Two Approaches to a Plug-and-Play Vision Architecture – CAVIAR and Psyclone

Thor List[1], José Bins[1], Robert B. Fisher[1], David Tweed[1], Kristinn R. Thórisson[2]

[1]University of Edinburgh
Institute for Perception, Action and Behaviour
King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
Email: thor.list@ed.ac.uk

[2]Reykjavík University
Center for Analysis & Design of Intelligent Agents
Department of Computer Science
Ofanleiti 2, 203 Reykjavik, Iceland

Abstract

This paper compares two solutions for human-like perception using two different modular "plug-and-play" frameworks, CAVIAR (List et al, 2005) and Psyclone (Thórisson et al, 2004, 2005a). Each uses a central point of configuration and requires the modules to be auto-descriptive, auto-critical and auto-regulative (Crowley and Reignier, 2003) for fully autonomous configuration of processing and dataflow. This allows new modules to be added to or removed from the system with minimal reconfiguration. CAVIAR uses a centralised global controller (Bins et al, 2005) whereas Psyclone supports a fully distributed control architecture.

We implemented a computer vision-based human behaviour tracker for public scenes in the two frameworks. CAVIAR's global controller uses offline learned knowledge to regulate module parameters and select between competing results whereas in Psyclone dynamic multi-level control modules adjust parameters, data and process flow. Each framework results in two very different solutions to control issues such as dataflow regulation and module substitution. However, we found that both frameworks allow easy incremental development of modular architectures with increasingly complex functionality. Their main differences lie in runtime efficiency and module interface semantics.

Introduction

Several software architectures for human-like perception have emerged over the past years in research areas such as computer vision, human-computer interaction and autonomous robotic platforms. Early examples are SIGMA (Matsuyama and Hwang, 1993) and VISIONS (Hanson and Riseman, 1988). More recent perception architectures include the Leeds People Tracker/ADVISOR (Siebel, 2003), the Robust Tracker (Crowley and Reignier, 2003) as well as very advanced neural network-like perception systems like the Neuron-Like Processing Machine (NLPM) (Cohen, 2004).

Computer vision research addresses challenging problems, requiring architectures that mix data flow and control structures in complex ways. The interest in modular frameworks rises from the need to explore relatively large variations in architecture during their construction. Variations to be explored can range from simple changes in how parameters are tuned to module swapping. The explorations are made over weeks or months; final solutions will certainly require parameter tuning at runtime and may even require dynamic module swapping.

Although prior approaches tackle many of the same kind of challenges such as data flow regulation, process scheduling and managing situations of high data rates between many modules which need more resources than the system can provide, each specific application and implementation has unique properties which have to be addressed. A plug-and-play perception architecture should provide solutions to the common problems while providing the flexibility each module needs for independent processing of information.

The CAVIAR framework (List et al, 2005) is based on past research by (Brown and Fisher, 1990) (Crowley, 1995) (Crowley and Reignier, 2003) and is designed specifically to allow multiple concurrent implementations of equivalent modules to work in competition with each other, to be plugged in to compare different approaches to problems given varying conditions. A key research goal is investigating the control mechanisms and algorithms needed to allow this flexibility by offline learning of module parameter spaces (Bins et al, 2005).

The Psyclone framework (Thórisson et al. 2004, 2005a) by Communicative Machines is based on prior research on interactive real-time systems (Thórisson, 1999) and their construction (Thórisson et al. 2004). A key goal of this framework is to allow the creation of flexible systems, enabling researchers to test various architectures quickly through a plug-and-play modular approach, and to enable A.I. researchers to share their work more effectively.

This paper investigates the solutions offered by each approach and compares the resulting architectures when applied to the task of monitoring human behaviour in public areas through vision. We will look at differences in the approach to runtime control of parameters and the flow of data, and how new modules can be inserted into an existing system during its development stages to comparatively develop a modular system. Other differences between the frameworks that directly affect the present task will also be discussed, such as run-time performance and

semantics of module interfaces. The actual vision algorithms used are identical in both cases.

The next sections will provide a quick overview of the general features of the CAVIAR and Psyclone frameworks, focusing on the features that are different between the two. The solutions to the vision problem are then presented for each framework, and we conclude by briefly discussing the merits of each one. A comprehensive overview of the two frameworks would be too much for this paper. We review mainly those parts which we believe to be sufficient and relative to the purpose of the comparison. The reader is directed to (List et al. 2005) and (Thórisson et al., 2005b) for more detail.

CAVIAR

The CAVIAR system is based on one global controller and a number of modules for information processing. Each module provides a complete description of itself using CVML (List and Fisher, 2004) including its input and output datasets and a full list of public parameters. Each parameter description includes recommended usage such as minimum, maximum and incremental step as well as dependencies on other parameters.

The implementation includes a Base Module which contains all the common module functionalities such as the interface to the controller and to other modules for managing parameters. This way module implementers only have to deal with their own algorithms without worrying about bookkeeping tasks.

After each *run* (where one set of input data is processed to produce output data) each module will report back to the controller on the overall quality and quantity of the results. This is called the *high-level feedback* and the Base Module adds information about time and resources spent.

Each module can be auto-regulative in that it receives feedback from the controller and other modules in the form of *more of this output* and *less of that output* in terms of *quality* and *quantity*. The module may know better than anyone else how to best achieve this by regulating its own parameters or perhaps switching to use another algorithm.

A special kind of modules called *Agents* are used for simple measurements such as overall scene brightness. They assist the controller when making choices or providing feedback.

The CAVIAR controller is a global implementation of a classical system controller. At startup it reads a list of possible modules to use along with the overall goal of the system in terms of available input and desired output. Based on this and the auto-description of each module the dataflow is computed including which modules are needed in which flow configuration and whether one or more modules are each other's equivalent.

The controller can run in two modes, an online real-time mode and an offline learning mode. During the offline learning the controller runs through one or more video sequences where ground truth labelling has been provided. It can step-by-step compare its own results to the ground truth and for each module explore parts of the parameter space to learn which parameters have which influence on the output of that module and the system as a whole. It stores this knowledge in either of two ways, neural networks or dynamically created rules, to make use of the learned knowledge in the online mode when real-time constraints prevent any complex exploration of the parameter space, but modules nonetheless need to be tuned to deal with varying external conditions. The learning phase can also be used to determine which equivalent modules work best in which circumstances.

CAVIAR modules communicate data, feedback and control information through a fixed API, described in detail in (List et al. 2005). All communication uses CVML (List and Fisher, 2004).

The CAVIAR architecture is written in a combination of C++, Scheme and logical rules using Clips. It makes use of both the Imalab image processing library (Lux, 2004), Intel's OpenCV (Bradski, 2000) and the CoreLibrary (List and Fisher, 2004). The modules communicate with each other and the controller via a public API which can be used either directly in-memory, through files written to disk or across the network using TCP communication. Data content is transferred using CVML (List and Fisher, 2004).

Psyclone

Psyclone is a generic framework for AI, which includes support for multimodal perception of vision, audio, speech and other sensor input, as well as modular cognitive processing and multimodal output generation, such as speech and animated output.

Psyclone implements an open standard called OpenAIR[1] for both local and networked messaging and data transmission. It is a proven protocol which contains semantic and ontological specification of information.

The Psyclone framework consists of a number of information dispatchers called whiteboards (Thórisson et al., 2005b) and any number of modules. An underlying support system facilitates system setup and maintains basic system information about modules and dispatchers.

Each whiteboard functions as a publish/subscribe server to which information is posted and from which information is dispatched to modules which are subscribed to that particular type of information. A whiteboard will keep data for a period of time, stored in a large searchable database for later retrieval by modules needing past information. When new data is posted to the whiteboard it consults the subscriptions and will make sure that all subscribed modules receive the information they requested, including retrieval of any additional past or current data.

Like CAVIAR, each module has a full description of itself which is usually entered into the central configuration file, but can be manually overwritten by the module at any time. This description includes a list of public parameters, a list

[1] http://www.mindmakers.org

of subscriptions including data types and dispatchers, additional information to be retrieved along side the triggering data, and the output data types and destination dispatchers. It also includes information about the actual module code to run which is automatically started either from a shared library or an external executable.

To manage the runtime behaviour of modules Psyclone supplies the concept of contexts, which are globally announced system states. Each module can be assigned one or more such contexts and will not run without at least one of them being true.

The dataflow is regulated completely autonomously based on module priority and subscriptions in different contexts, as is the computations that each module performs. There is no overall global control monitoring every part of the system to make sure that the right modules perform adequately based on its inputs and outputs. However, local control modules can be created to monitor the performance of individual or groups of modules and make decisions about when to change parameters or even the dataflow by changing contexts. This way control can be achieved in a distributed way where control units can monitor and regulate anything from a single module to every module in the system, and they base their decisions on current and past data available from one or more Whiteboards.

The Psyclone architecture is written completely in C++ and runs on a number of operating systems including Unix, Windows and Macintosh. It is based on the CoreLibrary (List and Fisher, 2004) and supports the OpenAIR protocol for local or network communication and CVML (List and Fisher, 2004) for information content. Modules are created either as Cranks in C++ to run inside Psyclone or as Plugs in languages such as C++, Java and LISP to run outside Psyclone.

Application: Computer Vision

We applied the two architectures to the practical problem of using a single static camera to monitor human behaviour in public scenes such as streets or shopping centres, where a few tens of people can be in the scene at the same time.

The modular approach taken includes image acquisition, low-level image analysis for dense and sparse image features, tracking and detecting movement patterns of objects and finally analysing these to identify the objects' roles. We first defined these modules in the CAVIAR framework and the resulting dataflow can be seen in Fig. 1.

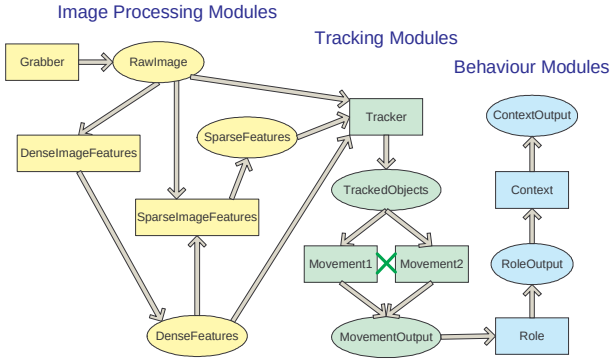

Fig. 1. The system data and processing flowchart from CAVIAR

In the top left corner the Grabber module outputs the RawImage dataset which contains an image acquired from the camera with a timestamp. The RawImage dataset is used by three modules, the DenseImageFeatures module, the SparseImageFeatures module and the Tracker module. The latter also uses the outputs from both the DenseImageFeatures and the SparseImageFeatures modules and based on all this data will output a list of TrackedObjects. There are now two Movement modules which both analyse the TrackedObjects for movement patterns and produce MovementOutput. The Role module uses this to assign roles to each TrackedObject and from these roles the Context module determines the context of the scene being monitored.

An example could be three people being tracked in a shopping centre. Each walks very slowly near shop windows, stop frequently and none of them interact with the others. The movement module will produce information on their slow and stopping movement, which the role module will assign to be a browsing behaviour. This becomes increasingly complex when more people are in the scene interacting with each other and for more information on this see (Bins et al, 2005).

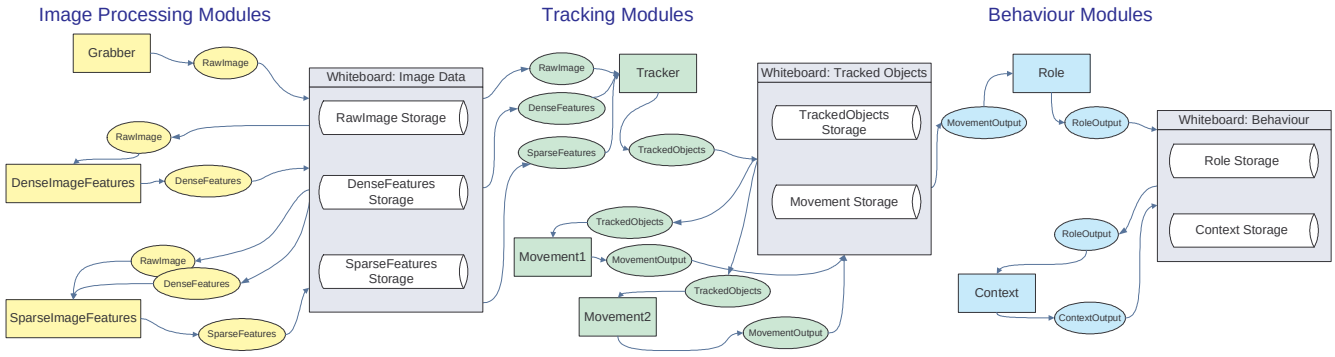

Fig. 2. The system data and processing flowchart from Psyclone. (Data structures are listed twice, once for input and once for output.)

The equivalent process and data flow produced by the Psyclone system is shown in Fig. 2. There are more elements because each data structure is listed once for every output and once for every input. This flow also includes the whiteboards from where the data is sent to the modules which have subscribed for this and stored for later retrieval.

Defining the modules

The overall idea behind both frameworks is to make it possible for people with topic-specific technical skills to implement modules in their areas of expertise into a system containing other people's work taken from a library of existing modules. Module implementation and testing should be quick without having to struggle with issues like network communication and dataflow.
In both CAVIAR and Psyclone modules are created using standard third-party libraries and can be developed and tested in the system directly. A module can be created with only a few lines of code and from there gradually increase the complexity while tested in the architecture using real data where other modules are using the output.
Equivalent modules or groups of modules can replace or compete with others to provide better or more confident results in a changing environment.
When creating a module in CAVIAR one can either choose an existing module from the project or create a new module from scratch. Either way, the object constructor code has to set the module description in XML which includes information about parameters and input and output datasets.

```
<description>
   <parameters count="2">
      <parameter name="MaxFeatures" type="integer" optional="no">
           <description>Maximum number of features</description>
           <range from="1" to="1000" step="5" />
           <default>100</default>
      </parameter>
      <parameter name="Sigma" type="float" optional="no">
           <description>Scale factor</description>
           <range from="0" to="1"/>
           <default>0</default>
      </parameter>
   </parameters>
   <dataflow>
      <inputs count="2">
           <input dataset="RawImage" />
           <input dataset="DenseFeatures" />
      </inputs>
      <outputs count="1">
           <output dataset="SparseFeatures">
              <variable name="Time" type="Time" />
              <variable name="FeatureVectorList" type="FeatureVectorList" />
           </output>
      </outputs>
   </dataflow>
</description>
```

Fig. 3: The CAVIAR description for the DenseImageFeatures module.

Fig. 3 shows the CAVIAR module description for the SparseImageFeatures module. It includes two public parameters, one called MaxFeatures which is used for setting a maximum number of features to output, and the other called Sigma, which is a scale factor. The module also specifies two input datasets, RawImage and DenseFeatures which it needs to compute its output dataset called SparseFeatures, containing the variables Time and FeatureVectorList.

Other than setting the description one needs to define a **`run()`** function which computes and stores the output dataset from the input data and parameters. Based on this information, the controller will include the module in the system dataflow after the Grabber and DenseImageFeatures modules and before the Tracker which needs the SparseFeatures dataset, as seen in Fig. 1.
When creating the same functionality in Psyclone one can define the module in the central XML configuration file. This definition is seen in Fig. 4 and like in CAVIAR it has two parameters, requires two inputs and produces one output.

```
<module name="SparseImageFeatures">
   <description>
      Produces sparse features from the raw image and dense image features
   </description>
   <parameter name="MaxFeatures" type="Int" max="1000" min="0" default="100" />
   <parameter name="Sigma" type="Double" max="1" min="0" default="0"/>
   <spec>
   <context name="Scene.Normal">
      <phase id="Process incoming messages">
         <triggers>
         <trigger from="ImageData" type="Input.Video.RawImage"/>
         <trigger from="ImageData" type="Features.DenseFeatures"/>
         </triggers>
         <cranks>
            <crank name="Image::SparseFeaturesCrank"/>
         </cranks>
         <posts>
         <post to="ImageData" type="Features.SparseFeatures" />
         <post to="ImageData" type="Psyclone.Context:Scene.Dark" />
         <post to="ImageData" type="Psyclone.Context:Scene.Bright" />
         </posts>
      </phase>
   </context>
   </spec>
</module>
```

Fig. 4. The Psyclone definition for the DenseImageFeatures module.

In addition, using the Psyclone context mechanism, this module can choose to change the system context if the scene is too dark or bright. Typically, vision systems would either use a module which could handle this; CAVIAR is able to swap out a module with another for cases like this. The global context system in Psyclone is a unique construct that provides for an even better solution to this problem.
The run() is specified as the function SparseFeaturesCrank in the library called Image, which contains the code to receive the input data and with the use of the parameters should produce the output dataset. The module's subscriptions results in the data flow seen in Fig. 2.

Plugging in an equivalent module

Initially we defined the systems with only one Movement module after which we decided that we wanted to add another module with a slightly different algorithm. In both architectures we renamed the Movement module Movement1 and added an identical module Movement2 which only varied from the original by the code it used to compute the output.
The CAVIAR system detected that this new module was equivalent with Movement1 as shown by the crossover lines between them in Fig. 1. The Psyclone system added this new module as another source of the output dataset MovementOutput which in turn is made available to the Role module in addition to the MovementOutput dataset produced by Movement1.
The main difference is that in CAVIAR the decision on whether to use the output from Movement1 or Movement2 is made by the controller based on the feedback of the

modules and information from Agents. In Psyclone both datasets are made available and it is up to the Role module to compare the two and decide which one to use. A detector module can also be created to switch on or off the appropriate Movement module or to provide additional information assisting the Role module in deciding which data to use.
In theory it is easy to create modules which do very simple isolated tasks, however, in the real world of research testing the functionality of a module during development can be difficult without proper insight into what the system is doing step by step. Although CAVIAR provides feedback mechanisms, having 'a look inside the running system' is often desirable and Psyclone provides this via a built-in monitoring system called psyProbe. With a standard web browser developers can monitor every activity in the system, from timestamped information on the whiteboards to specific content of individual messages. The running system can be temporarily halted by switching context and manual message posting can introduce new data. This proved to be an important functionality when developing the vision architecture.

The running system

When running the systems the CAVIAR controller starts by initializing all the modules and fully calculating the dataflow. It then runs each module in turn with the specific goal to maximise the availability of the datasets needed. After each module has completed the controller obtains the feedback with information about the quality and quantity of the output as well as the time and resources spent.
The Psyclone system starts by creating all the modules, but unlike CAVIAR it does not compute the dataflow as this is done dynamically as the system runs based on the global system contexts and module states. The Grabber acquires the first image from the camera, posts this to the Whiteboard ImageData which in turn triggers the DenseImageFeatures module and so on.
In CAVIAR the controller would compare the feedback from the two equivalent modules Movement1 and Movement2 and based on the quality report, as well as information from relevant Agents, choose which output to use. If the controller had access to offline learned information it would furthermore have been able to evaluate the quality statement from each module by adding its own confidence weight to the 'supposed' quality.
The same two modules in Psyclone both added their output to the TrackedObjects Whiteboard which in turn triggered the Role model. Because of the identical timestamps for the triggering datasets the Role model could see that the information was produced based on the same input, yet by different modules, and could use either or both for its own processing, without needing additional help to choose between them.

Results

We will now review the main differences between the two frameworks, from an architectural standpoint.
The main difference between the solution implemented in CAVIAR and Psyclone relates to the different way in which they enable global versus distributed control. CAVIAR completely relies on one global controller that knows everything about every module and consequently also has to deal with everything and every module. It has to know about modules individually, in groups and as a whole system and governing so many layers of information is quite a big task.
Psyclone uses completely distributed control; no global control mechanisms are provided as default. However, monitoring and control can be added to regulate any level of the system, from individual modules to the global state. The benefit is that one can construct a hierarchy of control where no individual controller has to know anything beyond the issues it is dealing with, much like (Crowley and Reignier, 2003), although they require a controller to be added for each module and each small group of modules in turn, making the Psyclone design simpler.
The drawback of a distributed control approach is that no single controller knows everything and it is therefore easy to lose sight of what is actually going on globally when trying to mend things locally. Psyclone imposes less structure on the design; increased flexibility puts a higher burden on the designer to ensure the soundness of the architecture.

Dynamical dataflow

The dataflow in both frameworks is automatically generated from the modules' XML auto-description. In CAVIAR the paths are initially created at startup and can be modified by the controller at any time, both to use an equivalent group of modules but also to change the flow completely if needed. The knowledge needed to decide about such changes is usually learned during the offline runs and consists of a combination of neural network decisions or logical rules. In other words, the controller will choose a change from a finite number of possible actions learned from past experience.
In Psyclone the change of dataflow can happen on many levels. Equivalent modules can, as in CAVIAR, compete with the quality of their results and decision modules can choose to activate or deactivate modules to change the flow of information. A Psyclone module mechanism allows a designer to put multiple named methods in each module and construct rules that determine complex conditions for using each method. This way a module can cycle through this set of methods at runtime, using whichever method is appropriate at the time. In effect it resembles a vast parameter change in CAVIAR, but in CAVIAR a single module would have to implement all the different algorithms in one mesh and use parameters to choose

between them. This would make the module implementation extremely complex and very hard to work with.

Any perception system will naturally go through many situations in which data processing will need to be done slightly or very differently from other situations. In CAVIAR each situation would activate a specific set of modules, probably different from the set used in other situations. When handling a large number of situations modelling and regulating the parameters of so many modules required could become unmanageable, especially if offline training is needed. The use of contexts in Psyclone solves this problem and adds another level of modular control. Contexts allow modules in Psyclone to radically change their behaviour based on the overall requirements of the system. A module could in one context be searching for one thing and in another doing something very different, using a similar or completely different algorithm. This mechanism can be used to reduce the number of separate modules in a system while still allowing the implementers to deal with disparate situations independently.

Handling the unexpected

Both resulting architectures are able to handle sudden changes to the perceived environment such as lighting changes, but they deal with these changes in very different ways.

The architecture implemented in CAVIAR uses agents to detect such changes and based on their output, changes in module parameter settings or even the global dataflow can be made. The CAVIAR controller is able to compensate for this faster if prior offline training included similar changes. The controller may then know which modules are more sensitive to these changes and which parameters to tune to stabilise the output from these. These decisions are made using either neural networks or learned logical rules. With no prior training the CAVIAR controller relies on the auto-regulation of each module to make adjustments for such situation, however, tuning them to work well under many varying conditions is very hard. This is one of the main reasons that knowledge-based systems have failed in the past (Draper, 2003).

Psyclone has no built-in support for offline learning, however one can create sets of control modules which modify parameters and monitor the results, in a kind of dynamic equilibrium. If going one way turns out to be wrong, decisions will be made to counter this rather than knowing how to solve the global problem as a whole. All modules, including control modules, have full access to all data in the system via the whiteboards, past and present (within reason) and can ultimately make use of contexts to get out of a situation that cannot be remedied by parameter tuning. Additionally, these modules could be configured for offline learning of the parameter spaces of individual or groups of modules and store this knowledge in another Psyclone construction called a *Catalog* for later online use.

Priorities

At runtime, different datasets may be needed at different times and with varying degrees of urgency. In CAVIAR a module will be activated when the data it needs is available; there is no notion of priority of data or processing. This means that when the system is stressed the more important data is treated with the same priority as analysing more subtle information.

In Psyclone each module has two priority settings, for data transmission and for runtime. The former determines when information is made available to the module by specifying the relative importance of a dataset compared to other sets, and the latter the relative processing priority of each module over others. This allows the system designer to prioritise certain information paths over others, creating a hierarchy of data significance, from the absolutely critical to the supernumerary while also taking into account the computing cost of producing these.

Ease of Use

We have discussed the main architectural differences; we will now look at some numbers regarding usage and runtime behaviour.

Table 1 shows the time it took to work with the basic and advanced building blocks of creating a modular system, from installing the example system, creating modules from the built-in pool of examples and new modules with own code, to working with control. The work was done by a single developer who knew the vision application intimately, as well as both frameworks. Basic control consisted of simple parameter tweaking and more advanced control included also changing the flow of data. Although the data is anecdotal for the two systems built, we believe it to be indicative of there being very little difference between the two frameworks in terms of usability, for someone who is experienced in the two systems.

Task	CAVIAR	Psyclone
Time to install the example system	3 hour	1 hour
Time to define a built-in module	1 hour	1 hour
Time to create a new module	2 hours	1 hour
Time to implement basic control	Built-in	1 hour
Time to implement advanced control	1 hours	2 hours

Table 1. Ease of use and developing new modules and control.

Runtime Performance

Table 2 shows the runtime performance characteristics computed for the two systems. Since the vision algoritms are completely identical in the two architectures the numbers expose the efficiency of the implementation of each framework, CAVIAR and Psyclone. We tested three modules performing low to medium image analysis tasks (two edge detectors and one optical flow); the numbers reflect the difference between the respective ways in each

framework of handling the flow of voluminous data and system control.

Task	CAVIAR	Psyclone
Average time to run all three modules	845 ms	65 ms
Same when changing 3 parameters	1212 ms	86 ms
Average time per parameter	122 ms	7 ms

Table 2. Runtime performance comparison.

The third row in Table 2 was calculated by taking the difference between the average runs with and without parameters and then dividing by three.

Lastly, we measured the time to run the whole system through 500 frames of video. This in CAVIAR included the system controller and in Psyclone three control modules to adjust the dataflow in case of changing conditions.

Task	CAVIAR	Psyclone
Time to run 500 frames	21 min	3.5 min
Frames per second	0.40 fps	2.38 fps

Table 3. Overall performance comparison.

Summary

We have found that it is very easy to use existing and to implement new vision modules in both the CAVIAR and Psyclone frameworks. The main difference between the two frameworks lies in the runtime control where CAVIAR chooses a global control approach and Psyclone a fully distributed control approach. Modules are in both frameworks created using ones own existing code and libraries, and then integrated into the system using an XML-based description of its parameters, the required inputs and the outputs produced.

Although modules in CAVIAR and Psyclone are created differently the basic principles are much the same and the central configuration, as well as the XML-based module description, makes it easy to test a large number of approaches to solving different scientific problems.

Each framework has a few features that the other doesn't; none of those features were essential for building the vision architecture described here.

Besides the solutions being different in the two architectures, and thus having implications for maintenance and extensions, the largest difference between the two is in the runtime efficiency inherent in the frameworks: Psyclone outperformed CAVIAR. Since Psyclone is intended for building real-time interactive systems and is available both in a commercial and a free research configuration, this is not too surprising. Psyclone also has extensive online documentation and support. This is likely to factor into the decision when choosing between the two frameworks. The CAVIAR website has a number of human-labelled video sequences available for download.

More information about the two architectures we refer the reader to the MINDMAKERS.ORG[3] and the CAVIAR[4] website, which includes a number of human-labeled video sequences available for download.

Acknowledgement

This research was supported by the CAVIAR project, funded by the EC's Information Society Technology's programme project IST 2001 37540. We thank the people supporting the Mindmakers.org efforts to create open architectures and standards for research in the areas of artificial intelligence.

References

R. Adler, *Blackboard Systems*, in S. C. Shapiro (ed.), The Encyclopedia of Artificial Intelligence, 2nd ed., 110-116. New York, NY: Wiley Interscience, 1989.

José Bins, Thor List, Robert B. Fisher and David Tweed, An Intelligent and Task-independent Controller for Video Sequences Analysis, accepted for publication to *IEEE CAMP05*, 2005.

Gary Bradski, The OpenCV Library, *Dr. Dobb's Journal* November 2000, Computer Security, 2000.

M. D. Brown and R. B. Fisher, A Distributed Blackboard System for Vision Applications, Proc. 1990 *British Machine Vision Association Conf.*, pp 163-168, Oxford, 1990.

A. A. Cohen, Addressing architecture for brain-like massively parallel computers, *Digital System Design*, pp 594 - 597, 2004.

J. L. Crowley, Integration and control of reactive visual processes, *Robotics and Autonomous Systems*, vol. 16, pp. 17-27, 1995.

J. L. Crowley and P. Reignier, Dynamic Composition of Process Federations for Context Aware Perception of Human Activity, *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, *KIMAS'03*, 2003.

B. Draper, From Knowledge Bases to Markov Models to PCA, *Workshop on Computer Vision System Control Architectures*, *VSCA'03*, 2003

R. B. Fisher and A. MacKirdy, Integrating iconic and structured matching, Proc. *5th Eur. Conf. on Computer Vision*, Vol. II, pp 687-698, Freiburg, Germany, June 1998.

A. R. Hanson and E. M. Riseman, The VISIONS Image-Understanding System, *Advances in Computer Vision* (Ed. C. Brown), pp 1-114, 1988.

Thor List, José Bins, Robert B. Fisher and David Tweed, A Plug-and-Play Architecture for Cognitive Video Stream Analysis, accepted for publication to *IEEE CAMP05*, 2005.

[3] `http://www.mindmakers.org`

[4] `http://homepages.inf.ed.ac.uk/rbf/CAVIAR`

Thor List and R. B. Fisher, Computer Vision Markup Language, *Proc. Int. Conf. on Pat. Rec.*, Cambridge, Vol 1, pp 789-792, 2004.

A. Lux, The Imalab Method for Vision Systems, *Machine Vision and Applications* Vol. 16 No. 1, pp 21-26, 2004.

T. Matsuyama and V. Hwang, *SIGMA: A Knowledge-Based Aerial Image Understanding System*, New York: Plenum, pp 277-296, 1990.

N. T. Siebel, Design and Implementation of People Tracking Algorithms for Visual Surveillance Applications, PhD thesis, Department of Computer Science, The University of Reading, Reading, UK, March 2003

K. R. Thórisson, A Mind Model for Communicative Creatures and Humanoids, *International Journal of Applied Artificial Intelligence*, 13(4-5), pp 449-486, 1999.

K. R. Thórisson, H. Benko, A. Arnold, D. Abramov, S. Maskey and A. Vaseekaran (2004). Constructionist Design Methodology for Interactive Intelligences. *A.I. Magazine*, Vol 25, Issue 4, pp 77-90. Menlo Park, CA: American Association for Artificial Intelligence

K. R. Thórisson, T. List, C. Pennock and J. DiPirro, Whiteboards: Scheduling Blackboards for Interactive Robots, accepted to the *AAAI-05 Workshop On Modular Construction of Human-Like Intelligence, AAAI-05*, Pittsburgh, PA, July 9-13, 2005a.

K. R. Thórisson, T. List, J. DiPirro and C. Pennock, A Framework for A.I. Integration, Reykjavik University Department of Computer Science Technical Report, RUTR-CS05001, 2005b.

K. R. Thórisson, C. Pennock, T. List and J. DiPirro, Artificial Intelligence in Computer Graphics: A Constructionist Approach, *Computer Graphics Quarterly*, Vol 38, Issue 1, pp 26-30, New York, February 2004.