# A Computational Framework for Distributed Robotic Systems Based on XML

*E. Mumolo*[1], *M. Mottica*[2], *E. Menegatti*[3], *E. Pagello*[3]

[1] DEEI, Universita' degli Studi di Trieste,
34127 Trieste, Italy, email: mumolo@univ.trieste.it
[2] A.I.B.S.Lab S.r.l., Via del Follatoio 12, Trieste, Italy
[3] DEI, Universita' degli Studi di Padova, Italy

## Abstract

In this paper we present a distributed computing system designed to provide computational resources to a mobile robot. The described system offers simple programming and debugging, easy modification of the data types coming from sensors, high computation power obtained by distributing the computation on remote nodes, portability and low overhead, making it well suited for implementing AI applications in the robotic domain. We developed an XML-based language, called XML-VM, that enables to transfer among remote machines not only the data to be processed, but also the algorithms by which they have to be processed. All the code implementing the system is written in ANSI C++ for efficiency and portability reasons.

We applied the system to the problem of image based localisation of a mobile robot using the properties of the Fourier transform of omnidirectional images. The image based localisation task is demanding in terms of computational power and memory requirements; therefore it is well suited to the described system. Experimental results for off-line robot localisation are reported in terms of absolute time and relative speed-up, confirming the high performances of the system.

**Keywords:** Mobile robot, localisation, omnidirectional images, XML, distributed computing, virtual machine.

## 1 Introduction

Autonomous behaviors are highly desirable in mobile robotic systems. To achieve this goal, sophisticated algorithms and many different sensors are needed. It must be considered, however, that processing of sensor data, for example processing of images, requires a high computational power. A powerful CPU means high power consumption, but for mobile robots the only available power is given by batteries which must also supply the motors and actuators. To overcome this problem we use the computational power existing in the environment i.e. desktop computers, powerful servers , etc. This computational power comes for free exploiting the unused CPU cycles of the heterogeneous computers available in the LAN where the robot is also connected. [1].

Besides the computational needs, robot systems have also special demands related to the complex interactions they have in real environments, and the complex sensors and actuators that robots use. This means that the robot has to handle different types of data and should be able to process these data with real-time algorithms. XML is becoming a standard method for the transmission of data through Internet as it is able to easily put structured data in an ACSII file. XML is also able to cope with different operating systems and hardware platforms.

Besides the task scheduling, the real-time operation is also greatly influenced by the language used to realize the distributed system. For instance, in C++ the garbage collection - which can introduce random variations in the computational time - can be controlled, as opposed to languages with automatic garbage collection like Java. Moreover, standard C++ cope with software portability.

The computing platforms used in the experiments may be heterogeneous, so the first problem we faced was the development of a virtual machine to assure portability among the various platforms. The first solution could have been to use the popular Java virtual machine or to use Corba interfaces. However, Java was rejected for the reason explained above and Corba interfaces are still quite complex both from an implementation and an execution point of view. We addressed the problem of portability using XML to describe both data and algorithms.

This paper is structured as follows. Section 1.2 describes the architecture of the system, while in Section 1.3 some issues related to distributed programming issues are reported. In Section 1.4 we summarize the XML-VM language, while in Section 1.5 we describe some information on the parsing and interpretation of a XML-VM program. In Section 1.6 deals with the image localization task. In Section 1.7 some experimental results are described while Section 1.8 deals with final remarks and conclusions. In the Appendix it is summarized an XML-VL code for image localization.

## 2 Design

The system used in this work is sketched in Fig.1. The mobile robot in the picture moves in a laboratory environment. Among other sensors, an omnidirectional camera for image acquisition and a microphone array for acoustic localization are available.

Consider the robot localization task. It is performed by matching the omnidirectional image taken in the current position with some reference images acquired in a previous phase. The robot then starts the matching phase by sending to the gateway the unknown image. Since an image takes about 48KB, this operation requires about 38 ms. The matching is then distributed on the PCs connected in the local network.

The gateway computer is called Root node in Fig.1. Every other computer is a computational node and it is configured as root or leaf in a logical tree structure, as in Fig.1. Each node has a reference to its higher level node and a

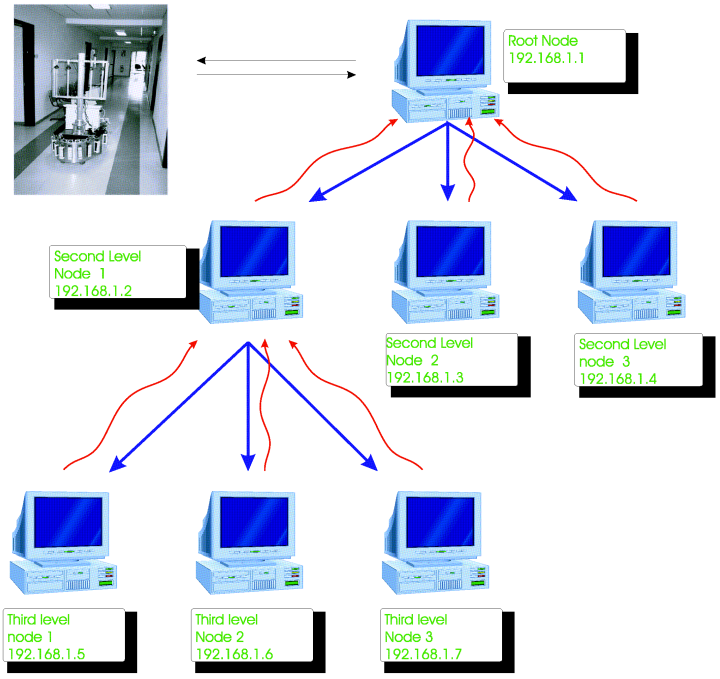



**Fig. 1.** System architecture

reference to a local table where a number of IP addresses is stored. The node configuration is defined in a configuration file read by every computational node during its initialization. During this phase, if a node is a leaf, it immediately notify itself to its higher level node, which stores the IP address of the leaf into its local table. If a node must distribute a computation task, it seeks its local table to get the IP address of the first available leaf. If the same node must distribute another computation task, then a second IP is identified in the table and so forth. If the local table of a node is empty, the node sends a request to its higher level node which returns the address of a node it takes from its local table. The address is temporarily stored in the local table of the node which made the request. When the remote task terminates, it is removed from the local table and becomes available in the local table of the higher level node. This recursive mechanism can proceed up to the gateway.

## 3 Distributed programming issues

Generally speaking, there are two main approaches for building a distributed programming system based on Java virtual machines [10]. One is to give the programmer an unique environment in which the threads are distributed on the different nodes by the operating system. This solution is quite complex to

develop, since many problems arise concerning both implementation and performance. Projects in this area include the IBM cluster VM for Java, the Kaffe virtual machine and the JDSM [9].

Other solutions are based on the development of communication mechanisms such as, for example, message passing. A typical approach is RMI (Remote Method Invocation). Other approaches are based on extensions of Java with parallel programming linguistic constructs. An example of the latter approach is JavaParty, developed at the University of Karlsruhe [5].

Orocos [11] focuses on object oriented components and patterns. CORBA inspired the distributed communications and is used in some work. Miro [12] is an object oriented layered client/server robot middleware system based on ACE [13] and the associated realtime CORBA ORB, TAO. MCA2 [14] focuses on reusable modules all with the same standardized interface.

Our solution, instead, is not based on Java nor Corba. Using XML as a framework to describe both data and algorithms we can provide a common base which different platforms can manage in different ways. One way may be to use transformation sheets, and another way may be to write an interpret using an appropriate programming language.

## 4 The XML Virtual Machine (XML-VM)

XML is a meta-markup language which allows to define a set of semantic tags which aims at describing a complex data structure by dividing it into sections. New tags can be created and defined by means of a Document Type Definition (DTD) document, which is the language vocabulary and syntax definition. The DTD allows to examine if an XML document is well-formed and to validate it. Since the structure of XML documents is described with a DTD, it is possible to translate the documents in different formats.

We implemented the communication between different machines using XML-RPC. A node acting as a client, which makes the RPC call, sends a request to a node acting as a server, including in the call the parameters needed for its execution. The server receives the request, identifies the procedure, execute it with the reported arguments, collects the response from the remote procedure and sends back to the client the answer. XML-RPC requests and responses are ASCII messages, transmitted through `http`.

The architecture of XML-VM is reported in fig.2

Data in XML-VM is stored in three sets of memory, which simulate a virtual disk, a local registry and a static registry, respectively. Control of execution flow is performed by means of IF-THEN-ELSE, WHILE and FOR constructs. Matrix and vector management tags include:

`<LOADIMAGE filename target node/>` loads an image into the virtual disk.
`<COMPAREIMAGE first second target/>` compares two images
`<COMPAREARCHIVE archive test distance target/>` compares the test image with all the images contained in an archive, yielding the name and the value of the minimum distance.

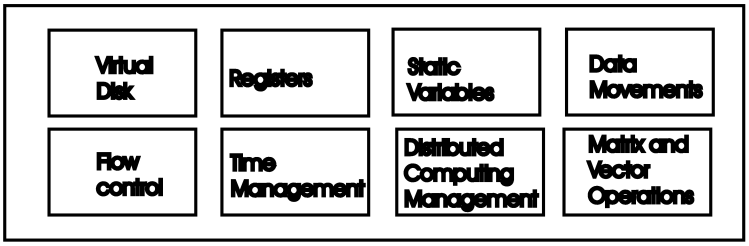



**Fig. 2.** Architecture of XML-VM

The distributed computing management tags implement the fork/join mechanism. The linguistic framework Fork/Join has been introduced by M.Conway [3] and J.Dennis [4]. Starting from the initial definition, many programming languages used the Fork/Join concept in several ways. The Fork/Join operations has been largely studied from a queueing point of view [2, **?**,**?**]. Fork generates a concurrent thread of execution, while the Join waits for termination; in this way it is possible to build concurrent processes. In our system, the fork tag has the following syntax:

```
<FORK id clone results>
    ...data and algorithms to be executed in
    the remote node, expressed in XML-VM...
</FORK>
```

and corresponds to the following actions: first, an available node is sought in the local table, then the code and data are sent to the remote node with XML-RPC. The Join tag has the following syntax: `<JOIN id/>` and the termination corresponds to the following operation: waits for the termination of the remote node and returns the results to the calling environment with a XML-RPC response.

## 5 Parsing and interpretation

A SAX parser treats an XML document in a linear fashion, working through the document from the beginning to end. In any case, the available SAX parsers deal with data streams because usually XML describes data structures which are parsed as they are downloaded from the net. In the case of the system described in this paper, however, a large part of the the XML documents are files already stored in the system. For this reason, a specific DOM parser has been developed in C++; the parser can process only the tags designed in XML-VM. Moreover, the parser is highly efficient from a computational point of view. From the XML document, the parser generates the execution tree which represents in a tree structure the sequence of instructions which compute the algorithm. Consider for example the algorithm reported in Fig.3 in XML-VM code.The tags and their

attributes are examined by the parser. The tags are associated to a numeric ID to accelerate the execution phase, and the tag's attributes are stored in a hash table for a fast access.

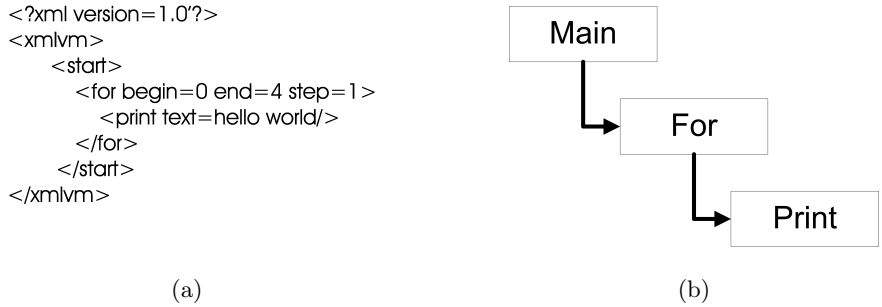



**Fig. 3.** (a) An algorithm expressed in XML-VM. (b)Execution tree of the algorithm.

¿From the code reported in Fig.3a the execution tree reported in Fig.**??**b is generated. The execution tree is finally analyzed by the interpreter, which calls the appropriate method to execute the tag. Some tags are associated to sub-tree of commands; for example the 'FOR' tag is associated to a list of commands to execute at each iteration and the 'IF' tag is associated to two lists of commands, one for each logical value of the condition.

At this point, we can summarize the tasks performed by a remote node.

- wait for an XML-RPC request
- get the XML-VM code from the request
- parse the XML-VM code and generate the execution tree
- start executing by visiting the execution tree and calling the method associated with the identifier assigned to the tag
- make a XML-RPC response
- wait for a new request

## 6 Omnidirectional image based localisation

The problem of robot localisation is one of the fundamental problems for autonomous mobile robots. In the literature there are several approaches to mobile robot localisation, in this work we used the image-based localisation approach. In the image-based localisation approach, the robot is driven around to collect a set of images of the environment, called the **reference images** at some distinguished positions, called the **reference positions**. The images are stored in the visual memory of the robot. In the navigation phase, the robot infers its location by comparing the current view of the environment, i.e. the **input image**, with the reference images stored in its memory. The reference image that is most

similar to the input image will give a topological localisation of the robot. In fact, the robot is closer to that reference position than to any other reference position.

If the environment is large, the number of reference image that the robot needs to store and to compare with the input image rapidly grows. We choose an omnidirectional camera to reduce the number of images necessary to represent the environment. Moreover, we do not store the omnidirectional images of the reference images, but the much more compact **Fourier signatures** (i.e. the Fourier transformations of each line of the images) associated to every reference image. As demonstrated in [15], the Fourier signature is a complete and compact representation of the omnidirectional image that, compared to other data reduction technique, has the great advantage of being invariant to image rotations, so the orientation of the robot does not need to be taken in consideration in the matching phase.

As described in previous works [17] [18], we developed an image-based Monte-Carlo localisation system in which the likelihood of the robot position is given by the dissimilarity between the input image and the reference images. The dissimilarity between the input image $O_i$ and the reference image $O_j$ is calculated as the $L_1$ norm

$$D(O_i, O_j) = \sum_{t=0}^{L-1} \sum_{k=0}^{M-1} |F_{it}(k) - F_{jt}(k)| \tag{1}$$

where $F_{it}$ is the module of the Fourier transform of the t-th row of image $O_i$ and k represent frequency. However, although the amount of data is reduced, the computations required by (1) can be very high if the number of reference images is high. Since the system presented in this paper offers a considerable computation power at a low price, we used instead the following distance measure:

$$D(O_i, O_j) = \sum_{t=0}^{L-1} (\underline{F_{it}} - \underline{F_{jt}}) W_t^{-1} (\underline{F_{it}} - \underline{F_{jt}})^T \tag{2}$$

where $\underline{F_{it}} = [F_{it}(0)F_{it}(1) \dots F_{it}(M-1)]$ and $\underline{F_{jt}} = [F_{jt}(0)F_{jt}(1) \dots F_{jt}(M-1)]$ represent respectively the unknown and reference images in the frequency domain. Moreover, in eq.(2), $W^{-1}$ is a positive-definite matrix that allows different weighting for individual reference images depending on their utility in identifiying the correct location. The common Euclidean distance sets $W$ (and $W^{-1}$) to be the identity matrix I, whereas the general Mahalanobis distance sets $W$ to be the autocovariance matrix corresponding to the rows of the reference images. We computed the autocovariance matrix associated to each reference image using a set of images generated by interpolation between couples of reference images.

## 7 Experimental results

The experimental environment were formed by a mobile robot and a number of computers connected by TCP/IP. The robot is connected through a wireless link

operating at 10Mbps to a gateway which gives access to a local network operating at 100 Mbps. In the local network, three Athlon based desktop computers running at 2 GHz and three Pentium 4 based desktop computers running at 2.4 GHz are connected. The Athlon computers run the Gentoo operating system and the Pentium computers run Red Hat Linux. During the experiments, the desktop computers were employed in text editing and word processing tasks. Finally, the mobile robot is equipped with a Pentium MMX running at 200 MHz.

The goal of the experimental analysis is to study the performance of the distributed computing system presented in this paper. In general, the efficiency of a distributed application is related to various factors, including: the network speed, the load of the remote nodes, the homogeneity of the used computers, the degree of parallelism of the algorithm, the protocol used for method distribution.

The computing time depends on the number of machines used in parallel for the elaboration of the algorithm. What it is expected is an hyperbolic behavior, since the T(n) function should be of the type 1/n, where n is the number of machines.

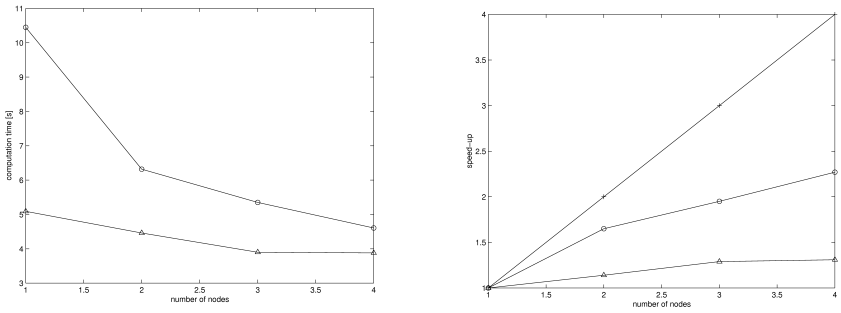
The experimental measurements are related to the comparison of the unknown image with 512 reference images using the Mahalanobis distance. Considering only the signal processing computational complexity, this means slightly more than 12 million of floating point multiply and add operations. Of course the numerical operations are only a part of the whole operations. The infrastructure introduce an overhead in terms of parsing, interpretation and communication of data and code which can be evaluated only with experimental measurements.

The algorithm equally distribute the distance computation among the available node: 512 comparison per node if the stystem uses one node, 256 comparisons per node using 2 nodes and 128 comparisons per node if four nodes are avaible for distribution.

To this goal, three types of experiments have been made. The first is related to the computational time obtained with a similar system but realized completely in Java as described in [**?**]. This experiment aims at measuring computational efficiency and the overhead of the system. The second test have been made with a similar system architecture but realized in RMI, which is a popular method for making distributed computations in Java. RMI uses serialization of data and code. The third experiments are performed with the system described in this paper and realized completely in C++.

In Fig.1.4a the absolute computation times required by the system described in [16] are reported for a number of remote nodes from 1 to 4. As shown in Fig.1.4a the time required by the XML-VM virtual machine written in Java with 1 node is 10.44 seconds, and drops to 4.6 seconds with 4 remote nodes, which means a speed-up of more than two times. It is worth noting that the theoretical speed-up for four remote nodes is four. In Fig.1.4a with triangular bullets the performances obtained with RMI are shown. Using RMI, the absolute time is quite lower, since it goes from about 5 seconds for one node to about 4 seconds for 4 nodes. It is important to note that RMI uses the new I/O class, which is available in the Java Enterprise 1.4 package since November 2003. However, the

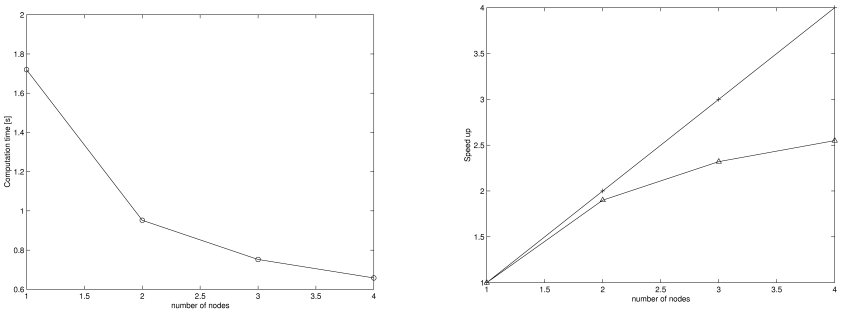
speed-up of RMI is much lower than that obtained with the previous structure. Finally, Fig.1.5a and 1.5b show the result obtained with the system described in this paper. In Fig.1.5a we can see that the absolute time with one node is much lower than all the other measurements made earlier, being about 1.7 seconds. Moreover, the system presents important decrements for increasing number of nodes. With four nodes, in fact, the absolute time is about 650 milliseconds.




(a)Results in Java (squares) and RMI (triangles)

(b)Speed-up in Java (squares) and RMI (triangles)

**Fig. 4.** (a) . (b)




(a)

(b)

**Fig. 5.** (a)Result with C++ (b) Speed-up with C++. The continuous line in the diagonal depicts theoretical performances

## 8 Conclusions and Acknowledgements

In this paper we dealt with the problem of designing and developing an efficient architecture based on XML for realizing a distributed image-based localization for mobile robots. By means of XML we realized an efficient framework for distributed programming; the distribution of data and algorithms is done by means of HTTP protocols and the code execution is performed with an interpreter written in C++.

From the experimental evaluation, it has been shown that the system described in this paper is able to distribute the comparison of an unknown image with 520 reference images using a Mahalanobis distance, in 0,658 seconds when taking advantage of distributed computation on four remote nodes. The same task performed with RMI under the same experimental conditions, requires 3.88 seconds. Thus, the proposed system performs six times faster on the same computational task. If XML-VM is realized in Java, then the same computation is performed more than seven times faster.

Moreover, the system is simple and easy to debug.

Many problems regarding robotics application using the framework described in this paper are still open. For example, the distribution of the workload, which is related to the choice of the nodes where the tasks are distributed, has not been considered yet. Another open aspect is the fault tolerance of the system.

The system described in this paper is currently being integrated in real mobile robotic system. Other activities under study concern the use of the system for other applications in the robotic field.

The authors wish to thank Prof. Hiroshi Ishiguro of Osaka University (Japan) for his advices and his support, and AIBSLab for making available to us the computers used in the experiments.

## Appendix

The XML document interpreted on the Root node dor the distributed distance computation is highlighted below.

```
<?xml version='1.0'?> <xmlvm> <start> ...registers and virtual
disk initialization... <loadimage filename="unknown.ppm"
target='8' mode='FF'/> <for from='0' to '4' step='2' target='1'>
    <load register='31' index='30'>
    <fork id='1' done='6' result='16r5'>
        <load register='7' index='6'/>
        <comparearchive archive='s1' ind_img='7' start='r20 end='r21'
                                          targetval='11' targetname='12'
typec='1'/>
        <store from='11' to='r5'/>
        <store from='12' to='r6'/>
   </fork>
...registers and static variables updating </for> <load
```

```
register='5' index='2'/> <load register='6' index='2'/> <inc
index='6' /> <for from='0' to '4' step='1' target='1'>
   <join id='1'>
   <load register='11' index='r5'>
   <load register='12' index='r6'>
   <if first='11' second='19' type='<'>
      <move source='11' target='19'>
      <move source='12' target='20'>
   <else>
      <if first='1' type='int' value='0' typec='=='>
           <move source='12' target='20'>
           <move source='12' target='20'>
      </if>
   </if>
...registers and static variables updating </for> </start>
</xmlvm>
```

## References


1. C. E. Catlett, J. Toole, "Testbeds: From Research to Infrastructure", in "The Grid: Blueprint for a New Computing Infrastructure," Ian Foster and Carl Kesselman, ed., Morgan Kaufmann, August 1998.
2. Ray Jinzhu Chen, "A Hybrid Solution of Fork/Join Synchronization in Parallel Queues", IEEE Transactions on Parallel and Distributed Systems 12 (8), August 2001
3. M Conway, "Multiprocessing system design", Proc. Of the AFIPS Fall Computer Conf., 1963
4. J.G.Dennis, E.C.Van Horn, "Programming semantics for multiprogramming computations", Communications of ACM, March 1966
5. http://www.ipd.uka.de/JavaParty/features.html
6. Doug Lea, "A Java Fork/Join Framework", ACM Java Grande 2000 Conference, June 3-5 2000
7. Y.C.Liu, H.G.Peros, "A Decomposition Procedure for the Analysis of a Closed Fork/Join Queuing System", IEEE Transactions on Computers, vol.40, n.3, march 1991
8. R.Nelson, A.N.Tantawi, "Approximate analysis of Fork/Join Synchronization in Parallel Queues", IEEE Transactions on Computers, vol37, n.6, June 1988
9. Y.Sohda, H.Nakada, S.Matsuoka, "Implementation of a Portable Software DSM in Java", ACM JavaGrande Int. Conference, June 2001
10. M.Surdeanu, D.Moldovan, "Design and Performance Analysis of a Distributed Java Virtual Machine", IEEE Transactions on Parallel and Distributed Systems, Vol.13, N.6, June 2002
11. http://www.orocos.org
12. H.utz, S.Sablatnog, S.Enderle, G.Kraetzschmar, "Miro-middleware for mobile robot applications", IEEE Trans. Robot. Automat., vol.18, n.4, pp.493- 497, Aug.2002
13. D.Schmidt, "Adaptive communication environment", http://www.cs.wustl.edu/schmidt/ACE.html
14. http://mca2.df.net

15. H. Ishiguro and S. Tsuji, "Image-based memory of environment", Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-96), pages 634–639, 1996.
16. E. Mumolo, E. Menegatti, E. Pagello, "Omnidirectional Image Based Localisation using an XML Virtual Machine for Distributed Computing", Proc. of 8th International Conference on Intelligent Autonomous Systems (IAS-8), 2004
17. E. Menegatti, M. Zoccarato, E. Pagello, and H. Ishiguro, "Hierarchical image-based localisation for mobile robots with monte-carlo localisation", in Proc. of European Conference on Mobile Robots (ECMR'03), pages 13–20, September 2003.
18. E. Menegatti, M. Zoccarato, E. Pagello, and H. Ishiguro, "Image-based monte-carlo localisation with omnidirectional images", in Robotics and Autonomous Systems, Elsevier, 2004.