INVICON: A Toolkit for Knowledge-Based Control of Vision Systems

Olena Borzenko, Yves Lespérance, and Michael Jenkin
Computer Science and Engineering, York University
4700 Keele Street, Toronto, ON M3J 1P3, Canada
{olena, lesperan, jenkin}@cse.yorku.ca

Abstract

To perform as desired in a dynamic environment a vision system must adapt to a variety of operating conditions by selecting vision modules, tuning their parameters, and controlling image acquisition. Knowledge-based (KB) controller-agents that reason over explicitly represented knowledge and interact with their environment can be used for this task; however, the lack of a unifying methodology and development tools makes KB controllers difficult to create, maintain, and reuse. This paper presents the INVICON toolkit, based on the IndiGolog agent programming language with elements from control theory. It provides a basic methodology, a vision module declaration template, a suite of control components, and support tools for KB controller development. We have evaluated INVICON in two case studies that involved controlling vision-based pose estimation systems. The case studies show that INVICON reduces the effort needed to build KB controllers for challenging domains and improves their flexibility and robustness.

1. Introduction

Modern vision systems typically perform multiple processing stages to acquire and map images to high-level interpretations. Each of these stages can perform well only under a limited set of conditions. Vision systems that operate in dynamic environments must be able to adapt to varying operating conditions by configuring and tuning themselves, preferably with minimum human intervention. A separate software component, an autonomous *vision controller*, can be used to achieve the desired quality of operation. Its task generally involves solving one or more of the following sub-problems:

- **Parameter Adjustment** Generic vision algorithms typically incorporate tunable parameters. A control problem is to continually adjust the algorithms' parameter values for the current conditions [16, 2].
- **Module Selection** Even the best vision algorithms have a limited operating range. Another control problem is to select appropriate algorithms or software modules for various sub-tasks [16, 4].
- **Image Acquisition** Image acquisition conditions greatly influence the quality of subsequent steps. Another control problem is to select acquisition parameters such as the intensity of illuminants or the position and orientation of the camera [2, 13].
- **Monitoring and Reconfiguration** A vision system architecture can support system integration and coordination of its multiple processes. A related control problem is to monitor the system's execution and reconfigure the system when necessary, e.g., by autonomously detecting and recovering from failures [6].

Vision system controllers are typically evaluated by how they meet *domain-specific* requirements imposed by the application. These can include various degrees of flexibility, efficient use of resources, processing speed, accuracy, and robustness. At the same time, as domain requirements change and new application domains arise, *software engineering* requirements such as scalability, reusability, and transparency also need to be taken into account [19].

Consider the example of vision-based pose estimation for space docking [2] which is the subject of our first case study. To correctly approach a docking platform, the docking spacecraft needs a precise estimate of its relative position and orientation. The vision system can compute this estimate from one or more images of the docking target. Controllable illuminants can be used to compensate for poor and variable on-orbit illumination. In this context, the task of a vision controller is to select which images to use so that the vision system can produce as precise results as possible (*accuracy*) whenever a pose estimate is required (*anytimeness*). Meanwhile, it is desirable to minimize the time and effort required for the controller's design, maintenance, and possibly reuse.

Several approaches to vision control have appeared in the literature. Generic feedback control architectures can be used to create vision controllers, but building mathematical models of the vision systems controlled and their operating environments is often unfeasible. Knowledge-based (KB) approaches offer alternative models which explicitly represent the available incomplete and imprecise knowledge about vision systems and their operating environment; this knowledge can be reasoned with to make control decisions. Among major KB approaches, KB intelligent agents are particularly suitable for dynamic environments. In contrast to expert systems, KB agents are able to perceive, reason about, and interact with their environment. The main problem with KB vision controllers is that without a generic framework they can be difficult to create, maintain, and reuse: domain and control knowledge can be expensive to obtain and represent and is often represented in ways which are not easily generalizable to other tasks and domains. What is needed is a methodology and tools that alleviate the effort of the designer and allow the construction of efficient and reusable KB agent-controllers.

To this end, we have developed the INVICON (INtelligent/INdigolog VIsion CONtrol) toolkit[1] [1] which supports the development of KB vision controllers. Our agent programming framework is based on the Situation Calculus [12] for representing dynamic domains and the high-level model-based agent programming language IndiGolog [8] for specifying controller behaviour. We also adopt some elements from classical control theory. The main idea behind INVICON's design is that a agent-controller manages one or several *vision modules*. An abstract vision module may refer to an implementation of a particular vision method or to an integrated vision system with several processing stages. In either case, following the control metaphor, the vision module is described in terms of control inputs (*parameters*) and output results and performance metrics (*results*). In control theory terms, the vision module is then a *plant* integrated with a *QoS* (quality of service) module. Following the basic action theory approach to domain modelling, the controller maintains representations of the module's dynamic state as *fluents*, queries and changes the module's parameter values via *sensing* and *world-changing actions*, and reacts to reports about the module's results viewed as *exogenous actions*.

The rest of the paper is organized as follows. Section 2 introduces the theoretical foundations of our framework, the Situation Calculus and IndiGolog. Section 3 describes the architecture of a stand-alone IndiGolog-based vision controller. Sections 4 and 5 present the INVICON template controller architecture and controller development guidelines. Section 6 discusses the results of two case studies where INVICON was used. Section 7 discusses related work on vision control. Finally, Section 8 summarizes our contributions and outlines future research directions.

[1]The toolkit can be obtained by contacting the first author.

α	primitive action
$\phi?$	wait for a condition
$\delta_1 ; \delta_2$	sequence
$\delta_1 \mid \delta_2$	nondeterministic branch
$\pi\, \vec{x}[\delta]$	nondeterministic choice of argument
δ^*	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2 **endIf**	conditional
while ϕ **do** δ **endWhile**	while loop
proc $\beta(\vec{x})\ \delta$ **endProc**	procedure definition
$\beta(\vec{t})$	procedure call
$\delta_1 \parallel \delta_2$	concurrency with equal priorities
$\delta_1 \gg \delta_2$	concurrency with different priorities
$\delta^{\parallel}$	concurrent iteration
$< \phi \rightarrow \delta >$	interrupt
$\Sigma(\delta)$	search for an execution of δ
noOp	do nothing

Figure 1. IndiGolog programming constructs.

2. Theoretical Foundations

The *Situation Calculus* [12] is a language of predicate logic for representing dynamically changing worlds. All changes in the world are assumed to be caused by the execution of some *actions*. At any moment, the world is considered to be in a particular *situation*, which is viewed as the sequence of actions that has led to it. The initial situation S_0 is the situation in which no action has yet occurred. A binary function $do(a, s)$ denotes the successor situation of the situation s after the execution of action a. Any other domain entities are referred to as domain *objects*. The changing state of the domain is represented through *fluents*, which are predicates and functions that take a situation term as their last argument and may change value from situation to situation. The configuration of a domain is captured by the actual values assigned to its fluents in the current situation. Overall, the dynamic world is specified by a *basic action theory*, that includes the following types of axioms: domain-independent *foundational axioms* for situations [12], *initial state axioms*, which describe the initial situation, *action precondition axioms*, one for every action, which specify the conditions under which the action can be performed in a situation, *successor state axioms*, one for every fluent, which characterize how the fluent is affected by actions, and *unique names axioms* for the primitive actions.

IndiGolog (IG) [8] is a high-level model-based agent programming language, a successor of Golog and ConGolog [3]. In these languages, the programmer provides a declarative specification of the domain in the Situation Calculus and develops control programs in terms of the primitive actions and fluents. IG provides a rich set of programming constructs for specifying agent's behaviour (Figure 1). This includes support for concurrency, process priorities, interrupts, as well as for planning/search within an overall deterministic program that is to be executed incrementally in conjunction with sensing of the environment. The IG planning mechanism automatically monitors the execution

```
 % One aspect of the world state
prim_fluent(
  moduleParam(vismodule, threshold1)).
 % A sensing action
prim_action(                % declaration
  getPara(vismodule, threshold1)).
poss(                       % precondition
  getPara(vismodule, threshold1),
    connected_to(visserver) = true).
real_execute(               % implementation
  getPara(vismodule, threshold1), Value):-
    send_get_msg(vismodule, threshold1),
    get_response(Value).
senses(                     % fluent sensed
  getPara(vismodule, threshold1),
    serverPara(vismodule, threshold1)).
```

Figure 2. Fragment of a sample domain specification.

of the generated plan and re-plans when the current plan is no longer appropriate due to sensed changes in the environment. IG is implemented on top of Prolog. For a controller-agent to be able to reason about its perceived world and its vision system, the controller needs to model all relevant aspects of the environment and vision modules. An agent programming language such as IG is an ideal basis for developing high-level vision controllers. IG provides a transparent and scalable framework that encourages declarative description of the domain and performs automatic state updates. IG supports both planning and high-level reactivity, and it supports rapid prototyping and programming at a high level of abstraction.

3. Stand-alone Controllers in IndiGolog

Declarative specifications of the states of vision modules controlled and the actions that affect those states form the basis of a KB vision controller as they model the modules' states and their dynamics. A code fragment for a sample vision controller in IG is provided in Figure 2. The specifications of a sensing action, for example, contain the action declaration, its preconditions, the procedure that carries out the action in the physical world, and the fluent whose value it updates. Without a generic framework, the patterns of definitions need to be repeated for every action and fluent.

The components of an IG vision controller are shown in Figure 3. The Declarative Part provides a declarative specification of the domain in terms of fluents and actions that affect them. The Procedural Part provides specifications for control procedures and complex conditions tested, using the IG programming constructs. The Parser performs conversion between the text messages sent between the controller and the vision module and the actions that they rep-

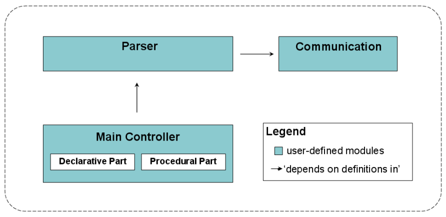


Figure 3. Stand-alone controller.

resent. The Communication component implements low-level message handling and vision module connectivity. In the case of a stand-alone development effort, all of these components need to be supplied by the vision controller designer.

In summary, although the agent-based approach provides a rich framework for building vision controllers, the stand-alone development of a controller-agent can be a difficult task: large domain descriptions can easily become unmanageable; the bulk of the control and support components needs to be developed from scratch.

4. The INVICON Template Controller

The INVICON toolkit aims to overcome these limitations by offering an alternative, *template-based architecture* (Figure 4) which covers the same functionality as the stand-alone controller architecture (Figure 3), but eases the designer's task. We describe it component by component below.

Declarative Part. To bring down the complexity of domain specification, INVICON provides a template basic action theory specification for an abstract vision module, the Vision Module Declaration Template. This includes fluents for the module's parameters and results and actions for the basic operations of setting/sensing parameter values and receiving results. Instead of a lengthy, repetitive domain description, a high-level description of a Vision Module, which contains the module's name, a list of parameters, and a list of results, can now be specified using Prolog predicates (see Figure 5). INVICON automatically instantiates a corresponding IG representation in terms of the fluents `serverPara(Module,Time,Param)` and `serverResult(Module,Time,Result)`, the primitive actions `setPara(Module,Param)` and `getResult(Module,Result)` which update the parameter and request the result by sending messages to the vision module, the sensing action `getPara(Module,Param)` which senses the parameter value by communicating with the vision module, and the exogenous actions `para(Module,Param,Value)` and `result(Module,Result,Value)` which update the values of the parameter and result fluents and are generated

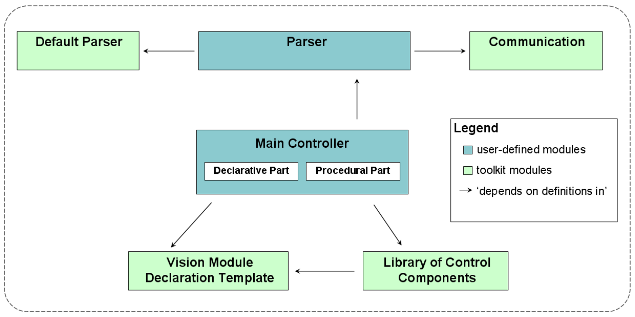


Figure 4. Template controller.

```
vision_module(vismodule, [threshold1],
                         [certainty]).
data_type(threshold1, float).
data_range(threshold1, 0.0, 10.0).
data_type(certainty, float).
data_range(certainty, 0.0, 1.0).
result_is_better(certainty, X, Y):- X > Y.
result_is_same(certainty, X, X).
```

Figure 5. An example high-level description of a Vision Module.

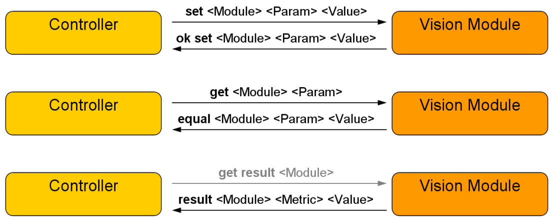
when messages are received from the vision module. These work similarly to the specifications provided in Figure 2. To manage several vision modules, a controller can include several vision module declarations.

Procedural Part. Building upon the generic vision module representation, we can now define domain-independent, reusable control components. The control task is generally posed as an optimization problem and solved by searching through a space of possible combinations of parameter values, seeking to maximize/minimize a performance metric. Several classes of search methods can be used for vision parameter control, including flavours of hill-climbing, heuristic search, genetic algorithms, and other methods. The initial version of the Library of Control Components contains implementions of basic search methods using Prolog and IG:

- *Hill-climbing search*, which iteratively increments/decrements the value of a parameter to find a local optimum of a performance metric, based on the declared ordering.

- *Random search*, which picks random valid values for all parameters of the module, that is, finds a random element of the search space.

- *Heuristic search*, which picks a random element that satisfies a certain condition.



Controller
set <Module> <Param> <Value>
ok set <Module> <Param> <Value>
Vision Module
Controller
get <Module> <Param>
equal <Module> <Param> <Value>
Vision Module
Controller
get result <Module>
result <Module> <Metric> <Value>
Vision Module

Figure 6. Default communication protocol for a Controller and a Vision Module.

New search and other control components can easily be added to the Library: the designer can abstract from the specifics of the controlled module, use random value generators and other components supplied by the toolkit, and rely on INVICON to automatically validate parameter values.

Parser and Communication Protocol. INVICON supports a default communication protocol (see Figure 6). Three types of message exchanges can occur: the Controller can set a parameter value and receive a confirmation, it can request a parameter value and receive a report, or it can receive a report of results (a requesting message may or may not be required). The Default Parser, implemented in Prolog, converts incoming messages to the corresponding exogenous actions understood by the controller; it executes the actions chosen by the controller by converting them into outgoing messages. These generic conversion procedures are also automatically instantiated by the vision module declaration.

Low-Level Communication. These utilities implemented using SWI Prolog libraries for threads and socket connectivity are application-independent.

5. Controller Development using INVICON

In addition to the template controller architecture, INVICON provides guidelines for KB vision controller design and implementation. We illustrate this with examples from Case Study I which involved the development of high-level controllers for the “Lights and Camera” (L&C) vision-based pose estimation testbed system [2] (see Figure 7). Within this system, the Image Server operates a camera and three controllable lights; the Vision Server constructs a composite edgel map from the collection of images and employs a model-matching genetic algorithm to estimate the target’s pose; the Controller needs to select light settings and images to use at each iteration.

5.1 Design

Choose vision parameters to control (actuator variables). Actuator variables are those adjustable parameters

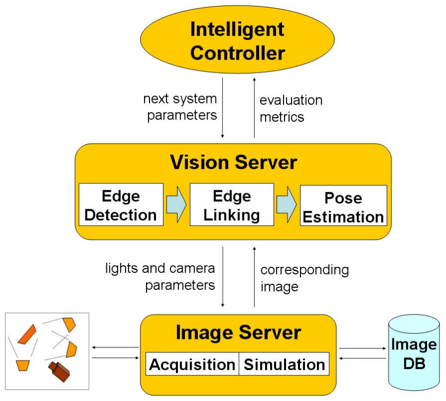


Figure 7. The "Lights and Camera" pose estimation system.

of a processing stage in a vision system that directly influence the performance of this stage [13]. In the L&C system, possible parameters to control fall into the following groups: *image acquisition* (shutter speed, aperture, focus, and light intensity levels), *image set selection* (parameters that control adding and removing images from the set), and *image processing* (parameters for the edge detection, edge linking, and pose estimation stages). Among these groups, the determining factors of the system's performance are the image acquisition conditions, specifically, the shutter speed of the camera ($shutter$) and the light intensities, with 8 possible levels each ($light_1$, $light_2$, $light_3$). In the L&C testbed, we assumed that the shutter speed had been initially adjusted under medium-range illumination. The system is not sensitive to the order of insertion of images in the image set, so the Controller updates the set incrementally, by extending it with a newly acquired image or by replacing a previously added image in the set with a new one. To simplify the control task, image processing parameters are adjusted offline and set to fixed values during system operation. Manipulating the intensities of the three lights and deciding which images to keep in the set is the focus of the vision controller.

Select control granularity/architecture. Is one control loop sufficient, or are several sub-loops for tuning different processing stages or parameters necessary? On one hand, if the variability in performance is coming mostly from one stage, parameters that control this stage can be manipulated in a single control loop, while the rest of system parameters can be set to reasonable values, perhaps by tuning them offline. On the other hand, if parameter values for certain stages do not generalize well for all possible inputs to those stages, then fixing these parameters will compromise performance. So separate control loops may be dedicated to tuning parameters for these critical stages, starting with the parameters which affect earlier processing stages or cause more variability in performance. Similarly, if the performance of a particular stage depends jointly on several parameters, these parameters may have to be considered together and tuned inside a single control loop. Otherwise, under the assumption that the parameter search dimensions are independent, the parameters can be tuned separately, e.g, in sequence.

Choose the performance metrics (controlled variables). One needs to estimate performance quality to be able to optimize it. The basic approach is to look at the requirements of the next processing step (or at the application requirements when evaluating the whole system). The goal of the L&C system is to obtain a precise estimate of the target's pose. Since the actual pose of the target is not directly available during system operation, the measure of the system's performance is given by the value of a fitness function that corresponds to the output of the pose estimation algorithm. Minimizing this *pose estimation uncertainty* optimizes the match between the L&C 3D model data and the target's projection into the imaging system; the better the overall match, the more confident the system is in its interpretation of the visual data.

In general, choosing a controlled variable/performance metric is not a trivial problem [13]. From the point of view of vision processing, this metric should reflect which characteristics of output images or other intermediate representations would be beneficial at the next processing stage. From the point of view of control, the value of this metric should be largely affected by the changes in actuator variables; in addition, its computation should be relatively easy.

Consider additional domain knowledge. If the effects of parameter changes interact with each other, is there a way to qualitatively describe (or perhaps learn) the relation? One can find an abstract representation of the search space, reduce the search problem to a set of simpler problems, or look for additional domain knowledge that could otherwise simplify the search. In general, the vision module declaration does not represent the whole action theory for the domain, so it is essential that the designer identify and provide axioms for additional, application-specific knowledge.

In the L&C application, the three light intensities cannot be considered independent search dimensions and optimized separately, so a search space reduction approach was taken. The Controller maintains qualitative representations of system states, e.g., "low lighting coming evenly from all the three light sources", in essence discretizing along the "light brightness" and "light directionality" dimensions. The reduced search space contains four brightness and five directionality types. Images taken under the same directionality and brightness of light are assigned to the same category. They are similar – not necessarily in terms of pixel data, but rather in originating from comparable physical se-

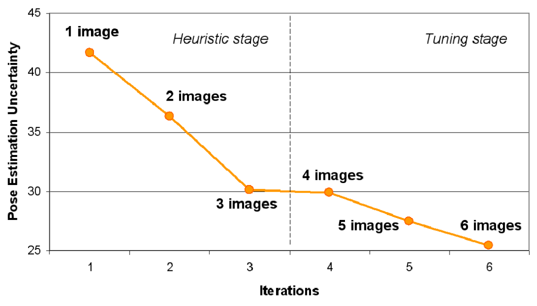


Figure 8. Sample run of the two-stage controller: uncertainty of the best pose estimate at each iteration is given.

tups. The assumption is that images similar to those previously acquired are less likely to improve pose estimation uncertainty because they carry similar information.

Experiment with control strategies. A preliminary examination of the structure of the application domain typically suggests a number of possible control strategies. Experimenting with those helps narrow down the choice. A good place to start is the INVICON library of control components. New search components can also be created to suit the specifics of the application domain.

In Case Study I we experimented with 4 control approaches:

- The **heuristic** KB controller, implemented using the *heuristic search* component, acquires images which are *not similar* to images previously acquired; after each acquisition it evaluates pose estimation uncertainty and discards the last image from the image set if the uncertainty does not decrease.

- The **two-stage** KB controller (Figure 8) initially follows the *heuristic* control procedure and then enters the *tuning* stage: it determines which face of the target contains the line matched with the highest uncertainty and randomly selects illumination parameters that are similar to a template associated with that face in the knowledge base (and considered likely to reduce uncertainty).

- The **simple** controller acquires and adds randomly selected images, using the *random search* component.

- The **greedy** controller also acquires randomly selected images, but keeps the new image in the image set only if that improves the pose estimation uncertainty.

We discuss experimental results in Section 6.

5.2 Implementation

Instantiating vision module templates. Once the control granularity, parameters, and performance metrics are chosen, the designer can provide high-level *vision modules* declarations (see Figure 5). Each vision module generally corresponds to a set of parameters and performance metrics that are handled by one or more control loops.

Representing additional knowledge. Additional domain and control knowledge can be represented in both Prolog and IG. The designer can take advantage of the strengths of these languages by choosing a representation suitable for the task at hand. In Case Study I, the categorization scheme for representing the states of the light sources could be concisely defined as a Prolog predicate as follows:

```
img_similarity(Img1,Img2,1.0):-
           same_brightness(Img1,Img2),
           % AND
           same_directionality(Img1,Img2), !.
img_similarity(Img1,Img2,0.5):-
           same_brightness(Img1,Img2);
           % OR
           same_directionality(Img1,Img2), !.
img_similarity(_Img1,_Img2,0.0).
```

Control procedures such as the *heuristic* controller – which augments the image set with images that are *not similar* to the sets of current or discarded images, confirming or discarding the new image until the uncertainty threshold or the maximum number of iterations have been reached – could be defined in IG as follows:

```
initialize(imgset),
while(and(high_uncertainty(pose),
          less_than_max(iterations)), [
   add_img(imgset, not_similar),
   if(uncertainty_improved(pose),
     % then
     confirm_img(imgset),
     % else
     discard_img(imgset))
   ])
```

Changing the communication protocol. To change the interpretation of incoming messages, the designer can add new rules to the Parser, for example, to modify the message format or add a new message type. To change the generation of outgoing messages, the designer can provide custom implementations of the primitive and sensing actions.

6. Results of Case Studies

Experiments for **Case Study I** consisted of 30 runs of each controller type for a simple cubical target. The average performance profiles are presented in Figure 9. Points on the graph represent the average pose estimation uncertainty at a given iteration for a given controller type. The

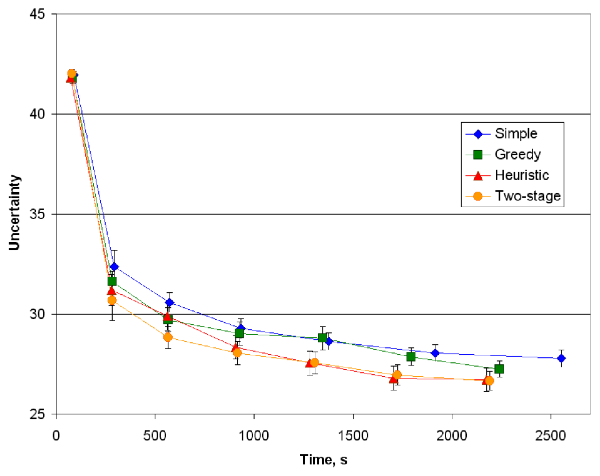


Figure 9. Average performance profiles. The error bars show the standard error for the pose estimation uncertainty.

time stamps for each data point are averaged across the controller runs. The *heuristic* and *two-stage* KB controllers perform comparably overall, outperforming the baseline *simple* and *greedy* controllers, although the trend is not statistically significant during the initial iterations. Both KB controllers have been used to successfully control the L&C system in laboratory conditions. More details can be found in [1]. The use of INVICON has facilitated several aspects of the development of the L&C controllers. Using INVICON, the essential domain and control knowledge could be specified in a high-level, compact, and extendable way. This avoided the problems of ad-hoc controller development and saved time and effort in testing and debugging. Due to the appropriate level of abstraction and the decoupling of the domain and control knowledge, control procedures can now be easily reused. The availability of generic control components sped up the implementation and allowed for experimentation with different control methods. Finally, the INVICON software infrastructure fully supported high-level communication with the vision system.

Case Study II involved improvements to an existing IG vision controller which supervises a vision module that performs two-dimensional model-matching to estimate a target's size and position on a single image. The controller adjusts the module's edge detection and model-matching parameters in sequence using search techniques such as hill-climbing. The use of INVICON drastically reduced the amount of code required, from 1051 lines in the original version to 354 lines in the INVICON-based version. INVICON also made the code easier to modify and extend. Should a new parameter of the module need to be adjusted, one would only need to add the parameter in the module declaration and provide an additional control procedure call to adjust it. Should new control procedures be defined, they could replace the currently used hill-climbing procedures simply by changing the appropriate procedure calls. In the original version of the controller, the control procedures were heavily (and unnecessarily) mixed with domain knowledge (parameters and results of the module), and the logic of the control procedures themselves was not transparent. In the revised version that used the INVICON toolkit, more abstract control procedures were used and the separation between the domain and control knowledge was made explicit. The toolkit also helped correct minor errors and eliminate some unused fluents and actions.

7. Related Work

Approaches to vision control. *Control theory*-based approaches, more specifically, feedback control strategies have been used in computer vision systems for tasks such as character recognition [13], object recognition [11], and image segmentation [14]. Although control theory notions such as stability analysis are not directly applicable to the vision system domain [14], empirical results show that feedback is essential; evaluating system performance and optimizing its processing stages increases robustness and confidence of the vision system.

Early KB vision control systems were *expert systems* that mostly targeted image segmentation and high-level image interpretation tasks (e.g., [10]). They were typically ad hoc and difficult to port from one application domain to another. In addition, expert system shells do not directly support dealing with dynamic domains. KB *program supervision* techniques [17] can automate the selection and tuning of image processing procedures, e.g., for offline batch image processing [16]. The existing systems, however, have not been proven suitable for autonomous, online vision system control as they rely on feedback from a human user. An *intelligent agent* is defined as a computer system that is able to *autonomously* perceive its environment, react to changes in it, exhibit goal-directed behaviour, and interact with other agents to meet its design objectives [18]. Examples of agent-based vision control systems include an active perception framework by [15], which incorporates feedback and expectation and is based on the Event Calculus, and KB vision system supervisors described in [1], which are based on the IG language, as is our framework, but are developed using a stand-alone approach.

Another set of approaches is based on *probabilistic modeling* of the control process itself, typically as a Bayesian network or Markov decision process [4]. Probabilistic reasoning typically enhances the agent's ability to act autonomously in uncertain, noisy domains, but combining quantitative and qualitative approaches can be challenging.

Tools for Vision and Robotics. One limitation of INVICON is that it only supports the vision controller-side of system development. In this sense, the INVICON toolkit is complementary to programming platforms such as AV-shell [5]; AV-shell is a full vision system programming environment, while INVICON provides for the actual agent-based specification of high-level behaviours that AV-shell lacks. Robotic architectures such as Saphira [9] do include support for high-level specification of behaviours, e.g., for navigation using stereo vision, but they are mostly tailored towards robotic applications and are, therefore, difficult to use in other vision domains. INVICON does not share this restriction. Another robotic toolkit, GenoM [7] allows for the specification and integration of functional modules which perform servo-control, data processing, and event monitoring operations. As in our work, a generic model for modules is provided, although this is done in the context of a distributed reactive robot architecture, while we focus on the development of single agent module controllers.

8. Conclusions and Future Work

This paper has presented INVICON, an agent-oriented toolkit for the developement of KB vision controllers, based on the IndiGolog programming language. INVICON provides a design methodology, a generic vision module specification, a library of control components, and support utilities. Two case studies showed that INVICON reduces the effort needed to build KB vision controllers and improves their flexibility and robustness.

In the future, the library of control components needs to be expanded to include implementations of other search methods. In the case studies, we developed controllers for vision systems that perform pose estimation. Other computer vision tasks such as image segmentation, object recognition, and action recognition need to be explored to fully evaluate our tools and to develop control components suitable for these tasks. Finally, INVICON could be extended to support a flexible meta-controller architecture that could perform failure handling and algorithm selection.

References

[1] O. Borzenko. Knowledge-based control of vision systems: Design tools and case studies. Master's thesis, Dept. of Comp. Science and Eng., York University, Toronto, October 2006.

[2] O. Borzenko, W. Xu, M. Obsniuk, A. Chopra, P. Jasiobedzki, M. Jenkin, and Y. Lespérance. Lights and Camera: Intelligently controlled multi-channel pose estimation system. In *IEEE Int. Conf. Vision Systems, ICVS'06*, New York, USA, Jan 2006.

[3] G. De Giacomo, Y. Lespérance, and H. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.

[4] B. A. Draper. From knowledge bases to Markov models to PCA. In *Workshop on Computer Vision System Control Architectures*, Graz, Austria, 2003.

[5] J. A. Fayman, E. Rivlin, and H. I. Christensen. AV-shell, an environment for autonomous robotic applications using active vision. *Autonomous Robots*, 6(1):21–38, Jan 1999.

[6] A. Finzi, F. Pirri, M. Pirrone, M. Romano, and M. Vaccaro. Autonomous mobile manipulators managing perception and failures. In *5th Int. Conf. Autonomous Agents*, pages 196–203, Montreal, Canada, May 2001.

[7] S. Fleury, M. Herrb, and R. Chatila. GenoM: a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems, IROS'97*, volume 2, pages 842–849, Sep 1997.

[8] G. D. Giacomo and H. Levesque. *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, chapter An incremental interpreter for high-level programs with sensing, pages 86–102. Berlin: Springer, 1999.

[9] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1):215–235, 1997.

[10] T. Matsuyama. Knowledge-based aerial image understanding systems. In T. Guyenne and J. Hunt, editors, *IGARSS '86. Remote Sensing: Today's Solutions for Tomorrow's Information Needs (ESA SP-254)*, volume 1, pages 277–282, 1986.

[11] M. Mirmehdi, P. Palmer, J. Kittler, and H. Dabis. Feedback control strategies for object recognition. *IEEE Transactions on Image Processing*, 8(8), 1999.

[12] R. Reiter. *Knowledge in action : Logical foundations for specifying and implementing dynamical systems*. MIT Press, Cambridge, Mass., 2001.

[13] D. Ristic, S. Vuppala, and A. Graser. Feedback control for improvement of image processing: An application of recognition of characters on metallic surfaces. In *IEEE Int. Conf. Computer Vision Systems, ICVS'06*, New York, NY, Jan 2006.

[14] P. Robertson and M. Brady. Adaptive image analysis for aerial surveillance. *IEEE Intelligent Systems*, 14(3):30–36, 1999.

[15] M. Shanahan and D. Randell. A logic-based formulation of active visual perception. In *Int. Conf. Principles of Knowledge Representation and Reasoning*, pages 64–72, Whistler, BC, 2004.

[16] C. Shekhar, S. Moisan, R. Vincent, P. Burlina, and R. Chellappa. Knowledge-based control of vision systems. *Image and Vision Computing*, 17:667–683, 1998.

[17] M. Thonnat and S. Moisan. What can program supervision do for program re-use? *IEEE Proc.-Softw.*, 147(5):179–185, Oct 2000.

[18] M. J. Wooldridge. *An introduction to multiagent systems*. New York: J. Wiley, 2002.

[19] S. Wrede, M. Hanheide, C. Bauckhage, and G. Sagerer. An active memory as a model for information fusion. In *Int. Conf. Information Fusion*, number 1, pages 198–205, 2004.