# Design and Evaluation of Communication Middleware in a Distributed Humanoid Robot Architecture

Victor Ng-Thow-Hing,[1] Thor List,[2] Kristinn R. Thórisson,[3] Jongwoo Lim,[1] Joel Wormer[1]

[1]Honda Research Institute, 800 California St., Suite 300, Mountain View, CA, 94041, USA, {vng,jlim,jwormer}@honda-ri.com
[2]Communicative Machines Inc., 455 W22nd St., Suite 3, New York, NY 10011, list@cmlabs.com
[3]CADIA, Reykjavik University, Kringlunni 1, 103 Reykjavik, Iceland. thorisson@ru.is

***Abstract*—Distributed architectures for implementing tasks on humanoid robots is a design challenge, both in theory and practice. Although important functionality resides within the component modules of the system, the performance of the middleware – the software for mediating information between modules – is critical to overall system performance. We have designed an architecture serving various functional roles and information exchange within a distributed system, using three different communication subsystems: the Cognitive Map (CogMap), Distributed Operation via Discrete Events (DiODE), and Multimodal Communication (MC). The CogMap is implemented in Psyclone, a framework for constructing large AI systems, and allows sharing and transformation of information streams dynamically between modules. DiODE provides a direct connection between two modules while MC implements a multi-modal server that streams raw sensory data to requesting external (off-board) perceptual modules. These have been implemented and tested on the Honda Motor Corporation's ASIMO humanoid robot. To identify trade-offs and understand performance limitations in robots with distributed system architectures, we performed a variety of tests on these subsystems under different network conditions, operating systems and computational loads. The results indicate that delays due to our middleware is negligible compared to computational costs associated with actual processing within the modules, provided a network with high enough bandwidth. The Cognitive Map appears to be scalable to an increasing number of connected modules with negligible degradation of package delays.**

## I. Introduction

IN the pursuit of general-purpose robots, adaptive to a variety of environments, many technologies and software algorithms must be integrated to create a coherent robotic system capable of using different skills in the completion of a task. For example, sensor measurements are used as inputs to perceptual modules that extract useful features to reconstruct environmental information such as object pose and identification. This environmental state must be accessible to other modules in the system that in turn can create new knowledge of the environment. Modules are also needed for planning, decision making and sending final motor commands to the robot's body. In the course of robot execution of the task, frequent feedback of the environmental state is needed to robustly complete the task and to handle uncertainty or sudden changes in the environment.

We have found that a single architectural design does not support all tasks equally well. Instead, architectural components and choice of communication protocols should be reconfigurable at runtime, according to the demands of the task at hand. For example, for a robot pointing and gazing at a moving object (Figure 1), constant sensory feedback of the object position must be provided to the pointing command to update the robot's end effector position. In contrast, if a static environment is assumed, traditional sense-plan-act [1] pipelines can be employed, provided the sensors are sufficiently accurate for a task like pushing objects on a table (Figure 2).

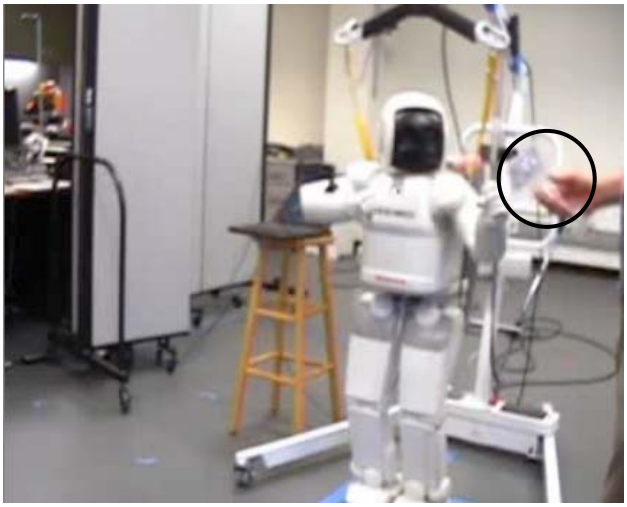

**Fig. 1: ASIMO pointing and gazing at a moving hand-held object (circled).**

The computational capabilities on board a robot are usually not as powerful as the latest modern desktop computers due to restrictions in energy consumption and heat management. To facilitate easy reconfiguration of the modules and communication between them, we employ a distributed system design where modules can reside externally from a robot's own control and actuator hardware. The advantage of a distributed system is that the

Manuscript received September 13, 2007. This work was supported in part by the Honda Research Institute USA.

computational demands of the task can be shared across disparate and specialized hardware and performance and functionality can be extended without significant changes to the original robot's design.


**Fig. 2: ASIMO pushing a block on a table.**

The computational capabilities on board a robot are usually not as powerful as the latest modern desktop computers due to restrictions in energy consumption and heat management. To facilitate easy reconfiguration of the modules and communication between them, we employ a distributed system design where modules can reside externally from a robot's own control and actuator hardware. The advantage of a distributed system is that the computational demands of the task can be shared across disparate and specialized hardware and performance and functionality can be extended without significant changes to the original robot's design. In our system each module can also be implemented in different software environments, operating systems and programming languages, maximizing the flexibility of solutions for researching and creating system components. This has the added benefit that teams are also free to work at different geographic locations with specialized skill sets appropriate to each subset of modules.

We have designed a distributed system architecture for Honda Motor Corporation's ASIMO humanoid robot [2]. The system allows experimental features to be added and expanded while incrementally modeling new tasks on the robot in its operating environment. To facilitate communication between modules, standardized abstraction interfaces were created for accessing motor commands and sensor information between modules. This approach has allowed us to design system components for managing tasks and environmental state that are independent of particular robot hardware configurations.

However, the components and their interface design are only two of many parts that shape the overall system performance of the robot. This paper will focus on the evaluation of a third part – the middleware: the collection of communication software and protocols for exchanging information between software modules and hardware components of the system. Delays in the channels of communication between modules can create bottlenecks in information flow which can cause adverse effects on performance, both in the receiving module and overall system performance. The achievable throughput in the various communication subsystems play an important role in determining where modules should reside (same or different machines) and at what rate they should operate.

In this paper we will introduce several different communication schemes that we have developed for various functionalities and examine the throughput performance of data samples transferred under several situations encountered in the operation of our robot architecture.

## II. APPROACH

For ease of integration, our approach takes into account the traditional pipeline of building components in a research environment. Typically, robot developers and researchers build specialized components in isolation of the main system. They perform tests to verify the performance and functionality of these components. Often, assumptions about interactivity with other modules are implicit, underestimated or overlooked – the act of integration itself is given lower priority or is often assigned to someone other than the original component maker. It is also often assumed that once the component is finalized and robust, the delivery of the component can be handed off to an integration team to incorporate into the system. In reality substantial iteration is required after initial integration is achieved. Unforeseen side-effects, e.g., changes in lighting conditions, motion blurring due to a moving camera, or incorrect assumptions about the quality of inputs to the module, can significantly reduce the robustness of its performance once the module is running within the rest of the system. All such architectural design requires frequent changes to the original (isolated) module software to improve operational performance.

To minimize the necessary code changes and make modules ready for integration, we employ a framework based on the Constructionist Design Methodology (CDM)[3]. CDM was developed to create systems capable of a large number of functionalities that must be carefully coordinated to achieve coherent system behavior. CDM is based on iterative design steps that lead to the creation of a network of named interacting modules, communicating via explicitly typed streams and discrete messages.

Our communication middleware consists of three specific subsystems: the Cognitive Map (CogMap), Distributed Operation via Discrete Events (DiODE) and Multimodal Communication (MC), each created for particular communication needs in our system. While the Cognitive Map is used for sharing and custom selection of information between many modules using a centralized structure, DiODE is used for more dedicated direct communication and MC is designed for networked sharing of sensor

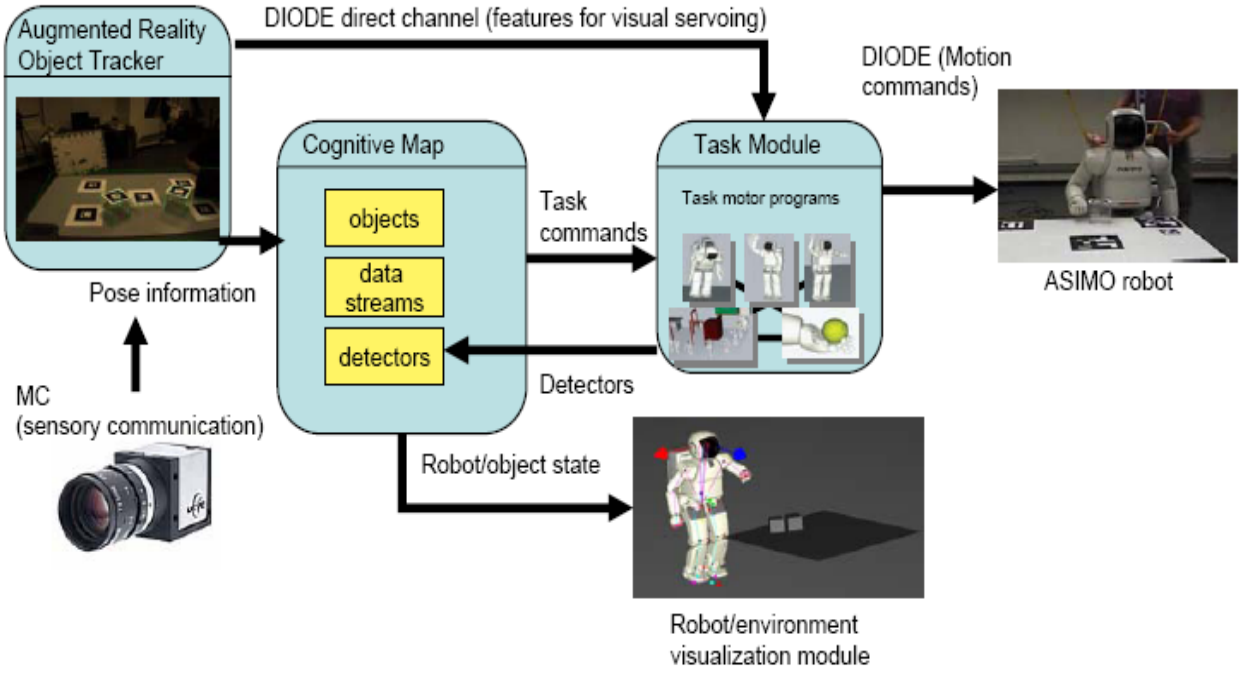


**Fig. 3: Sample representation of modules and types of communication.**

information, especially from optical cameras (Figure 3).

### *A. Shared, Selective Communication*

We have designed a blackboard-based cognitive map for manipulating perceptual information gathered from cameras and other sensors for the purposes of organizing information that represents both the robot's internal state and its external environment. As knowledge must be shared in the architecture to avoid redundant information and computation, and it must be combined from different sources to accomplishing tasks like localization and dynamic world state modeling, a blackboard model is an ideal framework for integration [6]. The development of distributed, large architectures, however, raises issues of dynamic access to shared data of distributed processes, transmission delays, and fluctuating resource availability and computational load [4].

The Cognitive Map (CogMap) is a system of interacting software elements, or modules, that together have the goal of enabling a mobile robot to perceive objects and actions, use them to compose plans for operating on the world and then monitor the execution of those plans via their effects on the environment. The CogMap itself maintains the current state of the world, the state of the robot's plans as well as being the storage and access point for all non-transient data in the system. Most of the robot's sensor and control systems interact via the CogMap.

To manage shared information, the CogMap is built on the Psyclone "Whiteboard" system [5] which combines the shared information concepts of a blackboard architecture [6] with data streams that can be shared, have their data samples timestamped for synchronization, and data content transformed (e.g. coordinate conversion) or selectively screened while being transmitted between modules

Objects within the CogMap represent physical objects identified from sensory input or conceptual objects generated by the modules' algorithms (e.g., observed actions). Physical objects can have 3-D pose and geometric information. They can be symbolically labeled with an object type if it can be identified. Objects of a specific type can also have custom fields associated with them. For example, a table object has its length and width parameters for the tabletop in order to allow reconstruction of its geometry from a relatively small set of parameters.

Modules of various types – Indexers, Deciders, Detectors - can interact with the Cognitive Map in various ways:

*Indexers* provide different ways to access stored data coming into the CogMap via streams. By default, samples on a stream are accessed by timestamps and frame count. Indexers can provide other search criteria for accessing an object, such as proximity to a specific point in space.

*Detectors* are instantiated dynamically by the CogMap upon requests from other system components, to set up tests for world conditions and events based on data from one or more Indexers. Typically, the creation of a Detector automatically initiates the creation of an Indexer.

A Detector implements a Boolean function that can produce true or false answers to specific questions about the data in a stream. One Indexer is created to facilitate this process upon the creation of a Detector. The criteria for a detector can originate from a module and be dynamically specified at run-time. In our system, detectors can be specified using pre-set Boolean operators written in C or be entirely scripted and interpreted at run-time using the Lua language [7]. All samples coming through a stream monitored by the Detector are evaluated and allowed to pass if the Boolean expression has a true value.

*Deciders* subscribe to information from two or more Detectors, or other Deciders, and make decisions on how to respond to events. Deciders can be created by the CogMap upon request by any module in the system.

Modules can either reside externally from the CogMap, using TCP/IP socket-based communication, or be dynamically-loaded internal modules, communicating directly through memory. Modules are free to dynamically subscribe and unsubscribe to information from the CogMap, minimizing the bandwidth of the communications channel. Information can be in the form of discrete messages or continuous streamed data. For example, Detectors subscribe to data streams from the Indexers. The Detectors can then publish information on those streams which can be read and utilized by Deciders.

### B. Direct Robust Communication

For situations where the data does not need to be stored, shared or transformed while in transit, the DiODE (Distributed Operation Via Discrete Events) communication subsystem was created to provide robust open communications to allow applications to share information asynchronously. DiODE is used in situations where the overhead of sharing and processing streams through a centralized entity like the CogMap is not desired. For example, visual servoing applications require a tight feedback loop between the perceptual modules and the task modules to ensure timely environment updates. The key features of DiODE are:

1. *Location independence:* Modules are unaware of the network location of peer modules. All communication is performed via a publish/subscribe system over named channels.
2. *Location optimization:* Low-level connections can be automatically optimized to use the most efficient transmission mechanism, based on location. (e.g., use shared memory instead of TCP for channel peers running on the same machine.)
3. *Role based optimization:* A module subscribes in a specific role ("CLIENT", "SERVER", "PEER", etc.) to each channel of interest. DiODE applies knowledge of complementary subscribed roles to open the optimal number of connections between peers. (For example, modules subscribed to a channel in the "CLIENT" role connect to the application in the "SERVER" role, but not to other "CLIENTS").
4. *Robustness:* DiODE is tolerant of disconnection and reconnection of subscribed modules. All modules in complementary roles receive notification events when peers join or leave a channel. The underlying connections are automatically established/re-established as needed. (E.g., a service can be shutdown on a malfunctioning host system and restarted on another host system, and all its clients will be automatically reconnected and notified.)
5. *Quality of Service:* The profile of each channel can be pre-defined to provide some required quality-of-service. (e.g., lossy for higher throughput, guaranteed delivery, total ordering, etc.)

### C. Multimodal Sensor Communication

Not all components of a system can be easily distributed. Sensors are constrained to be attached to particular locations on the robot and therefore must be physically close to the computational module that will process the raw data. On the other hand, this data is often further filtered and processed by perceptual modules to obtain important features necessary to estimate environmental state. Since many advanced computer vision algorithms can be expensive computationally, running several different modules on the same computer can be prohibitive. The Multimodal Communication (MC) system was created as a server for sensor data so that it can be redistributed to remote clients for distributed processing. Multiple sensors of different modalities can have their output streamed together to produce hybrid multimodal channels of information.

## III. COMPARATIVE EVALUATION

Test modules were created and connected to each other using the three communication subsystems described in Section II, Various scenarios were selected to assess the efficiency of the various data transmission patterns that can occur in the running system. The modules used in the tests were selected to be representative of the module setup in the implemented architecture. In each performance test, effort was made to isolate specific sources of delay and highlight trade-offs that would need to be made between system performance and flexibility of module location within the system. We also tested the effect of different operating systems (Windows and Linux), number of connected modules, and different networking bandwidth.

### A. Cognitive Map tests

The following tests measured the package delay of samples as they passed through the CogMap under different conditions. The test machines consisted of an AMD Dual Opteron 246 with 2GB RAM (AMD) and an Intel Dual Core 2 with 2GB RAM (INTEL). Different network bandwidth connections were also tested: direct memory with internal

CogMap modules (Direct), the local IP stack only (Localhost), Gigabit Ethernet (1 Gbit/s), Fast Ethernet (100 Mbit/s) and WiFi 802.11g (54 Mbit/s). All tests were run 1000 times to produce the data reported.

**Test 1: Media communication**

Synthetic samples (data package size: 10 kbytes) consisting of CogMap objects with position and orientation information were sent over several types of network connections.

**Table 1: Media communication**

| Network connection | Package Delay in msec | | | |
|---|---|---|---|---|
| | Average | Min | Max | STD |
| Direct[1] | 0.6 | 0.1 | 5.2 | 3.5 |
| Localhost[1] | 1.5 | 0.9 | 13.21 | 6.24 |
| 1Gbit/s[1,2] | 5.0 | 2.4 | 114.2 | 7.87 |
| 100Mbit/s[1,2] | 9.6 | 9.0 | 110.0 | 15.0 |
| WiFi 802.11g[1,2] | 11.9 | 10.3 | 98.0 | 30.6 |

Tests were run on the following computers (see superscripts): 1) AMD with Windows XP and 2) INTEL with Windows XP. As expected, Table 1 indicates that package delays clearly increase with lower bandwidth rates, but not in a linear relationship. Knowing the average frame rate allows the system designer to decide where to place modules within the system topology depending on their desired throughput needs.

**Test 2: Operating System Dependency**

Data packages of 10 kbytes each are streamed to modules residing on a single AMD machine (Direct and Localhost conditions) running different operating systems: Windows XP (Win) and Linux.

**Table 2: Operating system dependency**

| | Package Delay in msec | | | |
|---|---|---|---|---|
| | Average | Min | Max | STD |
| Direct Win | 0.6 | 0.1 | 5.2 | 3.5 |
| Direct Linux | 0.4 | 0.1 | 9.0 | 0.9 |
| Localhost Win | 1.5 | 0.9 | 13.21 | 6.24 |
| Localhost Linux | 1.1 | 0.6 | 21.1 | 2.1 |

Table 2 shows there is no notable difference in average package delays between Windows XP and Linux. However, package delays under Windows tend to have higher standard deviations due to extra overhead in the windowing system and services.

**Test 3: Stress Dependency**

Since the CogMap can share streams among many clients, it is important to determine how quickly performance degrades as more clients (recipients) access the same stream. 10 kbytes data packages are used on a single AMD machine. All clients reside on the same machine to eliminate network uncertainty.

**Table 3: Stress dependency**

| #recipients[1] | Package Delay in msec | | | |
|---|---|---|---|---|
| | Average | Min | Max | STD |
| 1 | 0.6 | 0.1 | 5.2 | 3.5 |
| 2 | 0.6 | 0.1 | 5.3 | 3.5 |
| 3 | 0.6 | 0.1 | 5.1 | 3.2 |
| 5 | 0.8 | 0.1 | 8.2 | 4.1 |
| 10 | 0.8 | 0.1 | 10.3 | 5.8 |

The CogMap was able to maintain a high level of performance with negligible degradation as the number of connecting modules increased from 1 to 10.

**Test 4: CogMap Detectors**

Three different CogMap Detectors were implemented: C-based Detector (C-based), a run-time interpreted Lua-based Detector (Lua-based), and a local C-based Detector at the receiving module (C-local). The last condition examines the trade-offs of performing additional processing of all samples at the receiving module versus receiving fewer samples due to filtering done within the CogMap. Timings are for 5 receiving modules accessing the same stream with data packages approximately 1 kByte in size on a single AMD machine.

**Table 4: CogMap detectors**

| | Package Delay in msec | | | |
|---|---|---|---|---|
| | Avg. | Min | Max | STD |
| C-based Direct | 11.1 | 0.5 | 51.6 | 15.9 |
| Lua-based Direct | 14.0 | 1.4 | 52.0 | 15.6 |
| C-local Direct | 20.0 | 1.7 | 78.2 | 16.7 |
| C-based Localhost | 13.5 | 2.7 | 46.0 | 11.9 |
| Lua-based Localhost | 18.7 | 3.2 | 78.0 | 18.7 |
| C-local Localhost | 22.4 | 3.3 | 88.5 | 22.8 |

The use of the more flexible, interpreted Lua-based detectors did not result in significant degradation of performance compared to the C-based detectors. This is due to the byte-code compilation of Lua. Interpreting the Lua script and converting it to byte-code only needs to be done once when the detector script code is initially delivered to the CogMap from a module. Subsequent function calls are performed on the byte-code compilation of the Lua-based detector.

Table 4 also indicates that using the detectors to filter out samples in the CogMap reduces package delay compared to processing all samples at the receiving module (C-local conditions). This confirms the utility of the CogMap's sample selection mechanism.

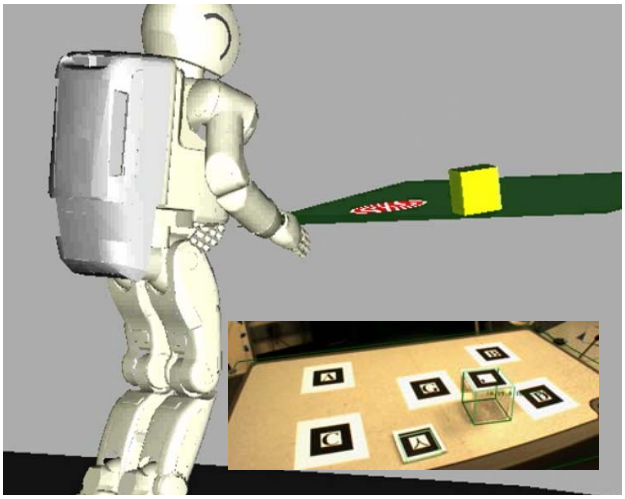

**Fig. 4: Visualization module recovering environment state from augmented reality vision module. Inset: Camera view from ASIMO of scene.**

### *B. Cognitive Map versus DiODE tests*

Comparison tests were performed to evaluate the streaming performance between the Cognitive Map and DiODE. Since the Cognitive Map provides a centralized intermediary for further processing of data, we expected DiODE to outperform the CogMap. The results will later show that this was not universally true. The CogMap provides several flexible and dynamic features, like the use of Detectors to filter out samples in transit. The need for these features may outweigh the small degradation of performance that may be experienced.

One Detector module was created to detect and report object pose (position and orientation) while a Decider module read the information for further processing (3-D visualization of the environment as seen in Figure 4). Transmission of this information was implemented with DiODE and the CogMap. In order to isolate the potential sources of delay, several different configurations were tested. To isolate the delays due to front-end image processing, the Detector module was configured to process live video frames using the ARToolKitPlus augmented-reality based detector [8] (FRONT) or using synthesized, randomized object pose information (NOFRONT). For back-end processing of received data samples, the Decider module can have 3-D visualization of the object pose turned on (BACK) or 3-D visualization turned off (NOBACK). To maximize throughput of the samples, the Decider module processed the samples in a separate thread under both the BACK and NOBACK conditions. Finally, the CogMap and DiODE test cases were run with both modules existing on the same machine (SAME) and on two different machines (DIFF) to isolate network delays. The computers used were a IBM Thinkpad Laptop (2.13 GHz Pentium M, 787MHz 1 GB RAM) (THINKPAD) and a IBM Desktop (3.06 GHz Intel Xeon, 3.07 GHz 2GB RAM) (DESKTOP).

Tables 5 and 6 feature two average sample processing times for each test configuration, one for the Detector module side and the other for the Decider module side.

**Table 5: Average sample processing times for SAME on THINKPAD**

| SAME configuration | DiODE (in msec) | CogMap (in msec) |
|---|---|---|
| NOFRONT, NOBACK | 15.51/15.52 | 15.51/15.47 |
| NOFRONT, BACK | 15.51/15.52 | 15.51/15.41 |
| FRONT, NOBACK | 139.37/139.38 | 97.30/97.50 |
| FRONT, BACK | 172.94/172.98 | 103.26/113.67 |

**Table 6: Average sample processing times for DIFF with FRONT side on DESKTOP, BACK on THINKPAD.**

| DIFF configuration | DiODE | CogMap |
|---|---|---|
| NOFRONT, NOBACK | 15.51/15.36 | 15.49/15.47 |
| NOFRONT, BACK | 15.45/15.34 | 15.53/15.51 |
| FRONT, NOBACK | 93.00/92.94 | 99.46/99.44 |
| FRONT, BACK | 92.86/92.55 | 102.47/102.97 |

Comparing the SAME and DIFF configurations indicates that the main processing bottleneck is the computation required for image processing with the actual delays due to the Cognitive Map being negligible. This is indicated by the similar sample processing times for both modules. If there were significant network delays, the Decider processing times would be longer.

For the SAME configuration, we were initially puzzled by DiODE's significant inferior performance compared to the CogMap, especially given that DiODE outperformed the CogMap in the DIFF configuration tests. We hypothesized that DiODE's implementation may be CPU-intensive because CPU utilization was observed to be high on the THINKPAD. We reran the FRONT/NOBACK test on the faster DESKTOP computer for the SAME configuration and indeed observed that DiODE regained its performance advantage (93.32/93.41 msec versus 101.403/100.28 msec for the CogMap).

In the DIFF tests with the CogMap, we also tried running the CogMap system on a third machine, different from those of the two modules. Our initial expectation was

to observe slower performance compared to DiODE due to the extra network links to the third machine. Instead, we were surprised to see slightly faster performance than when the CogMap was residing on the same machine as one of the modules (average 103.26/113.67 msec for CogMap on the same machine versus 97.64/98.18 msec for CogMap on a third machine). A possible explanation is that by putting the CogMap and modules on the same machine, their combined computational demands reduced the overall resources on the computer. This further supports the advantages of building a distributed system where loads can be balanced among several machines.

### C. *Multimodal Communication tests*

The MC subsystem was evaluated for the effectiveness of streaming out video to multiple connected clients (numbering 1-10). Video is captured to a local buffer and is subsequently streamed out to connecting clients over different network conditions. The speed of buffer capture depends on the source of the data. We tested two conditions where video is read from a file (RAWFILE) and from a live firewire IEEE 1394 camera (1394). In the former RAWFILE case, we were able to test the fastest frame rate possible, bounded only by computational resources. For camera hardware, frame rates are usually fixed (e.g., 30, and 60 fps). Finally, we ran tests comparing Localhost with Gigabit Ethernet (1000 Mbits/sec) and 802.11g wireless. The latter condition is important to consider as we typically would like the robot to move around unencumbered by an umbilical bundle of wired connections. The following two graphs plot the average frame rate from a group of 6 trials under each condition versus the number of connected clients receiving streams from the MC subsystem from a single video server. All tests were run on a quad-core 3GHz Intel/Xeon Mac Pro.

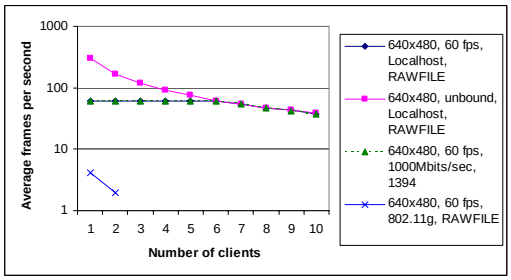


**Fig. 5: Comparison over different network bandwidth conditions (logarithmic scale)**

The first graph (Figure 5) shows several interesting trends. The 1000 Mbits/s trial performance is near-identical to the Localhost trial. The degradation of frame rate occurs at the same time and rate after 6 clients. Looking at the unbound versus 60 fps-bounded conditions, the computer is able to keep up for the first 6 clients due to its excess computational capacity (as seen by the higher frame rates for the unbounded case).

For network conditions, using a Gigabit Ethernet connection for a remote client produces essentially equivalent performance to a client co-existing on the same machine. More worrying is the very low frame rates for the wireless 802.11g connection (4.16 fps for 1 client, and 1.96 for 2 clients). The low performance was most likely exacerbated by the fact that the wireless access point was being shared with regular network traffic in our lab. In any case, using wireless for video transmission over many clients is prohibitive for remote computer vision clients that require fast frame rates.

The second graph (Figure 6) illustrates the decline in frame rate for different video resolutions (640x480 versus 1024x768). Higher resolution at 1024x768 is often desirable for many vision algorithms since features are more easily discernible. Figure 6 allows one to determine how many modules can share the stream before frame rate drops to an unacceptable level of performance.

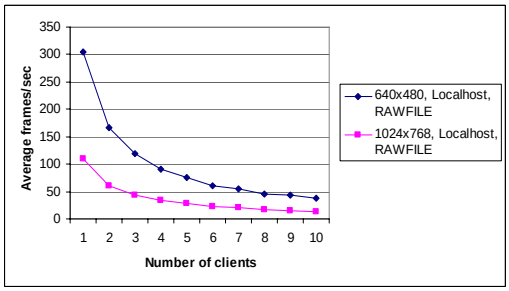


**Fig. 6: Plot comparing different image sizes.**

## IV. CONCLUSION AND DISCUSSION

The overall conclusion of this work is that delays due to our middleware are negligible compared to computational costs associated with actual processing within modules, provided we have a network with high enough bandwidth (gigabit in our case). The Cognitive Map is likely to be scalable to an increasing number of modules with negligible degradation of package delays.

In the process of performing these tests, several of our assumptions were proven incorrect. For example, we had hypothesized that if the Cognitive Map or modules resided on the same machine, average sample throughput would increase. In fact, for machines that are CPU-limited, performance can actually decrease. In several cases, early trials produced anomalous results that upon investigation, resulted in important code fixes that improved system performance, highlighting another important reason to conduct performance tests.

Current wireless technology poses a technical barrier to streaming important video or other high bandwidth data to remote modules. If we eventually desire to have autonomous, untethered humanoid robots moving around an environment, this problem will need to be addressed. One solution is to perform as much processing of the raw sensory

data on-board the robot and stream the resulting smaller-sized sample information produced (e.g., 6 floats = 24 bytes for object pose versus a 640x480x3 bytes = 921600 bytes for a frame of video). Another option is to wirelessly stream out a single channel of video to a remote machine that in turn can serve the video to multiple clients over a much faster wired network.

By knowing the effective frame rates and data sample rates one can achieve in a distributed system, robot developers can understand the bounds of performance that can be expected from the robot and to design the topologies of their distributed systems accordingly to maximize the bandwidth allocation of all modules in the system.

## REFERENCES


[1] Nilsson, N. "Shakey the robot. Technical Report 323", SRI, Menlo Park, CA, 1984.

[2] Honda Motor Co., Ltd., "Asimo year 2000 model", world.honda.com/ASIMO/technology/spec.html, 2000.

[3] Thórisson, K. R., H. Benko, A. Arnold, D. Abramov, A. Vaseekaran, "A constructionist methodology for interactive intelligences", A.I. Magazine, 2004, 25(4), pp. 70-90.

[4] Tanenbaum, A. S., M. van Steen,."Distributed Systems: Principles and Paradigms." New York: Prentice-Hall, 2002.

[5] Thórisson, K. R., T. List, C. Pennock, J. DiPirro , "Whiteboards: scheduling blackboards for semantic routing of messages & streams". In K. R. Thórisson, H. Vilhjalmsson & S. Marsela (Eds.), AAAI-05 Workshop on Modular Construction of Human-Like Intelligence, Pittsburgh, PA, July 10. AAAI Technical Report WS-05-08, 2005, pp. 8-15.

[6] B. Hayes-Roth, "A blackboard architecture for control." Artificial Intelligence, vol. 26, pp. 251-321, 1985.

[7] Ierusalimschy R, de Figueiredo LH and Celes W . "*Lua 5.1 Reference Manual*". Lua.org, publishers, 2006.

[8] D.Wagner and D.Schmalstieg,"Artoolkitplus for pose tracking on mobile devices,"in Computer Vision Winter Workshop 2007, February 6-8 2007, st. Lambrecht, Austria.