MIRO: MIDDLEWARE FOR CONTROLLING BIOLOGICALLY INSPIRED MOBILE ROBOTS

Rodrigo Cerón
CANNES Laboratory
Instituto Tecnológico Autónomo de México
Río Hondo # 1, Tizapán San Ángel
México, D. F., 01000
MÉXICO
rodrigo@cannes.itam.mx

Alfredo Weitzenfeld
Department of Computer Engineering
Instituto Tecnológico Autónomo de México
Río Hondo # 1, Tizapán San Ángel
México, D. F., 01000
MÉXICO
alfredo@itam.mx

Abstract

The study of animal behavior has inspired many robotic designs. In general, there are two ways to design a biologically inspired robot: (1) incorporate powerful hardware into the robot, so it can internally process all tasks; and, (2) have a distributed architecture delegating time-consuming processes to a remote computational server, allowing reduction of robot's hardware complexity. There are advantages and disadvantages to these two approaches. In particular, we have developed MIRO, a distributed architecture supported by a neural model to control biologically inspired mobile robots, with wireless communications. The goal within this architecture is to reduce robot complexity, size, cost and energy consumption by utilizing off-board computational resources. In this paper we discuss our approach to overcoming communication challenges arising from such a distributed wireless architecture. This includes the addition of a middleware layer to manage communication in a transparent way to the application layer.

1. Introduction

There are many ways of controlling autonomous robots; one of the most popular is based in a behavioral model, including those inspired in biological studies [1]. The study of sensory guided behavior in live animals has become of importance not only to scientists in neurosciences but also in robotics and distributed artificial intelligence. Researchers are using functional principles generated by these studies in constructing autonomous systems performing complex behaviors [2]. There have been experimental studies with amphibians (toads), insects (mantis), canines (domestic dogs) [3] and rodents (rats) [4]. These animals have multiple forms of sensory input, including in particular vision, of particular interest to our project.

These animals react both to fixed and mobile objects. Within the fixed group we find, for example, a water deposit or a rock. Within the mobile group stand out predators, preys as well as animals from their same species, including same or opposite gender. Objects usually influence the animal's way of acting which, in general, is focused on increasing its survival opportunities, for example, approaching a prey or a water deposit, while moving away from predators, in addition to object avoidance. Thus, it is necessary to bear in mind the time – space relationship between the animal and the object in order to understand and take correct decisions when designing a robot [2]. While there exists different biologically inspired behavioral robot implementations [5], these systems tend to lack adaptation and learning capabilities, such as those implemented by neural networks [6].

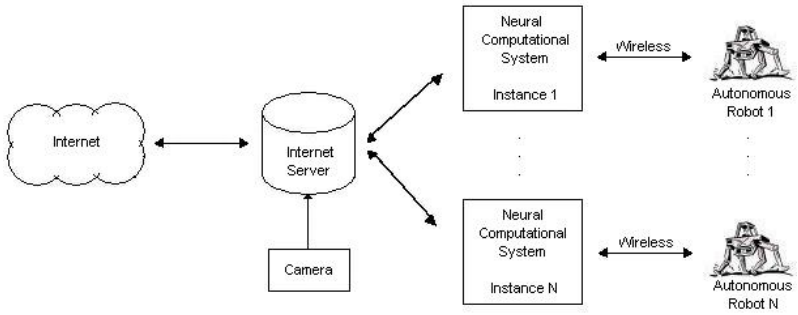


Figure 1. MIRO architecture consisting of multiple autonomous robots linked via wireless communication to their own instance of the distributed computational system.

2. MIRO System

MIRO (Mobile Internet Robot Laboratory) is a robotic architecture integrating fixed and mobile computational resources. With MIRO robots behave as sensors and actuators in a distributed architecture where each robot is connected to its own computational system instance (see figure 1). Processing is distributed between robot and remote computational system, where more computational resources, such as neural libraries, can be better exploited. In our current version, each robot incorporates a local camera, with additional aerial cameras for visualization of the actual experiment from different perspectives, something offering a number of advantages: (a) more robust event detection, validating crossed information (increases confidence) and (b) the information ranks can be estimated by triangulation [7].

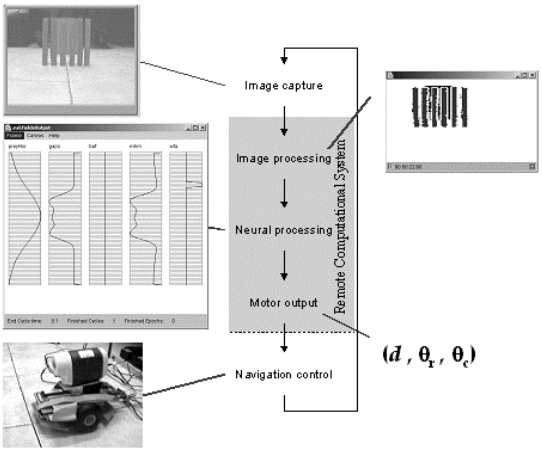


Figure 2. MIRO model computing cycle.

The robot computational process (See figure 2) starts when the robot sends information from its sensors to the remote computational system (See figure 3), mainly in the form of visual input. A remote vision module receives images from different cameras, doing a color based segmentation to identify perceived objects. This module also formats and sends the images to a remote web client for real time monitoring purposes. Given the information obtained from the received images, the distributed system determines the behavior. The movement module sends instructions back to the robot. This process continues indefinitely or until one specific task is executed.

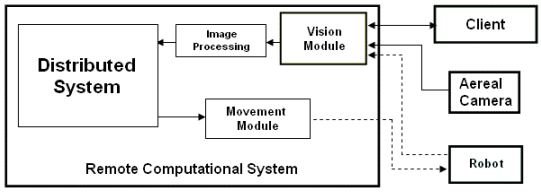


Figure 3. MIRO architecture. It shows vision and movement modules to interact with the robot, the cameras and remote web client.

3. Distributed Communication Challenges

The main system challenge with the MIRO architecture has to do with the always changing network and environment conditions (failures, disconnections, reduced connectivity). The most important issue in the design of camera-based networks is the managing of video information flow. The key for handling large amounts of information produced by cameras is optimizing available local computational resources, including interconnection protocols, network access with QoS and energy consumption. Depending on available bandwidth and user requirements the approach should involved transmission of (a) complete video streams; or (b) segmented images with different transmission rates. The knowledge of the actual network state is critical in evaluating the video transmission model to be used at a given time [7].

With the purpose of reducing the size of the information flow through the network, the multimedia objects can be converted or reduced before transmission. Some techniques such as scaling (redefining image size), color reduction, including reduction close to black and white, are used to reduce image size. These can be sent directly to the application or can be first formatted for transmission, all done in a dynamic fashion [8].

We have to deal with different scenarios in order to control the network conditions without affecting the application execution. In each scenario there are different issues that we need to control so that an experiment can be carried out successfully.

The different modeling levels contemplated in MIRO architecture are: Application Level, Data Links, Communications and Physical Level (See figure 4). Here we describe the situations that can be presented in each scenario and how can it be solved.

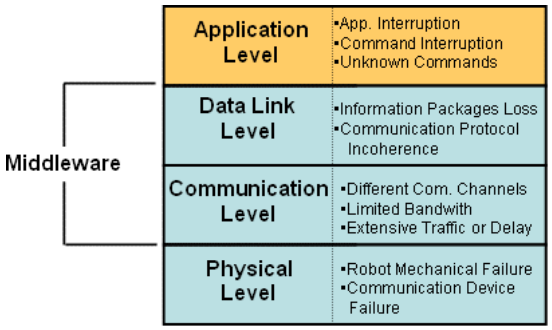


Figure 4. Multiple modeling levels in the MIRO architecture.

3.1 Application Level

The application level considers three different scenarios.

- *Application interruption.* This can be caused by an application exception, such as a lack of communication signal. In this case the application would not be able to resolve the problem. For example, in such situation, the robot middleware could include an emergency behavior directing the robot towards certain location while it receives a communication signal with new application instructions. As the application recovers robot experiments can proceed.
- *No I/O commands (command interruption) or unrecognized commands.* This occurs when incoming commands are not recognized by the robot. For example, when performing a biological behavior such as moving towards a prey, an interruption in received action commands from its server could cause the robot to continue with a previous behavior for some time while communication is reestablished. Additionally, the robot could execute an emergency behavior to direct itself towards a specific location while communication with the application is reestablished.

3.2 Data Link

The data link problems involve two possible scenarios:

- *Transmission package loss.* When either or both the application and the robot are receiving incomplete information. Possible solutions:
  a) middleware could verify if previous links have been established, otherwise, it could try using different channels or package destination address should be corrected.
  b) either the robot or the application have to take the correct decisions according to the received information, its quantity and type, avoiding a behavior interruption.

The second scenario appears when

- *Communication protocol incoherence.* A possible solution could be to:
  a) introduce protocol status verification during initialization, such as a series of test commands.

3.3 Communication Level

The most important considerations appear when we have:

- *Incompatible communication channels or frequencies.* In order to avoid different communication channels:
  a) scanning can be done over the frequencies or permitted channels, to quickly find a connection.
  b) robot can send the server a signal indicating it's turned on; this signal should always use the same channel so that the server can identify the robot is ready and send the channel or frequency in which they are going to be communicating.
- *Limited communication bandwidth, high communication traffic, or extensive communications delay.* This can be solved
  a) select a different channel with less traffic.
  b) use multimedia compression methods before the transmission.
  c) establish parameters to offer QoS according to traffic, modifications in quality, compression, size, colors, and even transmission rate would be done.

3.4 Physical Level

Physical risks could affect correct experiment execution, such as:

- *Robot mechanical failure or communication device failure.* It's important that the devices are in good condition for doing the experiment.
- *Device power failure.*
  a) the transmission rate and command reception can be lower, in order to use less power.
  b) If power consumption reaches a certain level, the completion probability of the experiment can be evaluated. See table 1 as an example of power consumption at different resolutions.

Frame Resolution	*Frames per Second*	*Avg. Power (W)*
320x240	15	5.86
340x160	15	5.60
160x120	15	5.30

Table 1. Average power consumption of the video capture using different configurations [9].

4. Middleware

In order to deal with the always changing network and environment conditions, an adapting robotic middleware has been designed managing communication between the robot and the computational system. The middleware needs to adapt to changing fixed and mobile network conditions in an application transparent way.

In general, middleware is a connectivity software consisting of a group of services to manage interaction between multiple processes executing in different devices in the network. The use of middleware has certain benefits for an application, among them modularly, separation of components, and hiding of network complexity to the application.

In order to take advantage of this architecture it is necessary to overcome restrictions on wireless transmission (bandwidth, communication failures and even total disconnections. In the presence of communication's fluctuations, it is the middleware's responsibility to determine how, when, and which information should be modified in order to better respond to such situations. The communication layer should be able to reconfigure by itself, adapting to changes on the communications environment. This aspect is very important in real time applications [10].

The middleware should also be able to add and remove communication service components during application execution without interrupting operation. It needs to distinguish and handle different kind of messages and communication protocols among different objects [11]. It should be integrated with the different hardware devices and methods for processing sensorial information and controlling output, including handling of computational vision and cognitive tasks [12].

Middleware solutions, such as CORBA (Common Object Request Broker Architecture) and RMI (Remote Method Invocation), have incorporated designs that separate functional aspects of a system from mechanisms used for its interconnection. Separating the functional aspects of a system from policies for interconnection control simplifies error detection and correction, as well as makes reutilization feasible [13].

With a middleware layer, implementation languages, operation systems and computer architectures are hidden to the application developer. This allows the application to operate without worrying about implementation details [14]. The middleware also hides information required to support QoS needs in real-time applications, such as delay, jitter and synchronization parameters.

Given the characteristics provided by middleware architecture and the need of MIRO for adaptation to changing communications conditions, we designed a middleware module connected to different modules and devices and at different interaction levels (See figure 5).

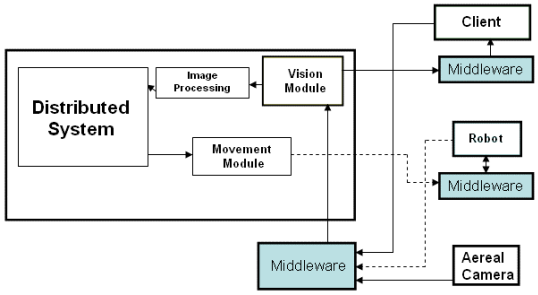


Figure 5. The adaptive MIRO architecture adds a middleware layer to overcome communication restrictions.

The middleware should control the link and communications level allowing manual and automatic monitoring and controlling of all the aspects all ready established.

Among the functionality required by the middleware, we find:

- Manual robot control channel or frequency selection.
- Change image characteristics.
- Traffic monitoring.
- Bandwidth monitoring.
- Monitoring the robots energy level.
- Executing a specific behavior.
- Autonomous execution.

The system manual control interface (See figure 6) it's divided in three main areas: communication parameters, monitoring and robot control. In the left side of the screen we can change the image resolution, the frame rate or even the communication frequency or channel. At the right, we see a graph displaying the networks traffic, in addition to available bandwidth and robot's energy level. Finally, at the bottom, we can see the Robot's Control Panel, were the user

can select the kind of movement and magnitude, and specific task or make the robot stop.

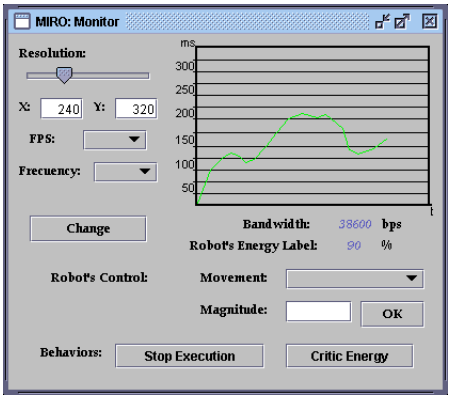
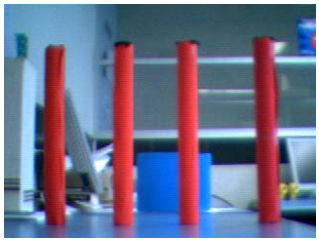


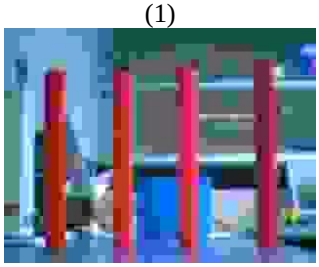
Figure 6. Manual control and monitor screen.

5. Experiments and Results

A middleware layer is being developed, to allow the dynamic images adaptation to changing network conditions. Parameters were established in order to determine traffic between the video capture server and its client through Internet. When a fixed or mobile remote client wants to watch images arriving to the video server, it synchronizes its clock with the server. It determines the compression level, quality, size and image transmission rate through the network. This result in a new set of parameters dynamically made according to the network state.



(1)



(2)

Figure 7. Different image resolution, changed by observing traffic in the network that communicates the remote web client with the computational server.

The work is divided into manual and automatic controlling and monitoring. Presently, tests have been made with manual parameter changes. It starts with the network conditions monitoring allowing manual verification (See figure 7). Once manual tests are finished the monitoring of current conditions of the data link and the communications level are done in order to automatically execute appropriate actions. This induce the dynamical adaptation changing network conditions to efficiently communicate the remote computational server with the robot and the remote web client.

In Table 2 we can see the delay in milliseconds for different transmission configurations and parameter combinations.

Resolution (%)	6 FPS	10 FPS	18 FPS	24 FPS
1.00	0.43	2.46	3.82	5.40
15.00	4.73	6.05	8.23	12.13
45.00	8.40	10.16	12.05	12.76
75.00	12.43	13.12	13.80	14.23
90.00	14.23	14.96	15.31	15.46

(a)

Resolution (%)	6 FPS	10 FPS	18 FPS	24 FPS
1.00	5.20	6.96	8.93	10.56
15.00	5.40	7.43	9.03	15.00
45.00	8.80	11.53	12.36	16.46
75.00	12.96	13.82	16.43	19.66
90.00	14.31	16.46	17.96	20.83

(b)

Table 2. Delays in milliseconds in the transmission from de computational server to the web client, using different configurations. Using (a) 160x140 and (b) 360x240 sizes.

If we consider the distance between the camera and the objects we can see that the closer the camera gets an object, the better the object resolution sent to the model. We tried different image compressions using different distances and we observed that the limit distance for the model to stop recognizing objects is 4 meters between the camera and the object. If it uses no compression, we can see objects some additional centimeters.

These results were obtained sending images to the model. The images were segmented in colors so it can recognize objects. When the objects were more than 4 meters from the camera, the system can't send information to the model because it don't recognize any object. If the model doesn't receive information, it sends instructions to robot so it can get closer to its objective.

6. Conclusions

Based on the obtained results, we can observe how the remote web client can receive images captured by the server with different parameter values. The performance metric is given by the visual perception of the final user. In addition, we consider another metric, that one provided by the model in the distributed system. We determine the resolution values by which the model stops recognizing objects. We calculated the time it takes to complete a full model cycle.

The relation between image resolution and object's size captured by the camera is given by the distance between the camera and the objects, so we are determining, based on this fact, how image resolution can affect object recognition depending on the distance to the objects.

In the future, we plan to migrate the system to digital cameras, to improve performance in object recognition. We are looking towards automatic adaptation in network information flow.

References

[1] Arkin, R., "Behavior – Based Robotics" MIT Press, Mayo 1998.

[2] Arkin, R., Cervantes – Perez, F., Weitzenfeld, A., "Ecological Robotics: A Schema – Theoretic Approach", AAAI Fall Symposium, 1996, Boston, Ma.

[3] Arkin, R., Fujita, M., Takagi, T., Hasegawa, R., "Ethological Modeling and Architecture for an Entertainment Robot", ICRA, 2001.

[4] Mataric, M., "Navigating with Rat Brain: A Neurobiologically – Inspired Model for Robot Spatial Representation", Proc. 1° International Conference on Simulation of Adaptive Behavior, 1990.

[5] Ali, K., Arkin, R., "Implementing Schema – Theoretic Models of Animal Behavior in Robotics Systems", Coimbra, 1998.

[6] Weitzenfeld, A., Arkin, R., Cervantes – Perez, F., Olivares, R., Corbacho, F., "A Neural Schema Architecture for Autonomous Robots", Proc. Of International Symposium on Robotics and Automation, 1998, Saltillo, Coahuila, Mexico.

[7] Obraczka, K., Manduchi, R., Garcia – Luna – Aceves, J. J., "Managing the Information Flow in Visual Sensor Networks", ISWPMC, 2002.

[8] Kreller, B., Sang – Bum Park, A., Meggers, J., Forsgren, G., Kovacs, E., Rosinus, M., "UMTS: A Middleware Architecture and Mobile API Approach", IEEE Personal Communications, April 1998.

[9] Weitzenfeld, A., Gutierrez-Nolasco, S., Venkatasubramanian, N., "Controlling Mobile Robots with Distributed Neuro – Biological Systems", Proc. AINS 2003, Menlo Park, California.

[10] Pal, P., Loyall, J, Schantz, R., Zinky, J., Shapiro, R., Megquier, J., "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration", ISORC, 2000.

[11] Gutierrez – Nolasco, S., Venkatasubramanian, N., "A Composable Reflective Communication Framework", Workshop on Reflective Middleware RM, 2000.

[12] Utz, H., Sablatnög, S., Enderle, S., Kraetzschmar, G., "Miro – Middleware for Mobile Robot Applications", IEEE Transactions on Robotics and Automation, June 2002. (Note: this is a different system to ours)

[13] Astley, M., Agha, G., "Customization and Composition of Distributed Objects: Middleware Abstractions for Policy Management", Proc. 6° International Symposium on the FSE, November 1998, Orlando, Florida.

[14] Loyall, J., Schantz, R., Zinky, J., Bakken, D., "Specifying and Measuring Quality of Service in Distributed Object Systems", IEEE, Proc. ISORC, April 1998, Kyoto, Japan.