Omnidirectional Image Based Localisation using an XML-based Language for Distributed Computing

Enzo Mumolo [1], Emanuele Menegatti [2], Enrico Pagello [2,3]

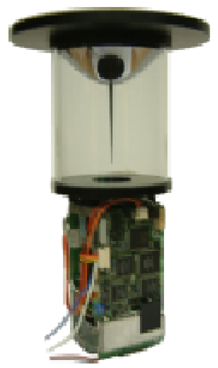
[1] Smartlab, Department of Electrotechnics, Electronics and Computer Science,
The University of Trieste, Italy
[2] Intelligent Autonomous Systems Laboratory (IAS-Lab)
Department of Information Engineering
The University of Padua, Italy
[3] *also with*: Institute ISIB of CNR, Padua, Italy

Abstract.
In this paper we present a distributed computing system that enables a mobile robot to exploit the computational resources available in the environment in which it moves. Our application is image based localisation of a mobile robot using the properties of the Fourier transform of omnidirectional images described in [12]. The image based localisation task is quite demanding in terms of computational power and memory requirements (i.e. the robot needs to store and to compare a large number of images); therefore it is well suited to apply the 'Grid Computing' paradigm. This work represents a preliminary effort toward the goal of building a computational GRID for robotics applications. We developed an XML-based language, called XML-VM, that enables to transfer among machines not only the data to be processed but also the algorithm by which they have to be processed. Experimental results of off-line robot localisation are reported.

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

1 Introduction

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.



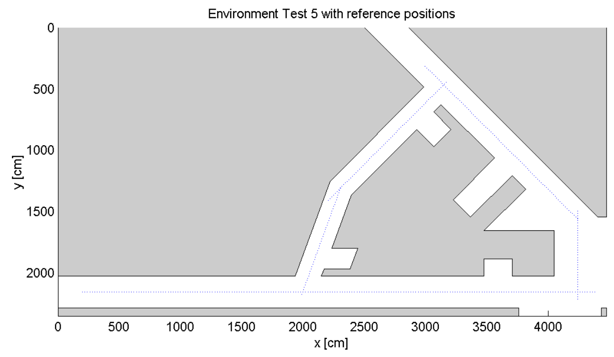


Figure 1: Left panel: omnidirectional camera. Right panel: the environment where the reference images were acquired.

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. People developed different approaches to reduce the burden of this large collection of images: dimension reduction exploiting properties of eigen-images [11], heading constrain of the image sensor [13], etc. We chose to use an omnidirectional camera to reduce the number of images necessary to represent the environment, see Fig.1. An omnidirectional image represents a complete view of the surroundings of the environment in one shot, so there is no need to rotate the camera. 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 [12], 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 [14] [15], 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 represents frequency. It has to be noted. however, that image matching can be performed also using computational simple space domain measurements without resorting to Fourier transform. There are two main reasons which explains why the norm depicted in (1) is based on the Fourier transform. First of all, the Fourier transform is computed only once, that is on the unknown image, as the reference images are already Fourier transformed. Secondly, in this way it is possible to use spectral distances, which could select image details which can improve the matching results. While this latter consideration is left to future work, in this paper we focussed on the distributed system.

The computations required by (1), in fact, can be very high if the number of reference images is high. For this reason we have designed a prototype of an XML-based language, called XML-VM, for the distributed computation of (1). XML-VM is suitable for representing, in a very general way, data structures and algorithms. It is important to note, in fact, that many distributed robotic tasks are based on the distribution of both data and algorithms. For image based localisation, for example, the data to distribute is an image in the FFT domain and the algorithm is a L_1 comparison. XML-VM provides a common set of rules for both data and algorithms which is very important for operation in an heterogeneous environment.

XML-VM uses the XML functionalities such as: self validation of document structure, default values for attributes, hyperlinks, entry replacement and internal referencing. Its main purpose is to distribute, among heterogeneous remote nodes, data and algorithms described using a generic model.

The XML-VM documents contain information about data structure and algorithms that must be executed on the remote node. The remote node can apply an XSLT or CSS transformation or it can directly interpret the document. In this work each remote note runs a Java program for the interpretation of an XML-VM document.

This paper is structured as follows. Section 2 describes the experimental environment of the system, while in Section 3 the architecture of the system is reported. In Section 4 we summarise the XML-VM language, while in Section 5 we describe some information on the parsing and interpretation of a XML-VM program. In Section 6 some experimental result are shown. In Section 7 some final remarks are reported.

2 A distributed system for robot localisation

The image-based Monte-Carlo localisation system described in previous works [14] [15] was implemented on a robot that incorporated all the necessary computational power, but for some application, like service robotics, we might be interested in building very cheap robot that can exploit the computational power existing in the environment: i.e. desktop computer, powerful server , etc. This computational power comes for free exploiting the unused CPU cycles of machines situated in the same environment of the robot or, if the system could access the Internet, also of remote servers. This is the computing paradigm known as ”GRID Computing” [1].

The computing environments for GRID computing are usually very heterogeneous from an hardware and software point of views; thus the first problem to be faced is the necessity of developing virtual machines to assure portability among the various platforms. One of the currently most popular virtual machines is Java. In fact, the Java virtual machine has been designed for running in different computing environments, from dedicated systems to general purpose machines. However, the Java virtual machine does not behave particularly well against attacks or intrusions. Moreover, the execution time of a Java process is not deterministic, mainly because of the memory management.

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].

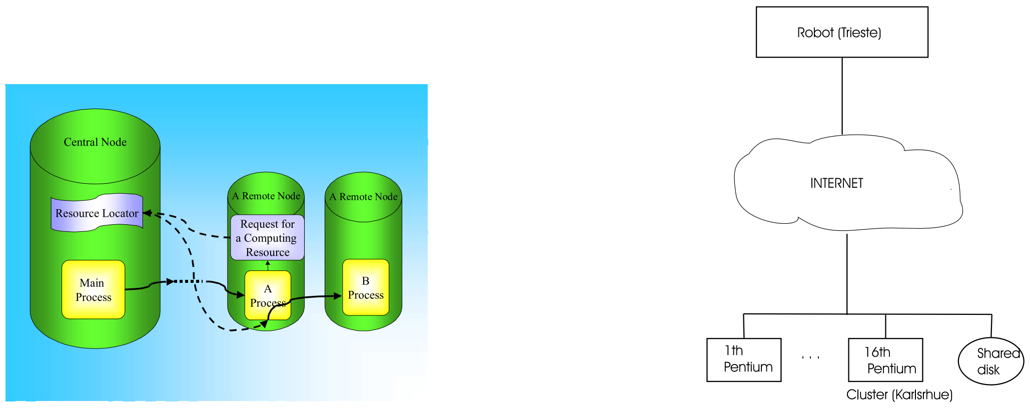


Figure 2: On the left: the name resolver mechanism. On the right: the system structure

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].

In conclusion, it emerges a need for a system which allows a distribution of data and algorithms among a large number of heterogeneous machines. 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 can be to use transformation sheets, and another way is to write an interpret. In this work an XML-VM interpret has been written in Java, and the distribution mechanism is based on XML-RPC packages.

Remark1

The language XML-VM describes data and algorithms for distribution on remote nodes of a distributed system; the node which receives the XML-VM document must interpret it to execute the algorithm.

Remark2

XML-VM is a virtual machine for robotics applications since it provides a common set of rules for description of data and algorithms. Its aim is to execute distributed algorithms on heterogeneous platforms

Moreover, writing the interpret in C language, the problems related to non determinism of the execution can be avoided.

3 System architecture

The distributed system described in this paper and described in Fig.2 is structured in a peer-to-peer style, limiting the tasks of the robot of the system mainly to the activation of the remote methods, the collection of the result and the measurements of the performances of the system.

The robot does not know nearly anything about what is happening in the remote nodes, where the computation effectively takes place. Each algorithm decides if and when to call other remote nodes and the method to execute. On the robot a particular daemon is running, called ’name resolving daemon’, which knows what remote nodes are available. When a generic node ’A’ needs to fork a procedure on a remote node, it calls the robot for determining the address of an available remote node. At this point, node ’A’ contacts directly the remote node for sending information on the distribution of the XML-VM code and its remote execution; this procedure is executed each time a remote call is needed. The remote node is therefore informed about what XML document it has to download and interpret, and downloads the related code from the robot, which acts as a Web Server.

Clearly, node ’A’ must join the conclusion of the remote call by waiting for the return of results. This procedure is implemented through the use of the linguistic framework Fork/Join[1]

Our experimental results were obtained with a cluster of 16 computers made available to this work from the Parallel Programming Lab of the University of Karlsruhe, Germany. Each machine is equipped with Pentium III bi-processors at 800 MHz, with 512 Mbyte of memory and running a Linux Operating System; the machines which make up the cluster shares a disk.

In addition to the cluster described above, a machine located in the Trieste Lab has been used to represent the robot. The machine used a Pentium II processor at 400 MHz and runs a Windows 2000 Operating System. The configuration of the system is described in Fig.2, right panel.

4 The language XML-VM

In this Section we will summarise the main characteristics of the XML-VM language. First of all, it is worth noting that two sets of memory are generated, declared in Java as Array of Object, which simulate registry and a virtual disk available to the virtual machine. The registry is constituted by 32 cells, while the virtual disk is constituted by 10000 blocks of data.

All the data related operations take place on the registers. There are no variables, and every operation must be performed specifying the involving register’s cells. The storage of data on the virtual disk is performed exclusively through the STORE instruction. Instead, the LOAD instruction is the only instruction that allows to copy the content of the cells of the virtual disk into the registers. Ten different data types are implemented in the language; nine of them follow the Java data types: int, long, short, byte, float, double, boolean, char and string, while the tenth data type, defined in XML-VM as ”index”, represents a pointer to another data cell in the virtual disk or in a register. The index data type can be used by the load, store and procedure call operations; moreover, this type is fundamental for parameter passing to the remote methods. The syntax of the language exploits the use of tags attributes as integrating parts of the instructions.

The tags are divided into arithmetic, data movement and calling tasks. Other tags are devoted to specialised functions, as highlighted below. There is a number of FFT tags, such as <fft> and <fftmod>. For example:

- <FFT type=”–” from=”r1” to=”r2”/>

This tag refers to a data structure described in the ‘type’ attribute. The actual data are contained in the virtual disc starting from the location expressed in ‘r1’ and the computed spectra

[1]The linguistic framework Fork/Join has been introduced by M.Conway [3] and J.Dennis [4]

is stored in the virtual disc starting from the location indicated in ‘r2’.

- <STOREIMAGE type=”—” index=”x” from=”y”/>

This tag, when interpreted, performs the following operations: the image referring to the data structure indicated in the ‘type’ attribute is written **synchronously** (in the sense that processes on the remote node wait for the writing completion) on the remote disk. The name of the remote file is pointed by to the ”x” cell and the pointer to the data to be written is stored in the ”y” cell. On the remote side, a daemon server manages this type of requests.

5 Parsing and interpretation

One of the most complete and used XML parser is the Apache Xerces XML Parser. Xerces supports SAX 1.0 and 2.0; SAX stands for Simple Api for XML. Once we completed the first version of XML-VM virtual machine, embedding the XML-RPC protocol for communication between remote nodes, we noticed a decrease of performance due to the slowness of the XML parser. In order to overcome this overhead we decided to use the same parser chosen by the developers of Apache XML-RPC, that is MINML.

The parser performs a complete analysis of the XML-VM document, expanding all the tags, attributes, values and calls the appropriate methods coded in Java. The high performance obtained with MinML is mainly due to the fact that the parser does not process the Data Type Definition (DTD). The interpreter of XML-VM language has been written in Java for portability reasons; the interpreter executes the action associated to the XML tags as they are analysed by the parser.

6 Experimental results

The goal of the experimental analysis is to study the performance of the Grid 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 machines which participate to the Grid, the degree of parallelism of the algorithm, the protocol used for method distribution.

The XML document interpreted on the robot is highlighted below.

```
<?xml version='1.0'?>
<xmlvm>
<start>
<!-- Initialization tags -->
<structure name="ppm">
  <array name="pixel" description"ppm pixel">
    <format>
      <integer width="3"/>
    </format>
    <axis name="x" axisid="pixel">
       <axisvalues count="3" start="1" step="1" />
    </axis>
  </array>
  <array name="img" description="ppm picture">
  <format>
     <integer width="5"/>
  </format>
  <axis name="x" axisid="ximg">
     <axispoints element="pixel"/>
     <axisvalues count="512" start="1" step="1" />
  </axis>
  <axis name="y" axisid="yimg">
     <axispoints element="pixel"/>
     <axisvalues count="512" start="1" step="1" />
  </axis>
<!-- other data structure definitions follow -->
</structure>

<loadimage type="img" file="acq_image" index="10" to="5" />
<load register="r1" index="5"/>
<fftmod type="fftimg" from="r1" to "r2"/>
<storeimage type="fftimg" index="11" from="r2"/>
<fork id="N01" file="http://140.105.50.110:80/Texel.xml"
   name=Texel" to "m8[m7]" />
. . .
<fork id="N16" file="http://140.105.50.110:80/Texel.xml"
   name=Texel" to "m8[m7]" />
. . .
<join topointed="m5[m4]"/>
<!-- next tags extract from the output cells the best images
<!-- with their related distances -->
<!-- other tags sort the images with increasing distances -->

</xmlvm>
```

The role of the Fork tag is to start a section of the algorithm, in parallel to other sections, on a remote node whose address is chosen by the name resolver. In this case the ‘fork’ tag is used to distribute on the remote node identified to the ‘id’ attribute the XMLVM document called ‘Texel.xml’. Texel.xml contain a description of the data structure and a description of the algorithm to compare the unknown image with the reference images.

The acquired image is first represented as Fourier signatures with the ‘fftmod’ tag and then is transmitted to the remote cluster where it will appear as a shared file. The writing is synchronous, so that the image comparisons which are performed thereafter can be started when the image is stable. The FORK tags distribute the comparison tasks on the remote nodes. The cluster nodes access, through the shared disk, all the reference images previously acquired and the unknown image, all of them are given as Fourier signatures. The method ”Texel.xml” performs a small number of comparisons, from 3 to 9. Since sixteen remote methods are distributed, this corresponds to 48 to 144 image comparisons.

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

In Fig.3, left panel, the computation time with 3, 6 and 9 image per remote method ”Texel.xml” is indicated with points, triangles and squares respectively. As it can be observed,

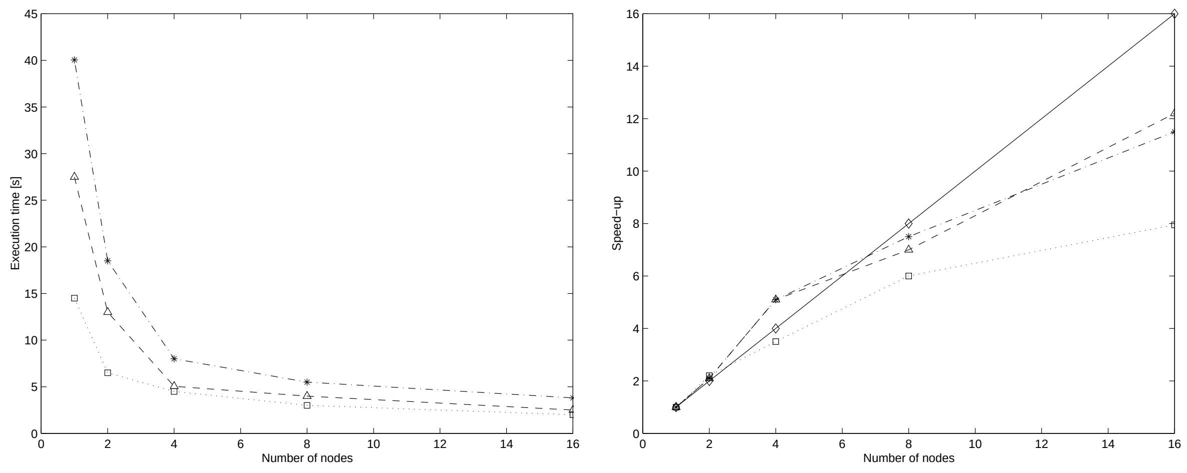


Figure 3: (Left) Absolute time versus number of nodes for the comparison of 48, 96 and 144 images. (Right) Speedups versus the number of remote nodes. The curves represented with dots, triangles and squares are related to 48, 96 and 144 total comparisons respectively.

the experimental measurements of Fig.3 confirm the expected theoretical results: the absolute time versus the number of machines follows a hyperbolic curve. With 16 nodes, the comparison absolute time is approximately 2, 2.5 and 4 seconds for 48, 96 and 144 comparisons respectively. In Fig.3, right panel, the speedups related to the XML system are shown.

Again, with dots, triangles and squares the results pertaining to 48, 96 and 144 total comparisons respectively are represented. The last picture gives rise to several comments. First of all the results for a different number of comparisons, mainly for 96 and 144 images, are very irregular. Second, it appears that in same case, for example with four nodes, the speedup is greater than the ideal curve. The irregularity of the results are due to the different loading factor of all the nodes but also for another reason, that is for caching. In fact, in the first run the first machine have to load all the required reference images from the disk, thus requiring a high number of disk I/O. If the number of nodes increases, some images are already loaded in the system and therefore the computation time is reduced, leading to a speed-up greater than the ideal one, which is based on the first run, which comprises all the I/O requests. As a second remark, slower machines in the GRID quickly lead to a performance decrement of the whole system down to the speed of the slower machines.

7 Conclusions and Acknowledgements

In this paper we dealt with the problem of designing and developing an efficient architecture based on XML for realising a distributed image-based localisation for mobile robots based on the Grid approach. By means of XML it is possible to realise an efficient Grid system; the distribution of data and algorithms is done by means of HTTP protocols and the code execution is performed with an interpreter written in JAVA. The scalability of the system is realised using a name resolving daemon.

Many problems of the GRID for robotics application 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. Another open aspect is the fault tolerance of the system. Ongoing activities are the development of portable C++ libraries for XML-RPC, parsing and XML-VM interpretation.

8 Acknowledgements

The authors wish to thank: Dr. Bernhard Haumacher of the Parallel Programming Lab of the University of Karlsruhe for making available to us the cluster of machines used for implementing the system. We wish to thank also Prof. Hiroshi Ishiguro of Osaka University (Japan) for the support to the first part of this work.

This research has been partially supported by: the Italian Ministry for the Education, the University, and Research (MIUR), the Italian National Council of Research (CNR), The University of Padua, and the University of Trieste.

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 Ssoftware DSM in Java”, ACM Java-Grande 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] H. Aihara, N. Iwasa, N. Yokoya, and H. Takemura. Memory-based self-localisation using omnidirectional images. In A. K. Jain, S. Venkatesh, and B. C. Lovell, editors, *Proc. of the 14th International Conference on Pattern Recognition*, volume vol. I, pages 1799–1803, 1998.

[12] H. Ishiguro and S. Tsuji. Image-based memory of environment. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-96)*, pages 634–639, 1996.

[13] B. Kröse, N. Vlassis, R. Bunschoten, and Y. Motomura. A probabilistic model for appareance-based robot localization. *Image and Vision Computing*, vol. 19(6):pp. 381–391, April 2001.

[14] 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)*, page 13–20, September 2003.

[15] E. Menegatti, M. Zoccarato, E. Pagello, and H. Ishiguro. Image-based localisation monte-carlo localisation without a map. In *AI*IA 2003: Advances in Artificial Intelligence : 8th Congress of the Italian Association for Artificial Intelligence, Proceedings*, page 423–435, September 2003.