the list of targets as well as the target parameters to maintain a desired quality. During the communication phase, the supervisor responds to requests

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

from other components. These requests may ask for descriptions of the component state or capabilities, or may provide specification of new recognition methods.

2.2 Component Supervisor

The component supervisor acts as a scheduler, invoking execution of modules in a synchronous manner. The execution report returned by each module allows the supervisor to monitor performance and to adjust module parameters for the next cycle. The results of processing are fed to a statistical classifier to detect incorrect operation. When such errors are detected a second classifier can be used to select a procedure to correct the error, as described below. The supervisor is able to respond to external queries with a description of the current state and capabilities. We formalize these abilities as the autonomic properties of self-monitoring, auto-regulation, self-repair, and self-description.

A self-monitoring process maintains a model of its own behavior in order to estimate the confidence for its outputs. Self-monitoring allows a process to detect and adapt to changing observational circumstances by reconfiguring its component modules and operating parameters. Techniques for acquiring an operating model, for monitoring operating conditions, and for repairing operational problems are described in section 3.

A process is auto-regulated when processing is monitored and controlled so as to maintain a certain quality of service. For example, processing time and precision are two important state variables for a tracking process. These two parameters may be traded off against each other. The process controllers may be instructed to give priority to either the processing rate or precision. The choice of priority is determined by the federation tool or by the federation supervisor. Techniques for monitoring output quality and for regulating internal parameters are also described in section 3.

Each target and each detection region contains a specification for the module to be applied, the region over which to apply the module, and the step size to apply processing. Recognition methods are loaded as snippets of code that can generate events or write data to streams. These methods may be downloaded to a component as part of the configuration process to give a tracking process a specific functionality.

Quality of service metrics such as cycle time and number of targets can be maintained by dropping targets based on a priority assignment or by reducing resolution for processing of some targets (for example based on size).

A self-descriptive controller can provide a symbolic description of its parameters, data structures, functions and current internal state. Such descriptions are useful for both manual and automatic composition of federations of components. During initialization, components may publish a description of their basic functionality and data types in an ontology server. During execution, components can respond to requests for information about current state, with information such as number and confidence of currently observed targets, current response time, or other quality of service information.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

2.3 Assembling Components to Provide Services

A user service is created by assembling a collection of software components. Available components may be discovered by interrogating the ontology server. An open research challenge is to provide an ontological system for indexing components based on function in a manner that is sufficiently general to capture future functionalities as they emerge. In addition the ontology server is used to establish compatible communications of data. The problem of aligning ontologies of data structures is manageable when components are co-designed by a single group. The problem becomes very difficult when components are developed independently with no prior effort to agree on specifications. This problem resembles to web-ontology alignment problem that currently receives attention in software engineering.

We have constructed a middle-ware environment [19] that allows us to dynamically launch and connect components on different machines. This environment, called O3MiSCID, provides an XML based interface that allows components to declare input command messages, output data structures, as well as current operational state.

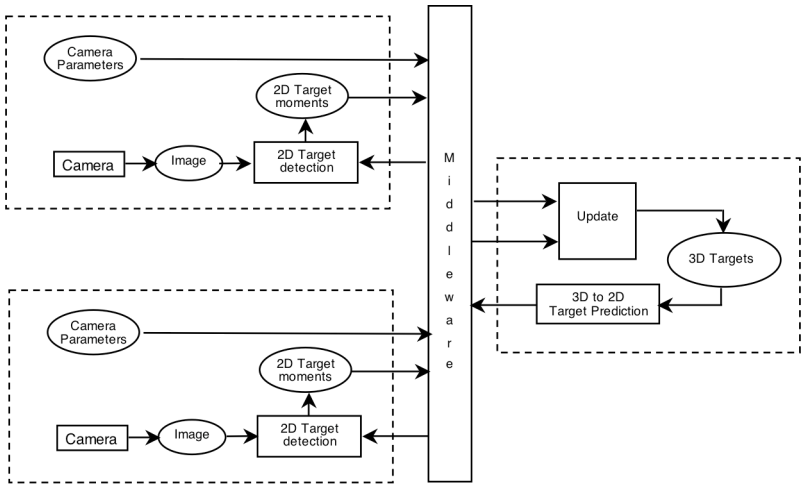

Fig. 5. An example of a system for tracking of blobs in 3D. The 3D Bayesian blob tracker provides a ROI and detection method for a number of 2D entity detection components. The result is used to update a list of 3D blobs.

As a simple example of a service provided by an assembly of perceptual components, consider a system that integrates targets from multiple cameras to provide 3-D target tracking, as shown in figure 4. A set of tracked entities is provided by a Bayesian 3D tracking process that tracks targets in 3D scene coordinates [20]. This process specifies the predicted 2-D Region of Interest (ROI) and detection method for a set of pixel-level detection components. These components use color, motion or background difference subtraction to detect and track blobs in an image stream from a camera. The O3MICID middle ware makes it possible to dynamically add or drop cameras to the process during tracking.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

3. Methods for autonomic perceptual components

In this section we briefly describe methods for self-monitoring, auto-regulation, and self repair of perceptual components.

3.1 Self-monitoring and Auto-regulation

As described above, a supervisor can sum the execution times from the auto-report from modules to determine the most recent cycle time. Excessive execution time can be reduced by such methods as removing targets from the tracked target list, reducing the frequency of calls to the detection regions used to detect new targets, or directing some detection modules to reduce resolution by processing 1 out of S pixels.

In addition, a component supervisor can be made to detect errors in the current targets using statistical pattern recognition. Our approach is to use a binary classifier whose parameters have been trained from data that is known to be valid. When the parameters of tracked entities are classified as erroneous, a second, multi-class classifier can be applied to select a repair procedure.

The goal of the auto-critical evaluation is to monitor the performance of the system in order to detect degradation in performance. This requires the definition of a measure that estimates the goodness of component output with respect to a reference model that is constructed in a learning phase. Such a reference model captures normal component output, and provides a likelihood that the current outputs are normal.

There are a variety of representation forms that could provide a reference models. These include histograms, graphs and Gaussian Mixture Models (GMMs). Histograms can often provide a good solution to concrete problems despite their simplicity. Probability density approximations such as histograms or GMMs have the advantage that a goodness score can be defined easily based on statistical estimation. All probabilistic reference models have in common that they estimate the true probability density function (pdf) of measurements. For example a pdf represented by a GMM can be obtained by applying a standard learning approach such as clustering to the training data and representing each cluster by a Gaussian. An alternative approach, proposed by Makris and Ellis, [21] is to learn entry and exit points from examples and represent them as a Gaussian mixture. Trajectories are represented by a topological graph.

The scene reference model together with a quality metric forms the knowledge base of the self-adaptive system. It allows the system to judge the quality of the system output and to select parameters that are optimal with respect to a quality metric. The success of the self-adaptive technique depends on the representativeness of this scene reference model and its metric. As a consequence, model generation is an important step within this approach.

For commercial applications, incremental techniques for learning the scene reference model are especially desirable, because only a limited number of ground truth data may be available for initialization for each environment. Such techniques have the great advantage, that they can be refined as more data becomes available.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

3.2 Error Recovery

According to Kephart and Chess [2], error recovery (or self-healing in their terminology) is composed of three phases: error detection, error diagnosis, and error repair. We have explored techniques for self-healing perceptual components based on the addition of modules for "Error Detection", "Error Classification" and "Error Repair" for our tracking-based perceptual component.

The "Error Detection" module monitors the output of a perceptual component maintaining a target history and a binary classifier. A variety of different binary classification methods may be used for error detection. In our experiments, we have used a support vector machine, followed by a running average of the distance from the decision boundary for the last N cycles. As long as the running average remains within the normal range, targets parameters are simply added to the target history. When a target is classified as abnormal recent sequence is extracted from the target history for further processing.

The recent target history for an abnormal target is passed to the error classification component for diagnosis. Error classification is a two-stage multi-class recognition process that assigns the target history to one of a known set of error classes, or a special "unkown" class. The first stage classifier discriminates known from unknown error classes. The second stage assigns the target history to one of the known error classes. Both stages are implemented as support vector machines.

Error classification is based on a set of features extracted from the recent target history. Features that we have used include

- the mean target area
- the mean target elongation
- the mean target motion energy
- the mean target area variation energy:

Both error classifiers are implemented as support vector machines (SVM) classifiers using the Radial Basis Function (RBF) kernel.

The first SVM classifier is trained by considering all training data from all known error classes as a single class. The implementation uses LIBSVM [22] to learn this one-class SVM. The second classifier is a multi-class SVM learnt using the samples of the known error classes. A RBF kernel is also used for this classifier.

A database of repair strategies associates each error class with a particular strategy. In a fully automatic system, the strategies would be discovered automatically from a large set of examples and a large set of possible commands for error repair. An appropriate method would be the acquisition of successful error repair procedures by trial and error. Unfortunately, such a fully automatic system would require a very long time to generate a successful set of repair procedures.

As with other systems that use expert knowledge to control a vision system [23], [24] use hand coded repair procedures. Such repair procedures capture the expert knowledge of the designer. We have used this approach to design simple but efficient repair strategies for perceptual components.

New repair strategies can be added during the system lifetime as new classes of errors are detected. Periodically, the "unkown" classes are reviewed by a human who may assign the history to a known error and thus update the error classifier, or assign

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8
This document and other contributions archived and available at: http://biecoll.ub.uni-bielefeld.de

them to a new error class. When a new error class is detected, the classification process is updated, and a repair procedure is selected or encoded by the human. Error repair may involve deleting spurious targets, changing target parameters or detection module used for that target, change parameters for initial target detection or changing the set of modules used by the supervisor during each cycle.

There is not necessarily a one-to-one relation between error classes and repair strategies: a given strategy can be used to repair error from various classes. In the case of the tracking system, we use some simple repair procedures such as killing targets that are identified as false positives by the classifier, refreshing the background in problematic regions or even doing nothing (e.g. in the case of a "not an error" class or when the erroneous target has already disappeared).

4. Conclusions

Autonomic computing makes possible the design of systems that adapt processing to changes in operating conditions. In this paper we have described how autonomic methods can be included within a component-oriented design. Software components for computer vision can be provided with automatic parameter initialization and regulation, self-monitoring and error detection and repair, to provide systems in services may be assembled dynamically from a collection of independent components.

References

1. P. Horn, “Autonomic Computing: IBM's Perspective on the State of Information Technology,” http://researchweb.watson.ibm.com/autonomic, Oct. 15, 2001.
2. J. O. Kephart, and D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, Vol. 36, No. 1, pp 41-50, Jan. 2003.
3. V. Poladian et al., “Dynamic Configuration of Resource-Aware Services,” Proc. 24th Int. Conf. Software Engineering, pp. 604-613, May 2004.
4. Murino, V., Foresti, G., and Regazzoni, C. (1996). A distributed probabilistic system for adaptive regulation of image processing parameters. IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics, 26(1):1–20.
5. Scotti, G., Marcenaro, L., and Regazzoni, C. (2003). A s.o.m. based algorithm for video surveillance system parameter optimal selection. In Advanced Video and Signal Based Surveillance.
6. Min, J., Powell, M., and Bowyer, K. (2004). Automated performance evaluation of range image segmentation algorithms. IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics, 34(1):263–271
7. P. Robertson and J. M. Brady, "Adaptive image analysis for aerial surveillance", IEEE Intelligent Systems, 14(3), pp30–36, May/June 1999.
8. D. Hall, R. Emonet, J. L. Crowley, "An automatic approach for parameter selection in self-adaptive tracking", International Conference on Computer Vision Theory and Applications (VISAPP), Springer Verlag, Setúbal, Portugal, Feb., 2006
9. D. Hall, "Automatic parameter regulation of perceptual systems", in Image and Vision Computing, Volume 24, Issue 8, pp 870-881, Aug. 2006.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8

10. J. L. Crowley, "Integration and Control of Reactive Visual Processes", Robotics and Autonomous Systems, Vol 15, No. 1, december 1995.
11. A. Finkelstein, J. Kramer and B. Nuseibeh, "Software Process Modeling and Technology", Research Studies Press, John Wiley and Sons Inc, 1994.
12. M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Disciplines, Prentice Hall, 1996.
13. K. Schwerdt and J. L. Crowley, "Robust Face Tracking using Color", 4th IEEE International Conference on Automatic Face and Gesture Recognition", Grenoble, France, March 2000.
14. J. Piater and J. Crowley, "Event-based Activity Analysis in Live Video using a Generic Object Tracker", Performance Evaluation for Tracking and Surveillance, PETS-2002, Copenhagen, June 2002.
15. A. Lux, "The Imalab Method for Vision Systems", International Conference on Vision Systems, ICVS-03, Graz, april 2003.
16. M. Danninger, T. Kluge, R. Stiefelhagen, "MyConnector: analysis of context cues to predict human availability for communication", International Conference on Multimodal Interaction, ICMI 2006: pp12-19, Trento, 2006.
17. J. L. Crowley, O. Brdiczka, and P. Reignier, "Learning Situation Models for Understanding Activity", In The 5th International Conference on Development and Learning 2006 (ICDL06), Bloomington, Il., USA, June 2006.
18. R. Kalman, "A new approach to Linear Filtering and Prediction Problems", Transactions of the ASME, Series D. J. Basic Eng., Vol 82, 1960.
19. R. Emonet, D. Vaufreydaz, P. Reignier, J. Letessier, "O3MiSCID: an Object Oriented Opensource Middleware for Service Connection, Introspection an Discover", 1st IEEE International Workshop on Services Integration in Pervasive Environments - June 2006
20. A. Ferrer-Biosca and A. Lux, "A Visual Service for Distributed Environments: a Bayesian 3D Person Tracker", PRIMA internal Report, 2007.
21. Makris, D. and Ellis, T. (2003). Automatic learning of an activity-based semantic scene model. In Advanced Video and Signal Based Surveillance, pages 183–188.
22. C.-C. Chang and C.J. Lin. Libsvm - a library for support vector machines. available
23. B. Géoris, F. Brémond, M. Thonnat, and B. Macq, "Use of an evaluation and diagnosis method to improve tracking performances", In International Conference on Visualization, Imaging and Image Proceeding, September 2003.
24. C. Shekhar, S. Moisan, R. Vincent, P. Burlina, and R. Chellappa, "Knowledge-based control of vision systems", Image and Vision Computing, 17(9), pp 667–683, 1999.

Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)
Published in 2007 by Applied Computer Science Group, Bielefeld University, Germany, ISBN 978-3-00-020933-8