A YARP-BASED ARCHITECTURAL FRAMEWORK FOR ROBOTIC VISION APPLICATIONS

Stefán Freyr Stefánsson, Björn Þór Jónsson
Database Lab and School of Computer Science, Reykjavík University, Kringlan 1, IS-103 Reykjavík, Iceland

Kristinn R. Thórisson
CADIA and School of Computer Science, Reykjavík University, Kringlan 1, IS-103 Reykjavík, Iceland

Keywords: YARP, Computer vision, Architecture, Performance evaluation.

Abstract: The complexity of advanced robot vision systems calls for an architectural framework with great flexibility with regards to sensory, hardware, processing, and communications requirements. We are currently developing a system that uses time-of-flight and a regular video stream for mobile robot vision applications. We present an architectural framework based on YARP, and evaluate its efficiency. Overall, we have found YARP to be easy to use, and our experiments show that the overhead is a reasonable tradeoff for the convenience.

1 INTRODUCTION

One of the most important sensory mechanisms for mobile robots is a sense of vision that robustly supports movement and manipulations in a three-dimensional world. Here, we use "vision" broadly to encompass any visuospatial sensory inputs and processing required for an understanding of the environment. Accumulated experience has shown, however, that for such robotic vision it is necessary to employ a number of sensors and processing mechanisms, integrated in various ways—often dynamically—to support realtime action in various contexts.

We are developing such a vision system, which will eventually employ a number of techniques, including (a) color video cameras, which provide shape and color information but do not easily give depth information, (b) time-of-flight cameras, which can yield sufficient depth information to create a depth map of the environment, (c) image descriptions, such as edge maps and SIFT descriptors (Lowe, 2004), which can be used for object recognition and obstacle detection, and (d) a communications infrastructure (Thórisson et al., 2007; Tweed et al., 2005; Thórisson et al., 2005) which allows basic real-time processing on the robot itself, while more advanced processing may take place on a dedicated off-board cluster.

In order to study the use and interactions of all

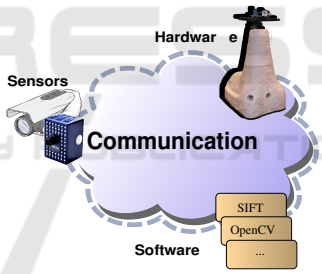

Figure 1: Architectural Requirements

these components it is clearly necessary to use an architectural framework which supports flexible manipulation of such compound, multimodal data, on diverse hardware platforms. Such a framework must allow for easy runtime configuration of the processing pipeline, while incurring limited overhead. Low-level options, such as shared memory and/or remote procedure calls, are not flexible enough, as they must be augmented with mechanisms for handling variable latency, priorities or other necessary features of complex architectures and soft-realtime response generation. What is needed is a higher-level framework that supports free selection of communication methods, including shared memory and TCP/IP, depending on the data and architectural constraints at any point in time. The architectural requirements are illustrated in Figure 1, which depicts four major categories of requirements: sensory aparati, other hardware, software

Freyr StefÃąnsson S., ÃđÃşr JÃşnsson B. and R. ThÃşrisson K.
A YARP-BASED ARCHITECTURAL FRAMEWORK FOR ROBOTIC VISION APPLICATIONS.
DOI: 10.5220/0001773600650068
In *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications (VISIGRAPP 2009)*, page
ISBN: 978-989-8111-69-2

and communication.

Several frameworks exist which partially address our needs, but very few address all of them. One of these frameworks is YARP (Yet Another Robot Platform), which is a set of libraries to cleanly decouple devices from software architecture (Metta et al., 2006; Fitzpatrick et al., 2008). It is an attempt to provide a foundation that makes robot software more stable and long-lasting, while allowing for frequent changes of sensors, actuators, processors and networks. As the authors themselves say: "YARP is written by and for researchers in robotics, particularly humanoid robotics, who find themselves with a complicated pile of hardware to control with an equally complicated pile of software" (Metta et al., 2006, p. 1).

We have constructed a preliminary vision system using YARP as the communication infrastructure. Overall, we have found YARP to be satisfactory and easy to use. Installing and learning to use YARP took about one man-week, while the rest of the time was spent on creating hardware drivers and working with the cameras.

In this paper we report on an effort to evaluate the use of YARP for our architecture, by exploring how well the platform supports our basic needs, such as for sequential processing in a pipeline architecture. An extended version of this paper may be found in (Stefánsson et al., 2008).

2 COMMUNICATION INFRASTRUCTURES

There are several potential candidates that can be chosen as underlying communication infrastructure for video data, including YARP (Metta et al., 2006; Fitzpatrick et al., 2008), OpenAIR (Thórisson et al., 2007), CAVIAR (List et al., 2005), Psyclone (Thórisson et al., 2005) and others. In the remainder of this section we first give a short description of YARP, and our reasons for evaluating it, and then briefly describe some of the alternatives.

2.1 YARP

YARP (Yet Another Robot Platform) is a set of libraries to decouple devices, processing, and communication. YARP provides loose coupling between sensors, processors, and actuators, thus supporting incremental architecture evolution. The processes implemented on top of YARP often lie relatively close to hardware devices; YARP does therefore not "take control" of the infrastructure but rather provides a set of simple abstractions for creating data paths.

A key concept in YARP is that of a communications "port". Processes can have zero, one or more input ports, and produce output on zero, one or more output ports. Ports are also not restricted to a single producer or receiver—many producers can feed a single port, and many receivers can read from a single port. To keep track of ports, YARP requires a special registry server running on the network. The data communicated over the ports may consist of arbitrary data structures, as long as the producer and receiver agree on the format. YARP provides some facility to translate common datatypes between hardware architectures and such translation can be easily implemented in user defined datatypes as well. Each port may be communicated via a host of transport mechanisms, including shared memory, TCP/IP and network multicasting. YARP is thus a fairly flexible communication protocol that leaves the programmer in control.

Our main reason for for evaluating YARP is the fact that it is unobtrusive and basic. Other reasons include the following:

- YARP abstracts the transport mechanism from the software components, allowing any software component to run on any machine. It supports shared memory for local communication, and TCP/IP, UDP, and multicast for communication over a network.
- YARP interacts well with C/C++ code, which is required for our time-of-flight camera. YARP can be used with several other languages as well.
- YARP can communicate any data structure as long as both receiver and sender agree on the format. Furthermore, it provides good built-in support for various image processing tasks and the OpenCV library.
- It is open-source software. As we wish to make our framework freely available, the communication infrastructure must also be freely available (indeed, we have already sent in a few patches for YARP, including camera drivers and utilities).
- Finally, although this was by no means obvious from any documentation, the support given by YARP developers has been both very responsive and useful.

These requirements are undoubtedly also met by alternative frameworks and libraries; we have not yet made any formal attempt to compare YARP to these other potential approaches. With a host of tradeoffs the choice of low-level or mid-level middleware/libraries can be quite complex, and we leave it for future work to compare YARP in more detail to the approaches described next.

2.2 Alternative Architectures

OpenAIR is "a routing and communication protocol based on a publish-subscribe architecture" (Thórisson et al., 2007). It is intended to help AI researchers develop large architectures and share code more effectively. Unlike YARP, it is based around a blackboard information exchange and optimized for publish-subscribe scenarios. It has thoroughly defined message semantics and has been used in several projects, including agent-based simulations (Thórisson et al., 2005) and robotics (Ng-Thow-Hing et al., 2007). OpenAIR has been implemented for C++, Java and C#.

CAVIAR (Tweed et al., 2005) is a system based on one global controller and a number of modules for information processing, especially geared for computer vision, providing mechanisms for self-describing module parameters, inputs and outputs, going well beyond the standard services provided by YARP and OpenAIR. The implementation contains a base module with common functionalities (interface to controller and parameter management).

Psyclone (see www.cmlabs.com) is an AI "operating system" that incorporates the OpenAIR specification. It is quite a bit higher-level than both OpenAIR and YARP and provides a number of services for distributed process management and development. Psyclone was compared to CAVIAR by List et al. (List et al., 2005) as a platform for computer vision. Like CAVIAR, Psyclone has mechanisms for self-describing semantics of modules and message passing. Unlike CAVIAR, however, Psyclone does not need to pre-compute the dataflow beforehand but rather manages it dynamically at runtime, optimizing based on priorities of messages and modules. Both CAVIAR and Psyclone are overkill for the relatively basic architecture we intend to accomplish at present, at least in the short term, but it is possible that with greater expansion and more architectural complexity, platforms such as Psyclone would become relevant, perhaps even necessary.

Compared to, e.g., CAVIAR and Psyclone, YARP looks like a fairly standard library—neither does it do its own message scheduling nor does it provide heavy-handed semantics for message definitions or networking. That may be its very strength.

3 EXPERIMENTAL EVALUATION

In this section, we report on an initial performance study of the YARP transport mechanisms. Further experiments are reported in (Stefánsson et al., 2008).

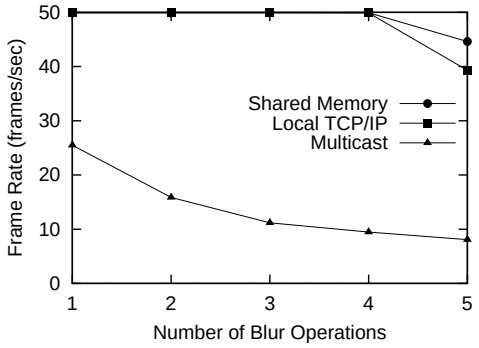

Figure 2: Frame rate at receiver.

3.1 Experimental Setup

In this study we focus on single processor configurations. At present, the goal is thus not to study the scalability of the system, but rather to compare some configuration choices of YARP for vision.

We set up a basic processing pipeline, meant to represent a typical setup, which consists of 1) a producer, which produces 320x240 pixel image frames at a given frame rate; 2) a number of blur operators, which run the "simple" OpenCV blur algorithm over the frames; and 3) a receiver, which receives the frames. We change the processing pipeline length, or the number of blur operators, to study the effects of overloading the computer.

Each frame is augmented by sequence numbers and time stamps by each of these components, which are then used to measure dropped frames and latency, respectively. Other metrics collected include the frame rate observed by the receiver (lower frame rate occurs when frames are dropped) and CPU load.

Our experimental setup runs on a 2.6GHz Pentium 4 Dell OptiPlex GX270 computer with 1.2Gb RAM.

3.2 Transport Mechanism Performance

In this experiment, the frame rate of the producer was set to 50 frames per second, which is similar to a high-quality video stream. The length of the processing pipeline was varied from one to five consequtive blur operators. We ran measurements using shared memory, local TCP/IP and network multicast connections, with the expectation that shared memory should be fastest. For each configuration, the experiment was run until the receiver had received 50,000 frames.

Figure 2 shows the frame rate observed by the receiver. The *x*-axis shows the length of the processing pipeline. Overall, two effects are visible in the figure. First, using local TCP/IP and shared memory maintains a frame rate of 50 frames per second, until

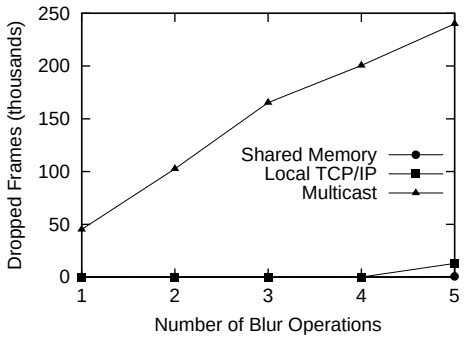

Figure 3: Frame drops in pipeline.

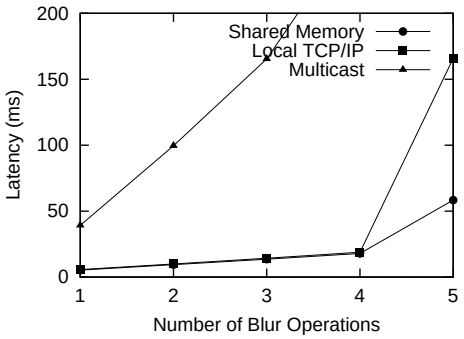

Figure 4: Latency of received frames.

the pipeline consists of more than four blur processes. At that point, the processor is overloaded and frames are dropped as a result, leading to lower frame rates observed by the receiver. Shared memory performs slightly better due to lower communication overhead.

Second, turning to the performance of multicast, Figure 2 shows that the processing pipeline achieves a much lower frame rate, ranging from 25 to 8 frames per second. The reason for the lower frame rate is clearly visible in Figure 3, which shows the number of frames that are dropped for each configuration. As Figure 3 shows, even with only one blur operator, every other frame is dropped with the multicast mechanism. The frame rate observed by the receiver is thus only half the frame rate of the producer. As more blur operators are added, more frames are dropped, explaining the lower frame rates seen in Figure 2.

Turning to latency, Figure 4 shows that, as expected, latency of the multicast transport mechanism is very high and constantly increasing with pipeline length as frames can be dropped anywhere in the pipeline. For the other two transport mechanisms, latency is relatively low until the pipeline consists of five blur operators. At that point, the CPU is saturated and scheduling conflicts occur. Again, latency is significantly lower using shared memory than TCP/IP due to the lower communication overhead.

Further experiments have shown that the typical overhead of YARP communcations will be about 50%, which is a reasonable overhead for the convenience of using YARP (Stefánsson et al., 2008).

4 CONCLUSIONS

We have described our efforts towards a flexible computer vision infrastructure based on the YARP toolkit. We have found YARP easy to use, as it greatly simplifies making the infrastructure flexible towards sensors, hardware, processing, and communication requirements, compared to starting from scratch. Our experiments show that the overhead is a reasonable tradeoff for the convenience.

REFERENCES

Fitzpatrick, P., Metta, G., and Natale, L. (2008). Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45.

List, T., Bins, J., Fisher, R. B., Tweed, D., and Thórisson, K. R. (2005). Two approaches to a plug-and-play vision architecture - CAVIAR and Psyclone. In *Workshop on Modular Construction of Human-Like Intelligence*, Pittsburgh, PA, USA.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: Yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1):43–48.

Ng-Thow-Hing, V., List, T., Thórisson, K. R., Lim, J., and Wormer, J. (2007). Design and evaluation of communication middleware in a distributed humanoid robot architecture. In *Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, San Diego, CA, USA.

Stefánsson, S. F., Jónsson, B. Þ., and Thórisson, K. R. (2008). Evaluation of a YARP-based architectural framework for robotic vision applications. Technical Report RUTR-CS08004, Reykjavík University School of Computer Science.

Thórisson, K. R., List, T., Pennock, C., and DiPirro, J. (2005). Whiteboards: Scheduling blackboards for semantic routing of messages & streams. In *Workshop on Modular Construction of Human-Like Intelligence*, Pittsburgh, PA, USA.

Thórisson, K. R., List, T., Pennock, C., and DiPirro, J. (2007). OpenAIR 1.0 specification. Technical Report RUTR-CS07005, Reykjavík University School of Computer Science.

Tweed, D., Fang, W., Fisher, R., Bins, J., and List, T. (2005). Exploring techniques for behaviour recognition via the CAVIAR modular vision framework. In *Proc. HAREM Workshop*, Oxford, England.